

# Calibrator

1.2.1

Generated by Doxygen 1.8.9.1

Fri Nov 27 2015 08:04:37



# 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	Calibrate Struct Reference . . . . .	13
4.1.1	Detailed Description . . . . .	15
4.1.2	Field Documentation . . . . .	15
4.1.2.1	thread_gradient . . . . .	15
4.2	Experiment Struct Reference . . . . .	15
4.2.1	Detailed Description . . . . .	16
4.3	Input Struct Reference . . . . .	16
4.3.1	Detailed Description . . . . .	17
4.4	Options Struct Reference . . . . .	17
4.4.1	Detailed Description . . . . .	18
4.5	ParallelData Struct Reference . . . . .	18
4.5.1	Detailed Description . . . . .	18
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 . . . . .	24
<b>5</b>	<b>File Documentation</b>	<b>25</b>
5.1	config.h File Reference . . . . .	25
5.1.1	Detailed Description . . . . .	27
5.2	config.h . . . . .	28
5.3	interface.h File Reference . . . . .	29

5.3.1	Detailed Description	31
5.3.2	Function Documentation	31
5.3.2.1	cores_number	31
5.3.2.2	input_save	31
5.3.2.3	window_get_algorithm	33
5.3.2.4	window_get_gradient	34
5.3.2.5	window_read	34
5.3.2.6	window_save	36
5.3.2.7	window_template_experiment	37
5.4	interface.h	38
5.5	mpcotool.c File Reference	40
5.5.1	Detailed Description	45
5.5.2	Function Documentation	45
5.5.2.1	calibrate_best	45
5.5.2.2	calibrate_best_gradient	46
5.5.2.3	calibrate_estimate_gradient_coordinates	46
5.5.2.4	calibrate_estimate_gradient_random	47
5.5.2.5	calibrate_genetic_objective	47
5.5.2.6	calibrate_gradient_sequential	48
5.5.2.7	calibrate_gradient_thread	49
5.5.2.8	calibrate_input	50
5.5.2.9	calibrate_merge	51
5.5.2.10	calibrate_parse	52
5.5.2.11	calibrate_save_variables	53
5.5.2.12	calibrate_step_gradient	54
5.5.2.13	calibrate_thread	55
5.5.2.14	cores_number	56
5.5.2.15	input_open	56
5.5.2.16	input_save	65
5.5.2.17	input_save_gradient	67
5.5.2.18	main	68
5.5.2.19	show_error	70
5.5.2.20	show_message	71
5.5.2.21	window_get_algorithm	71
5.5.2.22	window_get_gradient	72
5.5.2.23	window_read	72
5.5.2.24	window_save	74
5.5.2.25	window_template_experiment	75
5.5.2.26	xml_node_get_float	76
5.5.2.27	xml_node_get_int	76

5.5.2.28	<a href="#">xml_node_get_uint</a>	77
5.5.2.29	<a href="#">xml_node_set_float</a>	77
5.5.2.30	<a href="#">xml_node_set_int</a>	78
5.5.2.31	<a href="#">xml_node_set_uint</a>	78
5.5.3	Variable Documentation	78
5.5.3.1	<a href="#">format</a>	78
5.5.3.2	<a href="#">precision</a>	79
5.5.3.3	<a href="#">template</a>	79
5.6	<a href="#">mpcotool.c</a>	79
5.7	<a href="#">mpcotool.h</a> File Reference	131
5.7.1	Detailed Description	133
5.7.2	Enumeration Type Documentation	133
5.7.2.1	<a href="#">Algorithm</a>	133
5.7.2.2	<a href="#">GradientMethod</a>	133
5.7.3	Function Documentation	134
5.7.3.1	<a href="#">calibrate_best</a>	134
5.7.3.2	<a href="#">calibrate_best_gradient</a>	134
5.7.3.3	<a href="#">calibrate_genetic_objective</a>	135
5.7.3.4	<a href="#">calibrate_gradient_thread</a>	135
5.7.3.5	<a href="#">calibrate_input</a>	136
5.7.3.6	<a href="#">calibrate_merge</a>	138
5.7.3.7	<a href="#">calibrate_parse</a>	138
5.7.3.8	<a href="#">calibrate_save_variables</a>	140
5.7.3.9	<a href="#">calibrate_step_gradient</a>	140
5.7.3.10	<a href="#">calibrate_thread</a>	142
5.7.3.11	<a href="#">input_open</a>	143
5.7.3.12	<a href="#">show_error</a>	151
5.7.3.13	<a href="#">show_message</a>	152
5.7.3.14	<a href="#">xml_node_get_float</a>	152
5.7.3.15	<a href="#">xml_node_get_int</a>	153
5.7.3.16	<a href="#">xml_node_get_uint</a>	153
5.7.3.17	<a href="#">xml_node_set_float</a>	154
5.7.3.18	<a href="#">xml_node_set_int</a>	154
5.7.3.19	<a href="#">xml_node_set_uint</a>	154
5.8	<a href="#">mpcotool.h</a>	155
<b>Index</b>		<b>159</b>



# Chapter 1

## MPCOTool

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

### VERSIONS

- 1.2.1: Stable and recommended version.
- 1.3.7: 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:

- 1.2.1/configure.ac: configure generator.
- 1.2.1/Makefile.in: Makefile generator.
- 1.2.1/config.h.in: config header generator.
- 1.2.1/mpcotool.c: main source code.
- 1.2.1/mpcotool.h: main header code.
- 1.2.1/interface.h: interface header code.
- 1.2.1/build: script to build all.
- 1.2.1/logo.png: logo figure.
- 1.2.1/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/1.2.1
```

```
$ ln -s ../../genetic/0.6.1 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/1.2.1):

```
$ cd ../tests/test2
$ ln -s ../../genetic/0.6.1 genetic
$ cd ../test3
$ ln -s ../../genetic/0.6.1 genetic
$ cd ../test4
$ ln -s ../../genetic/0.6.1 genetic
```

2. Build all tests doing in the same terminal:

```
$ cd ../1.2.1
$ 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_name"
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="#">Calibrate</a>	Struct to define the calibration data . . . . .	<a href="#">13</a>
<a href="#">Experiment</a>	Struct to define experiment data . . . . .	<a href="#">15</a>
<a href="#">Input</a>	Struct to define the calibration input file . . . . .	<a href="#">16</a>
<a href="#">Options</a>	Struct to define the options dialog . . . . .	<a href="#">17</a>
<a href="#">ParallelData</a>	Struct to pass to the GThreads parallelized function . . . . .	<a href="#">18</a>
<a href="#">Running</a>	Struct to define the running dialog . . . . .	<a href="#">19</a>
<a href="#">Variable</a>	Struct to define variable data . . . . .	<a href="#">19</a>
<a href="#">Window</a>	Struct to define the main window . . . . .	<a href="#">20</a>





## 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 . . . . .	25
<a href="#">interface.h</a>	Header file of the interface . . . . .	29
<a href="#">mpcotool.c</a>	Source file of the mpcotool . . . . .	40
<a href="#">mpcotool.h</a>	Header file of the mpcotool . . . . .	131



## Chapter 4

# Data Structure Documentation

### 4.1 Calibrate Struct Reference

Struct to define the calibration data.

```
#include <mpcotool.h>
```

#### Data Fields

- `GMappedFile ** file [MAX_NINPUTS]`  
*Matrix of input template files.*
- `char ** template [MAX_NINPUTS]`  
*Matrix of template names of input 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 gradient based method step sizes.*  
double \* [gradient](#)
- *Vector of gradient 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 \* [thread](#)
- *Array of simulation numbers to calculate on the thread.*  
unsigned int \* [thread\\_gradient](#)
- *Array of best simulation numbers.*  
unsigned int \* [simulation\\_best](#)
- *Algorithm tolerance.*  
double [tolerance](#)
- *Mutation probability.*  
double [mutation\\_ratio](#)
- *Reproduction probability.*  
double [reproduction\\_ratio](#)
- *Adaptation probability.*  
double [adaptation\\_ratio](#)
- *Relaxation parameter.*  
double [relaxation](#)
- *Calculation time.*  
double [calculation\\_time](#)
- *Seed of the pseudo-random numbers generator.*  
unsigned long int [seed](#)
- *Variables number.*  
unsigned int [nvariables](#)
- *Experiments number.*  
unsigned int [nexperiments](#)
- *Number of input files to the simulator.*  
unsigned int [ninputs](#)
- *Simulations number per experiment.*  
unsigned int [nsimulations](#)
- *Method to estimate the gradient.*  
unsigned int [gradient\\_method](#)
- *Method to estimate the gradient.*  
unsigned int [nsteps](#)

- unsigned int [nestimates](#)  
*Number of steps for the gradient based method.*
- unsigned int [algorithm](#)  
*Number of simulations to estimate the gradient.*
- unsigned int [nstart](#)  
*Algorithm type.*
- unsigned int [nend](#)  
*Beginning simulation number of the task.*
- unsigned int [nstart\\_gradient](#)  
*Ending simulation number of the task.*
- unsigned int [nend\\_gradient](#)  
*Beginning simulation number of the task for the gradient based method.*
- unsigned int [niterations](#)  
*Ending simulation number of the task for the gradient based method.*
- unsigned int [nbest](#)  
*Number of algorithm iterations.*
- unsigned int [nsaveds](#)  
*Number of best simulations.*
- int [mpi\\_rank](#)  
*Number of saved simulations.*
- int [mpi\\_rank](#)  
*Number of MPI task.*

#### 4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 111 of file [mpcotool.h](#).

#### 4.1.2 Field Documentation

##### 4.1.2.1 unsigned int\* Calibrate::thread\_gradient

Array of simulation numbers to calculate on the thread for the gradient based method.

Definition at line 144 of file [mpcotool.h](#).

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

- [mpcotool.h](#)

## 4.2 Experiment Struct Reference

Struct to define experiment data.

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

#### Data Fields

- char \* [template](#) [MAX\_NINPUTS]  
*Array of input template names.*
- char \* [name](#)  
*File name.*
- double [weight](#)  
*Weight to calculate the objective function value.*

### 4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file [interface.h](#).

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

- [interface.h](#)

## 4.3 Input Struct Reference

Struct to define the calibration input file.

```
#include <mpcotool.h>
```

### Data Fields

- char \*\* [template](#) [[MAX\\_NINPUTS](#)]  
*Matrix of template names of input files.*
- char \*\* [experiment](#)  
*Array of experimental data file names.*
- char \*\* [label](#)  
*Array of variable names.*
- 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 \* [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 \* [weight](#)  
*Array of the experiment weights.*
- double \* [step](#)  
*Array of gradient based method step sizes.*
- unsigned int \* [precision](#)  
*Array of variable precisions.*
- unsigned int \* [nsweeps](#)  
*Array of sweeps of the sweep algorithm.*

- unsigned int \* [nbits](#)  
*Array of bits numbers of the genetic algorithm.*
- double [tolerance](#)  
*Algorithm tolerance.*
- double [mutation\\_ratio](#)  
*Mutation probability.*
- double [reproduction\\_ratio](#)  
*Reproduction probability.*
- double [adaptation\\_ratio](#)  
*Adaptation probability.*
- double [relaxation](#)  
*Relaxation parameter.*
- 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 [algorithm](#)  
*Algorithm type.*
- unsigned int [nsteps](#)  
*Number of steps to do the gradient based method.*
- unsigned int [gradient\\_method](#)  
*Method to estimate the gradient.*
- unsigned int [nestimates](#)  
*Number of simulations to estimate the gradient.*
- unsigned int [niterations](#)  
*Number of algorithm iterations.*
- unsigned int [nbest](#)  
*Number of best simulations.*

#### 4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 64 of file [mpcotoool.h](#).

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

- [mpcotoool.h](#)

## 4.4 Options Struct Reference

Struct to define the options dialog.

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

## Data Fields

- GtkWidget \* [dialog](#)  
*Main GtkWidget.*
- GtkWidget \* [grid](#)  
*Main GtkWidget.*
- GtkWidget \* [label\\_seed](#)  
*Pseudo-random numbers generator seed GtkWidget.*
- GtkWidget \* [spin\\_seed](#)  
*Pseudo-random numbers generator seed GtkWidget.*
- GtkWidget \* [label\\_threads](#)  
*Threads number GtkWidget.*
- GtkWidget \* [spin\\_threads](#)  
*Threads number GtkWidget.*
- GtkWidget \* [label\\_gradient](#)  
*Gradient threads number GtkWidget.*
- GtkWidget \* [spin\\_gradient](#)  
*Gradient threads number GtkWidget.*

### 4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 74 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 <mpcotool.h>
```

## Data Fields

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

### 4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 184 of file [mpcotool.h](#).

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

- [mpcotool.h](#)



## 4.6 Running Struct Reference

Struct to define the running dialog.

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

### Data Fields

- `GtkDialog * dialog`  
*Main GtkDialog.*
- `GtkLabel * label`  
*Label GtkLabel.*

### 4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 92 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 variable data.

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

### Data Fields

- `char * label`  
*Variable label.*
- `double rangemin`  
*Minimum value.*
- `double rangemax`  
*Maximum value.*
- `double rangeminabs`  
*Minimum allowed value.*
- `double rangemaxabs`  
*Maximum allowed value.*
- `unsigned int precision`  
*Precision digits.*
- `unsigned int nsweeps`  
*Sweeps number of the sweep algorithm.*
- `unsigned int nbits`  
*Bits number of the genetic algorithm.*

### 4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file [interface.h](#).

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

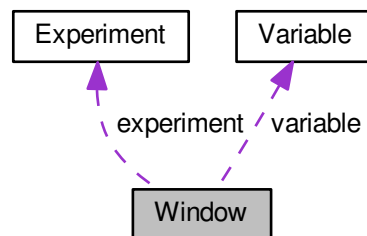
- [interface.h](#)

## 4.8 Window Struct Reference

Struct to define the main window.

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

Collaboration diagram for Window:



### Data Fields

- GtkWidget \* [window](#)  
*Main GtkWidget.*
- GtkWidget \* [grid](#)  
*Main GtkWidget.*
- GtkWidget \* [bar\\_buttons](#)  
*GtkWidget to store the main buttons.*
- GtkWidget \* [button\\_open](#)  
*Open GtkWidget.*
- GtkWidget \* [button\\_save](#)  
*Save GtkWidget.*
- GtkWidget \* [button\\_run](#)  
*Run GtkWidget.*
- GtkWidget \* [button\\_options](#)  
*Options GtkWidget.*
- GtkWidget \* [button\\_help](#)  
*Help GtkWidget.*
- GtkWidget \* [button\\_about](#)  
*Help GtkWidget.*
- GtkWidget \* [button\\_exit](#)

- *Exit 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\\_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.*
- GtkCheckBox \* [check\\_gradient](#)  
*GtkCheckBox to check running the gradient based method.*
- GtkGrid \* [grid\\_gradient](#)  
*GtkGrid to pack the gradient based method widgets.*
- GtkRadioButton \* [button\\_gradient](#) [NGRADIENTS]  
*GtkRadioButtons array to set the gradient 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.*
- 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.*
- 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 102 of file [interface.h](#).

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

- [interface.h](#)

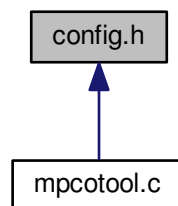
## Chapter 5

# 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 `NGRADIENTS` 2  
*Number of gradient estimate methods.*
- #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\_CALIBRATE (const xmlChar\*)"calibrate"  
*calibrate XML label.*
- #define XML\_COORDINATES (const xmlChar\*)"coordinates"  
*coordinates 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\_GRADIENT\_METHOD (const xmlChar\*)"gradient\_method"  
*gradient\_method 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\_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\*)"nsweeps"  
*nsweeps 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\\_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\\_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 2012-2014, 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: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00031 #ifndef CONFIG__H
00032 #define CONFIG__H 1
00033
00034 // Array sizes
00035
00036 #define MAX_NINPUTS 8
00037 #define NALGORITHMS 3
00038 #define NGRADIENTS 2
00039 #define NPRECISIONS 15
00040
00041 // Default choices
00042
00043 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00044 #define DEFAULT_RANDOM_SEED 7007
00045 #define DEFAULT_RELAXATION 1.
00046
00047 // Interface labels
00048
00049 #define LOCALE_DIR "locales"
00050 #define PROGRAM_INTERFACE "mpcotool"
00051
00052 // XML labels
00053
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_CALIBRATE (const xmlChar*)"calibrate"
00059 #define XML_COORDINATES (const xmlChar*)"coordinates"
00060 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00061 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00062 #define XML_GENETIC (const xmlChar*)"genetic"
00063 #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
00064 #define XML_MINIMUM (const xmlChar*)"minimum"
00065 #define XML_MAXIMUM (const xmlChar*)"maximum"
00066 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00067 #define XML_MUTATION (const xmlChar*)"mutation"
00068 #define XML_NAME (const xmlChar*)"name"
00069 #define XML_NBEST (const xmlChar*)"nbest"
00070 #define XML_NBITS (const xmlChar*)"nbits"
00071 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00072 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00073 #define XML_NITERATIONS (const xmlChar*)"niterations"
00074 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00075 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00076 #define XML_NSTEPS (const xmlChar*)"nsteps"
00077 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00078 #define XML_PRECISION (const xmlChar*)"precision"
00079 #define XML_RANDOM (const xmlChar*)"random"
00080 #define XML_RELAXATION (const xmlChar*)"relaxation"
00081 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00082 #define XML_RESULT (const xmlChar*)"result"
00083 #define XML_SIMULATOR (const xmlChar*)"simulator"
00084 #define XML_SEED (const xmlChar*)"seed"

```

```

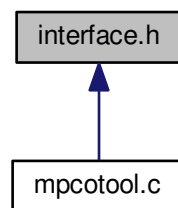
00111 #define XML_STEP (const xmlChar*)"step"
00112 #define XML_SWEEP (const xmlChar*)"sweep"
00113 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00114 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00116 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00118 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00120 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00122 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00124 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00126 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00128 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00130 #define XML_VARIABLE (const xmlChar*)"variable"
00132 #define XML_VARIABLES (const xmlChar*)"variables"
00133 #define XML_WEIGHT (const xmlChar*)"weight"
00135
00136 #endif

```

## 5.3 interface.h File Reference

Header file of the interface.

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



### Data Structures

- struct [Experiment](#)  
*Struct to define experiment data.*
- struct [Variable](#)  
*Struct to define variable data.*
- struct [Options](#)  
*Struct to define the options dialog.*
- struct [Running](#)  
*Struct to define the running dialog.*
- struct [Window](#)  
*Struct to define the main window.*

### Macros

- #define [MAX\\_LENGTH](#) ([DEFAULT\\_PRECISION](#) + 8)  
*Max length of texts allowed in GtkSpinButtons.*

## Functions

- void [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.*
- int [window\\_get\\_algorithm](#) ()  
*Function to get the stochastic algorithm number.*
- int [window\\_get\\_gradient](#) ()  
*Function to get the gradient base method number.*
- void [window\\_save\\_gradient](#) ()  
*Function to save the gradient based method data in the input file.*
- int [window\\_save](#) ()  
*Function to save the input file.*
- void [window\\_run](#) ()  
*Function to run a calibration.*
- void [window\\_help](#) ()  
*Function to show a help dialog.*
- void [window\\_update\\_gradient](#) ()  
*Function to update gradient based 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.*
- int [cores\\_number](#) ()
- Function to obtain the cores number.*

### 5.3.1 Detailed Description

Header file of the interface.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2015, all rights reserved.

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

### 5.3.2 Function Documentation

#### 5.3.2.1 int cores\_number ( )

Function to obtain the cores number.

#### Returns

Cores number.

Definition at line [4707](#) of file [mpcotool.c](#).

```
04708 {
04709     #ifdef G_OS_WIN32
04710         SYSTEM_INFO sysinfo;
04711         GetSystemInfo (&sysinfo);
04712         return sysinfo.dwNumberOfProcessors;
04713     #else
04714         return (int) sysconf (_SC_NPROCESSORS_ONLN);
04715     #endif
04716 }
```

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

Function to save the input file.

## Parameters

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

Definition at line 2670 of file `mpcotoool.c`.

```

02671 {
02672     unsigned int i, j;
02673     char *buffer;
02674     xmlDoc *doc;
02675     xmlNode *node, *child;
02676     GFile *file, *file2;
02677
02678     // Getting the input file directory
02679     input->name = g_path_get_basename (filename);
02680     input->directory = g_path_get_dirname (filename);
02681     file = g_file_new_for_path (input->directory);
02682
02683     // Opening the input file
02684     doc = xmlNewDoc ((const xmlChar *) "1.0");
02685
02686     // Setting root XML node
02687     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02688     xmlDocSetRootElement (doc, node);
02689
02690     // Adding properties to the root XML node
02691     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02692         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02693     if (xmlStrcmp ((const xmlChar *) input->variables,
02694         variables_name))
02695         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02696         variables);
02697     file2 = g_file_new_for_path (input->simulator);
02698     buffer = g_file_get_relative_path (file, file2);
02699     g_object_unref (file2);
02700     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02701     g_free (buffer);
02702     if (input->evaluator)
02703     {
02704         file2 = g_file_new_for_path (input->evaluator);
02705         buffer = g_file_get_relative_path (file, file2);
02706         g_object_unref (file2);
02707         if (xmlStrlen ((xmlChar *) buffer))
02708             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02709         g_free (buffer);
02710     }
02711     if (input->seed != DEFAULT_RANDOM_SEED)
02712         xml_node_set_uint (node, XML_SEED, input->seed);
02713
02714     // Setting the algorithm
02715     buffer = (char *) g_malloc (64);
02716     switch (input->algorithm)
02717     {
02718     case ALGORITHM_MONTE_CARLO:
02719         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02720         snprintf (buffer, 64, "%u", input->nsimulations);
02721         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02722         snprintf (buffer, 64, "%u", input->niterations);
02723         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02724         snprintf (buffer, 64, "%.3lg", input->tolerance);
02725         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02726         snprintf (buffer, 64, "%u", input->nbest);
02727         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02728         input_save_gradient (node);
02729         break;
02730     case ALGORITHM_SWEEP:
02731         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02732         snprintf (buffer, 64, "%u", input->niterations);
02733         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02734         snprintf (buffer, 64, "%.3lg", input->tolerance);
02735         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02736         snprintf (buffer, 64, "%u", input->nbest);
02737         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02738         input_save_gradient (node);
02739         break;
02740     default:
02741         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02742         snprintf (buffer, 64, "%u", input->nsimulations);
02743         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02744         snprintf (buffer, 64, "%u", input->niterations);
02745         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02746         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02747         xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02748         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02749         xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02750         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);

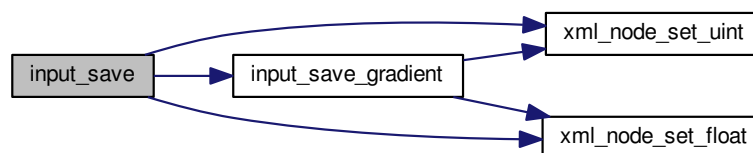
```

```

02749     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02750     break;
02751 }
02752 g_free (buffer);
02753
02754 // Setting the experimental data
02755 for (i = 0; i < input->nexperiments; ++i)
02756 {
02757     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02758     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02759     if (input->weight[i] != 1.)
02760         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02761     for (j = 0; j < input->ninputs; ++j)
02762         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02763 }
02764
02765 // Setting the variables data
02766 for (i = 0; i < input->nvariables; ++i)
02767 {
02768     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02769     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02770     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02771     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02772         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02773     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02774     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02775         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02776     if (input->precision[i] != DEFAULT_PRECISION)
02777         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02778     if (input->algorithm == ALGORITHM_SWEEP)
02779         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02780     else if (input->algorithm == ALGORITHM_GENETIC)
02781         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02782 }
02783
02784 // Saving the XML file
02785 xmlSaveFormatFile (filename, doc, 1);
02786
02787 // Freeing memory
02788 xmlFreeDoc (doc);
02789 }

```

Here is the call graph for this function:



### 5.3.2.3 int window\_get\_algorithm ( )

Function to get the stochastic algorithm number.

#### Returns

Stochastic algorithm number.

Definition at line 2887 of file [mpcotool.c](#).

```

02888 {
02889     unsigned int i;
02890     for (i = 0; i < NALGORITHMS; ++i)
02891         if (gtk_toggle_button_get_active
02892             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02893             break;
02894     return i;
02895 }

```

#### 5.3.2.4 int window\_get\_gradient ( )

Function to get the gradient base method number.

##### Returns

Gradient base method number.

Definition at line 2903 of file [mpcotool.c](#).

```

02904 {
02905     unsigned int i;
02906     for (i = 0; i < NGRADIENTS; ++i)
02907         if (gtk_toggle_button_get_active
02908             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02909             break;
02910     return i;
02911 }

```

#### 5.3.2.5 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 3908 of file [mpcotool.c](#).

```

03909 {
03910     unsigned int i;
03911     char *buffer;
03912     #if DEBUG
03913     fprintf (stderr, "window_read: start\n");
03914     #endif
03915     // Reading new input file
03916     input_free ();
03917     if (!input_open (filename))
03918         return 0;
03919     // Setting GTK+ widgets data
03920     gtk_entry_set_text (window->entry_result, input->result);
03921     gtk_entry_set_text (window->entry_variables, input->
variables);
03922     buffer = g_build_filename (input->directory, input->
simulator, NULL);
03923     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
03924     g_free (buffer);
03925     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
03926     if (input->evaluator)
03927     {
03928         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
03929         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER

```

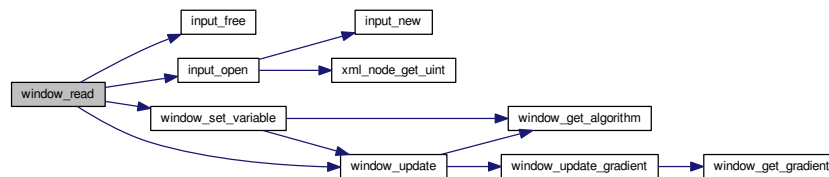


```

03934                                     (window->button_evaluator), buffer);
03935     g_free (buffer);
03936 }
03937 gtk_toggle_button_set_active
03938 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
03939 algorithm]), TRUE);
03939 switch (input->algorithm)
03940 {
03941     case ALGORITHM_MONTE_CARLO:
03942         gtk_spin_button_set_value (window->spin_simulations,
03943                                     (gdouble) input->nsimulations);
03944     case ALGORITHM_SWEEP:
03945         gtk_spin_button_set_value (window->spin_iterations,
03946                                     (gdouble) input->niterations);
03947         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
03948         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
03949         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
                                     input->nsteps);
03950     if (input->nsteps)
03951     {
03952         gtk_toggle_button_set_active
03953             (GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
03954         gtk_spin_button_set_value (window->spin_steps,
03955                                     (gdouble) input->nsteps);
03956         gtk_spin_button_set_value (window->spin_relaxation,
03957                                     (gdouble) input->relaxation);
03958         switch (input->gradient_method)
03959         {
03960             case GRADIENT_METHOD_RANDOM:
03961                 gtk_spin_button_set_value (window->spin_estimates,
03962                                             (gdouble) input->nestimates);
03963             }
03964         }
03965         break;
03966     default:
03967         gtk_spin_button_set_value (window->spin_population,
03968                                     (gdouble) input->nsimulations);
03969         gtk_spin_button_set_value (window->spin_generations,
03970                                     (gdouble) input->niterations);
03971         gtk_spin_button_set_value (window->spin_adaptation,
03972                                     (gdouble) input->adaptation_ratio);
03973         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
03974         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
03975         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
03976         }
03977     }
03978     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03979     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
03980     gtk_combo_box_text_remove_all (window->combo_experiment);
03981     for (i = 0; i < input->nexperiments; ++i)
03982         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
03983     g_signal_handler_unblock
03984         (window->button_experiment, window->
id_experiment_name);
03985     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
03986     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03987     g_signal_handler_block (window->combo_variable, window->
id_variable);
03988     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03989     gtk_combo_box_text_remove_all (window->combo_variable);
03990     for (i = 0; i < input->nvariables; ++i)
03991         gtk_combo_box_text_append_text (window->combo_variable,
input->label[i]);
03992     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03993     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03994     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03995     window_set_variable ();
03996     window_update ();
04000
04001 #if DEBUG
04002     fprintf (stderr, "window_read: end\n");
04003 #endif
04004     return 1;
04005 }

```

Here is the call graph for this function:



### 5.3.2.6 int window\_save ( )

Function to save the input file.

#### Returns

1 on OK, 0 on Cancel.

Definition at line 2951 of file `mpcotool.c`.

```

02952 {
02953     char *buffer;
02954     GtkFileChooserDialog *dlg;
02955
02956     #if DEBUG
02957         fprintf (stderr, "window_save: start\n");
02958     #endif
02959
02960     // Opening the saving dialog
02961     dlg = (GtkFileChooserDialog *)
02962         gtk_file_chooser_dialog_new (gettext ("Save file"),
02963                                     window->window,
02964                                     GTK_FILE_CHOOSER_ACTION_SAVE,
02965                                     gettext ("_Cancel"),
02966                                     GTK_RESPONSE_CANCEL,
02967                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02968     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02969     buffer = g_build_filename (input->directory, input->name, NULL);
02970     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02971     g_free (buffer);
02972
02973     // If OK response then saving
02974     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02975     {
02976
02977         // Adding properties to the root XML node
02978         input->simulator = gtk_file_chooser_get_filename
02979             (GTK_FILE_CHOOSER (window->button_simulator));
02980         if (gtk_toggle_button_get_active
02981             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02982             input->evaluator = gtk_file_chooser_get_filename
02983                 (GTK_FILE_CHOOSER (window->button_evaluator));
02984         else
02985             input->evaluator = NULL;
02986         input->result
02987             = (char *) xmlStrdup ((const xmlChar *)
02988                                   gtk_entry_get_text (window->entry_result));
02989         input->variables
02990             = (char *) xmlStrdup ((const xmlChar *)
02991                                   gtk_entry_get_text (window->entry_variables));
02992
02993         // Setting the algorithm
02994         switch (window_get_algorithm ())
02995         {
02996             case ALGORITHM_MONTE_CARLO:
02997                 input->algorithm = ALGORITHM_MONTE_CARLO;
02998                 input->nsimulations
02999                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
03000                 input->niterations
03001                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002                 input->tolerance = gtk_spin_button_get_value (window->

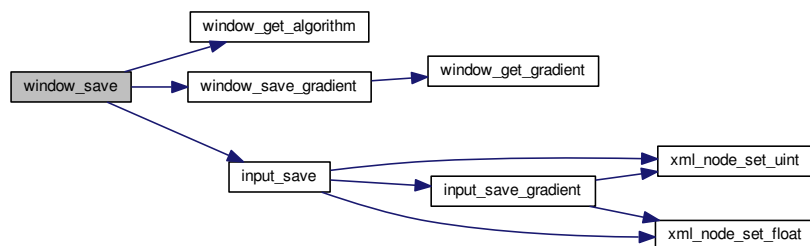
```

```

    spin_tolerance);
03003     input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
03004     window_save_gradient ();
03005     break;
03006     case ALGORITHM_SWEEP:
03007         input->algorithm = ALGORITHM_SWEEP;
03008         input->niterations
03009             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03010         input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
03011         input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
03012         window_save_gradient ();
03013         break;
03014     default:
03015         input->algorithm = ALGORITHM_GENETIC;
03016         input->nsimulations
03017             = gtk_spin_button_get_value_as_int (window->spin_population);
03018         input->niterations
03019             = gtk_spin_button_get_value_as_int (window->spin_generations);
03020         input->mutation_ratio
03021             = gtk_spin_button_get_value (window->spin_mutation);
03022         input->reproduction_ratio
03023             = gtk_spin_button_get_value (window->spin_reproduction);
03024         input->adaptation_ratio
03025             = gtk_spin_button_get_value (window->spin_adaptation);
03026         break;
03027     }
03028
03029     // Saving the XML file
03030     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03031     input_save (buffer);
03032
03033     // Closing and freeing memory
03034     g_free (buffer);
03035     gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037     fprintf (stderr, "window_save: end\n");
03038 #endif
03039     return 1;
03040 }
03041
03042 // Closing and freeing memory
03043 gtk_widget_destroy (GTK_WIDGET (dlg));
03044 #if DEBUG
03045     fprintf (stderr, "window_save: end\n");
03046 #endif
03047     return 0;
03048 }

```

Here is the call graph for this function:



### 5.3.2.7 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 3544 of file `mpcotool.c`.

```

03545 {
03546     unsigned int i, j;
03547     char *buffer;
03548     GFile *file1, *file2;
03549     #if DEBUG
03550     fprintf (stderr, "window_template_experiment: start\n");
03551     #endif
03552     i = (size_t) data;
03553     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03554     file1
03555         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03556     file2 = g_file_new_for_path (input->directory);
03557     buffer = g_file_get_relative_path (file2, file1);
03558     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03559     g_free (buffer);
03560     g_object_unref (file2);
03561     g_object_unref (file1);
03562     #if DEBUG
03563     fprintf (stderr, "window_template_experiment: end\n");
03564     #endif
03565 }
```

## 5.4 interface.h

```

00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00030 #ifndef INTERFACE__H
00031 #define INTERFACE__H 1
00032
00033 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00034
00035 typedef struct
00036 {
00037     char *template[MAX_NINPUS];
00038     char *name;
00039     double weight;
00040 } Experiment;
00041
00042 typedef struct
00043 {
00044     char *label;
00045     double rangemin;
00046     double rangemax;
00047     double rangeminabs;
00048     double rangemaxabs;
00049     unsigned int precision;
00050     unsigned int nsweeps;
00051     unsigned int nbits;
00052 } Variable;
```

```
00069
00074 typedef struct
00075 {
00076     GtkDialog *dialog;
00077     GtkGrid *grid;
00078     GtkLabel *label_seed;
00080     GtkSpinButton *spin_seed;
00082     GtkLabel *label_threads;
00083     GtkSpinButton *spin_threads;
00084     GtkLabel *label_gradient;
00085     GtkSpinButton *spin_gradient;
00086 } Options;
00087
00092 typedef struct
00093 {
00094     GtkDialog *dialog;
00095     GtkLabel *label;
00096 } Running;
00097
00102 typedef struct
00103 {
00104     GtkWidget *window;
00105     GtkGrid *grid;
00106     GtkToolBar *bar_buttons;
00107     GtkToolButton *button_open;
00108     GtkToolButton *button_save;
00109     GtkToolButton *button_run;
00110     GtkToolButton *button_options;
00111     GtkToolButton *button_help;
00112     GtkToolButton *button_about;
00113     GtkToolButton *button_exit;
00114     GtkGrid *grid_files;
00115     GtkLabel *label_simulator;
00116     GtkFileChooserButton *button_simulator;
00118     GtkCheckButton *check_evaluator;
00119     GtkFileChooserButton *button_evaluator;
00121     GtkLabel *label_result;
00122     GtkEntry *entry_result;
00123     GtkLabel *label_variables;
00124     GtkEntry *entry_variables;
00125     GtkFrame *frame_algorithm;
00126     GtkGrid *grid_algorithm;
00127     GtkRadioButton *button_algorithm[NALGORITHMS];
00129     GtkLabel *label_simulations;
00130     GtkSpinButton *spin_simulations;
00132     GtkLabel *label_iterations;
00133     GtkSpinButton *spin_iterations;
00135     GtkLabel *label_tolerance;
00136     GtkSpinButton *spin_tolerance;
00137     GtkLabel *label_bests;
00138     GtkSpinButton *spin_bests;
00139     GtkLabel *label_population;
00140     GtkSpinButton *spin_population;
00142     GtkLabel *label_generations;
00143     GtkSpinButton *spin_generations;
00145     GtkLabel *label_mutation;
00146     GtkSpinButton *spin_mutation;
00147     GtkLabel *label_reproduction;
00148     GtkSpinButton *spin_reproduction;
00150     GtkLabel *label_adaptation;
00151     GtkSpinButton *spin_adaptation;
00153     GtkCheckButton *check_gradient;
00155     GtkGrid *grid_gradient;
00157     GtkRadioButton *button_gradient[NGRADIENTS];
00159     GtkLabel *label_steps;
00160     GtkSpinButton *spin_steps;
00161     GtkLabel *label_estimates;
00162     GtkSpinButton *spin_estimates;
00164     GtkLabel *label_relaxation;
00166     GtkSpinButton *spin_relaxation;
00168     GtkFrame *frame_variable;
00169     GtkGrid *grid_variable;
00170     GtkComboBoxText *combo_variable;
00172     GtkButton *button_add_variable;
00173     GtkButton *button_remove_variable;
00174     GtkLabel *label_variable;
00175     GtkEntry *entry_variable;
00176     GtkLabel *label_min;
00177     GtkSpinButton *spin_min;
00178     GtkScrolledWindow *scrolled_min;
00179     GtkLabel *label_max;
00180     GtkSpinButton *spin_max;
00181     GtkScrolledWindow *scrolled_max;
00182     GtkCheckButton *check_minabs;
00183     GtkSpinButton *spin_minabs;
00184     GtkScrolledWindow *scrolled_minabs;
00185     GtkCheckButton *check_maxabs;
```

```

00186   GtkWidget *spin_maxabs;
00187   GtkScrolledWindow *scrolled_maxabs;
00188   GtkWidget *label_precision;
00189   GtkWidget *spin_precision;
00190   GtkWidget *label_sweeps;
00191   GtkWidget *spin_sweeps;
00192   GtkWidget *label_bits;
00193   GtkWidget *spin_bits;
00194   GtkFrame *frame_experiment;
00195   GtkWidget *grid_experiment;
00196   GtkComboBoxText *combo_experiment;
00197   GtkWidget *button_add_experiment;
00198   GtkWidget *button_remove_experiment;
00199   GtkWidget *label_experiment;
00200   GtkFileChooserButton *button_experiment;
00202   GtkWidget *label_weight;
00203   GtkWidget *spin_weight;
00204   GtkCheckButton *check_template[MAX_NINPUTS];
00206   GtkFileChooserButton *button_template[MAX_NINPUTS];
00208   GdkPixbuf *logo;
00209   Experiment *experiment;
00210   Variable *variable;
00211   char *application_directory;
00212   gulong id_experiment;
00213   gulong id_experiment_name;
00214   gulong id_variable;
00215   gulong id_variable_label;
00216   gulong id_template[MAX_NINPUTS];
00218   gulong id_input[MAX_NINPUTS];
00220   unsigned int nexperiments;
00221   unsigned int nvariables;
00222 } Window;
00223
00224 // Public functions
00225 void input_save (char *filename);
00226 void options_new ();
00227 void running_new ();
00228 int window_get_algorithm ();
00229 int window_get_gradient ();
00230 void window_save_gradient ();
00231 int window_save ();
00232 void window_run ();
00233 void window_help ();
00234 void window_update_gradient ();
00235 void window_update ();
00236 void window_set_algorithm ();
00237 void window_set_experiment ();
00238 void window_remove_experiment ();
00239 void window_add_experiment ();
00240 void window_name_experiment ();
00241 void window_weight_experiment ();
00242 void window_inputs_experiment ();
00243 void window_template_experiment (void *data);
00244 void window_set_variable ();
00245 void window_remove_variable ();
00246 void window_add_variable ();
00247 void window_label_variable ();
00248 void window_precision_variable ();
00249 void window_rangemin_variable ();
00250 void window_rangemax_variable ();
00251 void window_rangeminabs_variable ();
00252 void window_rangemaxabs_variable ();
00253 void window_update_variable ();
00254 int window_read (char *filename);
00255 void window_open ();
00256 void window_new ();
00257 int cores_number ();
00258
00259 #endif

```

## 5.5 mpcotool.c File Reference

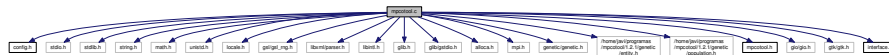
Source file of the mpcotool.

```

#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.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 "genetic/genetic.h"
#include "mpcotool.h"
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "interface.h"

```

Include dependency graph for mpcotool.c:



## Macros

- `#define _GNU_SOURCE`
- `#define DEBUG 0`  
*Macro to debug.*
- `#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.*
- `#define INPUT_FILE "test-ga.xml"`  
*Macro to define the initial input file.*
- `#define RM "rm"`  
*Macro to define the shell remove command.*

## Functions

- void `show_message` (char \*title, char \*msg, int type)  
*Function to show a dialog with a message.*
- void `show_error` (char \*msg)  
*Function to show a dialog with an error message.*
- int `xml_node_get_int` (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an integer number of a XML node property.*
- unsigned int `xml_node_get_uint` (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an unsigned integer number of a XML node property.*
- 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.*
- 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.*

  - void [input\\_new](#) ()
- Function to create a new [Input](#) struct.*

  - void [input\\_free](#) ()
- Function to free the memory of the input file data.*

  - int [input\\_open](#) (char \*filename)
- Function to open the input file.*

  - void [calibrate\\_input](#) (unsigned int simulation, char \*input, GMappedFile \*template)
- Function to write the simulation input file.*

  - double [calibrate\\_parse](#) (unsigned int simulation, unsigned int experiment)
- Function to parse input files, simulating and calculating the \ objective function.*

  - void [calibrate\\_print](#) ()
- Function to print the results.*

  - void [calibrate\\_save\\_variables](#) (unsigned int simulation, double error)
- Function to save in a file the variables and the error.*

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

  - void [calibrate\\_sequential](#) ()
- Function to calibrate sequentially.*

  - void \* [calibrate\\_thread](#) ([ParallelData](#) \*data)
- Function to calibrate on a thread.*

  - void [calibrate\\_merge](#) (unsigned int nsaveds, unsigned int \*simulation\_best, double \*error\_best)
- Function to merge the 2 calibration results.*

  - void [calibrate\\_synchronise](#) ()
- Function to synchronise the calibration results of MPI tasks.*

  - void [calibrate\\_sweep](#) ()
- Function to calibrate with the sweep algorithm.*

  - void [calibrate\\_MonteCarlo](#) ()
- Function to calibrate with the Monte-Carlo algorithm.*

  - void [calibrate\\_best\\_gradient](#) (unsigned int simulation, double value)
- Function to save the best simulation in a gradient based method.*

  - void [calibrate\\_gradient\\_sequential](#) (unsigned int simulation)
- Function to estimate the gradient sequentially.*

  - void \* [calibrate\\_gradient\\_thread](#) ([ParallelData](#) \*data)
- Function to estimate the gradient on a thread.*

  - double [calibrate\\_estimate\\_gradient\\_random](#) (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the gradient vector.*

  - double [calibrate\\_estimate\\_gradient\\_coordinates](#) (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the gradient vector.*

  - void [calibrate\\_step\\_gradient](#) (unsigned int simulation)
- Function to do a step of the gradient based method.*

  - void [calibrate\\_gradient](#) ()
- Function to calibrate with a gradient based method.*

  - double [calibrate\\_genetic\\_objective](#) (Entity \*entity)
- Function to calculate the objective function of an entity.*

  - void [calibrate\\_genetic](#) ()
- Function to calibrate with the genetic algorithm.*



- void [calibrate\\_save\\_old](#) ()  
*Function to save the best results on iterative methods.*
- void [calibrate\\_merge\\_old](#) ()  
*Function to merge the best results with the previous step best results on iterative methods.*
- void [calibrate\\_refine](#) ()  
*Function to refine the search ranges of the variables in iterative algorithms.*
- void [calibrate\\_step](#) ()  
*Function to do a step of the iterative algorithm.*
- void [calibrate\\_iterate](#) ()  
*Function to iterate the algorithm.*
- void [calibrate\\_free](#) ()  
*Function to free the memory used by [Calibrate](#) struct.*
- void [calibrate\\_open](#) ()  
*Function to open and perform a calibration.*
- void [input\\_save\\_gradient](#) (xmlNode \*node)  
*Function to save the gradient based 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.*
- int [window\\_get\\_algorithm](#) ()  
*Function to get the stochastic algorithm number.*
- int [window\\_get\\_gradient](#) ()  
*Function to get the gradient base method number.*
- void [window\\_save\\_gradient](#) ()  
*Function to save the gradient based method data in the input file.*
- int [window\\_save](#) ()  
*Function to save the input file.*
- void [window\\_run](#) ()  
*Function to run a calibration.*
- void [window\\_help](#) ()  
*Function to show a help dialog.*
- void [window\\_about](#) ()  
*Function to show an about dialog.*
- void [window\\_update\\_gradient](#) ()  
*Function to update gradient based 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.*

  - int `cores_number` ()
- Function to obtain the cores number.*

  - int `main` (int argn, char \*\*argc)
- Main function.*

## Variables

- int `ntasks`

*Number of tasks.*
- unsigned int `nthreads`

*Number of threads.*
- unsigned int `nthreads_gradient`

*Number of threads for the gradient based method.*
- GMutex `mutex` [1]

*Mutex struct.*
- void(\* `calibrate_algorithm` )()

*Pointer to the function to perform a calibration algorithm step.*
- double(\* `calibrate_estimate_gradient` )(unsigned int variable, unsigned int estimate)

*Pointer to the function to estimate the gradient.*

- [Input input](#) [1]  
*Input struct to define the input file to mpcotool.*
- [Calibrate calibrate](#) [1]  
*Calibration data.*
- `const xmlChar * result\_name = (xmlChar *) "result"`  
*Name of the result file.*
- `const xmlChar * variables\_name = (xmlChar *) "variables"`  
*Name of the variables file.*
- `const xmlChar * template [MAX_NINPUTS]`  
*Array of xmlChar strings with template labels.*
- `const char * format [NPRECISIONS]`  
*Array of C-strings with variable formats.*
- `const double precision [NPRECISIONS]`  
*Array of variable precisions.*
- `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.*

### 5.5.1 Detailed Description

Source file of the mpcotool.

#### Authors

Javier Burguete and Borja Latorre.

#### Copyright

Copyright 2012-2015, all rights reserved.

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

### 5.5.2 Function Documentation

#### 5.5.2.1 void [calibrate\\_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 [1420](#) of file [mpcotool.c](#).

```
01421 {
01422     unsigned int i, j;
01423     double e;
01424     #if DEBUG
01425         fprintf (stderr, "calibrate_best: start\n");
01426         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
```

```

01427         calibrate->nsaveds, calibrate->nbest);
01428 #endif
01429     if (calibrate->nsaveds < calibrate->nbest
01430         || value < calibrate->error_best[calibrate->nsaveds - 1])
01431     {
01432         if (calibrate->nsaveds < calibrate->nbest)
01433             ++calibrate->nsaveds;
01434         calibrate->error_best[calibrate->nsaveds - 1] = value;
01435         calibrate->simulation_best[calibrate->
01436             nsaveds - 1] = simulation;
01437         for (i = calibrate->nsaveds; --i;)
01438         {
01439             if (calibrate->error_best[i] < calibrate->
01440                 error_best[i - 1])
01441             {
01442                 j = calibrate->simulation_best[i];
01443                 e = calibrate->error_best[i];
01444                 calibrate->simulation_best[i] = calibrate->
01445                     simulation_best[i - 1];
01446                 calibrate->error_best[i] = calibrate->
01447                     error_best[i - 1];
01448                 calibrate->simulation_best[i - 1] = j;
01449                 calibrate->error_best[i - 1] = e;
01450             }
01451             else
01452                 break;
01453         }
01454     }
01455 #if DEBUG
01456     fprintf (stderr, "calibrate_best: end\n");
01457 #endif
01458 }

```

#### 5.5.2.2 void calibrate\_best\_gradient ( unsigned int *simulation*, double *value* )

Function to save the best simulation in a gradient based method.

##### Parameters

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

Definition at line 1733 of file [mpcotool.c](#).

```

01734 {
01735 #if DEBUG
01736     fprintf (stderr, "calibrate_best_gradient: start\n");
01737     fprintf (stderr,
01738         "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01739         simulation, value, calibrate->error_best[0]);
01740 #endif
01741     if (value < calibrate->error_best[0])
01742     {
01743         calibrate->error_best[0] = value;
01744         calibrate->simulation_best[0] = simulation;
01745 #if DEBUG
01746         fprintf (stderr,
01747             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01748             simulation, value);
01749 #endif
01750     }
01751 #if DEBUG
01752     fprintf (stderr, "calibrate_best_gradient: end\n");
01753 #endif
01754 }

```

#### 5.5.2.3 double calibrate\_estimate\_gradient\_coordinates ( unsigned int *variable*, unsigned int *estimate* )

Function to estimate a component of the gradient vector.

## Parameters

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

Definition at line 1870 of file [mpcotool.c](#).

```

01872 {
01873     double x;
01874     #if DEBUG
01875     fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01876     #endif
01877     x = calibrate->gradient[variable];
01878     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01879     {
01880         if (estimate & 1)
01881             x += calibrate->step[variable];
01882         else
01883             x -= calibrate->step[variable];
01884     }
01885     #if DEBUG
01886     fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01887             variable, x);
01888     fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01889     #endif
01890     return x;
01891 }
```

#### 5.5.2.4 double calibrate\_estimate\_gradient\_random ( unsigned int *variable*, unsigned int *estimate* )

Function to estimate a component of the gradient vector.

## Parameters

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

Definition at line 1843 of file [mpcotool.c](#).

```

01845 {
01846     double x;
01847     #if DEBUG
01848     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01849     #endif
01850     x = calibrate->gradient[variable]
01851         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
01852         step[variable];
01853     #if DEBUG
01854     fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01855             variable, x);
01856     fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01857     #endif
01858     return x;
01859 }
```

#### 5.5.2.5 double calibrate\_genetic\_objective ( Entity \* *entity* )

Function to calculate the objective function of an entity.

## Parameters

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

## Returns

objective function value.

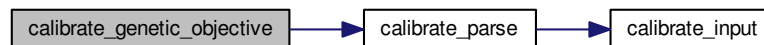
Definition at line 2036 of file [mpcotool.c](#).

```

02037 {
02038     unsigned int j;
02039     double objective;
02040     char buffer[64];
02041     #if DEBUG
02042     fprintf (stderr, "calibrate_genetic_objective: start\n");
02043     #endif
02044     for (j = 0; j < calibrate->nvariables; ++j)
02045     {
02046         calibrate->value[entity->id * calibrate->nvariables + j]
02047         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02048     }
02049     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02050         objective += calibrate_parse (entity->id, j);
02051     g_mutex_lock (mutex);
02052     for (j = 0; j < calibrate->nvariables; ++j)
02053     {
02054         snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02055         fprintf (calibrate->file_variables, buffer,
02056                 genetic_get_variable (entity, calibrate->
02057                                     genetic_variable + j));
02058     }
02059     fprintf (calibrate->file_variables, "%.14le\n", objective);
02060     g_mutex_unlock (mutex);
02061     #if DEBUG
02062     fprintf (stderr, "calibrate_genetic_objective: end\n");
02063     #endif
02064     return objective;
02065 }

```

Here is the call graph for this function:



#### 5.5.2.6 void calibrate\_gradient\_sequential ( unsigned int *simulation* )

Function to estimate the gradient sequentially.

##### Parameters

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

Definition at line 1763 of file [mpcotool.c](#).

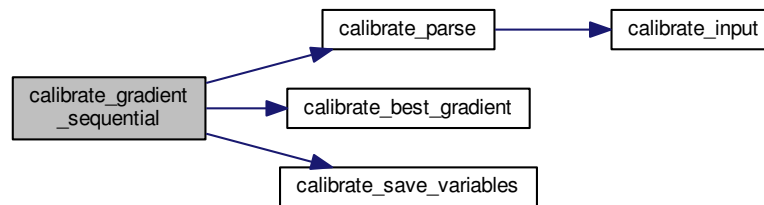
```

01764 {
01765     unsigned int i, j, k;
01766     double e;
01767     #if DEBUG
01768     fprintf (stderr, "calibrate_gradient_sequential: start\n");
01769     fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01770             "nend_gradient=%u\n",
01771             calibrate->nstart_gradient, calibrate->
01772             nend_gradient);
01773     #endif
01774     for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01775     {
01776         k = simulation + i;
01777         e = 0.;
01778         for (j = 0; j < calibrate->nexperiments; ++j)
01779             e += calibrate_parse (k, j);
01780         calibrate_best_gradient (k, e);
01781         calibrate_save_variables (k, e);
01782     }
01783     #if DEBUG
01784     fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01785     #endif
01786     #if DEBUG
01787     fprintf (stderr, "calibrate_gradient_sequential: end\n");
01788     #endif
01789 }

```

```
01787 #endif
01788 }
```

Here is the call graph for this function:



#### 5.5.2.7 void \* calibrate\_gradient\_thread ( ParallelData \* data )

Function to estimate the gradient on a thread.

##### Parameters

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

##### Returns

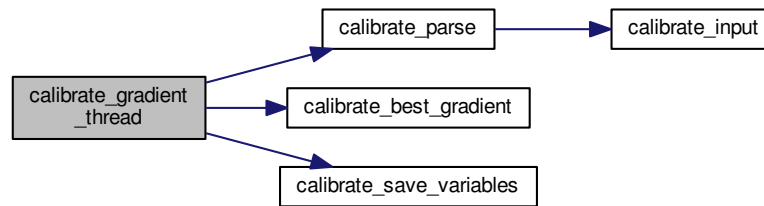
NULL

Definition at line 1798 of file [mpcotool.c](#).

```

01799 {
01800     unsigned int i, j, thread;
01801     double e;
01802     #if DEBUG
01803         fprintf (stderr, "calibrate_gradient_thread: start\n");
01804     #endif
01805     thread = data->thread;
01806     #if DEBUG
01807         fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01808                 thread,
01809                 calibrate->thread_gradient[thread],
01810                 calibrate->thread_gradient[thread + 1]);
01811     #endif
01812     for (i = calibrate->thread_gradient[thread];
01813          i < calibrate->thread_gradient[thread + 1]; ++i)
01814     {
01815         e = 0.;
01816         for (j = 0; j < calibrate->nexperiments; ++j)
01817             e += calibrate_parse (i, j);
01818         g_mutex_lock (mutex);
01819         calibrate_best_gradient (i, e);
01820         calibrate_save_variables (i, e);
01821         g_mutex_unlock (mutex);
01822     #if DEBUG
01823         fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01824     #endif
01825     }
01826     #if DEBUG
01827         fprintf (stderr, "calibrate_gradient_thread: end\n");
01828     #endif
01829     g_thread_exit (NULL);
01830     return NULL;
01831 }
```

Here is the call graph for this function:



#### 5.5.2.8 void calibrate\_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 1173 of file [mpcotool.c](#).

```

01174 {
01175     unsigned int i;
01176     char buffer[32], value[32], *buffer2, *buffer3, *content;
01177     FILE *file;
01178     gsize length;
01179     GRegex *regex;
01180
01181     #if DEBUG
01182         fprintf (stderr, "calibrate_input: start\n");
01183     #endif
01184
01185     // Checking the file
01186     if (!template)
01187         goto calibrate_input_end;
01188
01189     // Opening template
01190     content = g_mapped_file_get_contents (template);
01191     length = g_mapped_file_get_length (template);
01192     #if DEBUG
01193         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01194                 content);
01195     #endif
01196     file = g_fopen (input, "w");
01197
01198     // Parsing template
01199     for (i = 0; i < calibrate->nvariables; ++i)
01200     {
01201         #if DEBUG
01202             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01203         #endif
01204         snprintf (buffer, 32, "@variable%u@", i + 1);
01205         regex = g_regex_new (buffer, 0, 0, NULL);
01206         if (i == 0)
01207         {
01208             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01209                                                calibrate->label[i], 0, NULL);
01210         #if DEBUG
01211             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01212         #endif
01213         }
01214         else
01215         {
01216             length = strlen (buffer3);
01217             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01218                                                calibrate->label[i], 0, NULL);

```



```

01219         g_free (buffer3);
01220     }
01221     g_regex_unref (regex);
01222     length = strlen (buffer2);
01223     snprintf (buffer, 32, "@value%u@", i + 1);
01224     regex = g_regex_new (buffer, 0, 0, NULL);
01225     snprintf (value, 32, format[calibrate->precision[i]],
01226              calibrate->value[simulation * calibrate->
nvariables + i]);
01227
01228     #if DEBUG
01229     fprintf (stderr, "calibrate_input: value=%s\n", value);
01230     #endif
01231     buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01232                                     0, NULL);
01233     g_free (buffer2);
01234     g_regex_unref (regex);
01235 }
01236
01237 // Saving input file
01238 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01239 g_free (buffer3);
01240 fclose (file);
01241
01242 calibrate_input_end:
01243 #if DEBUG
01244 fprintf (stderr, "calibrate_input: end\n");
01245 #endif
01246 return;
01247 }

```

#### 5.5.2.9 void calibrate\_merge ( unsigned int *nsaveds*, unsigned int \* *simulation\_best*, double \* *error\_best* )

Function to merge the 2 calibration 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 1538 of file [mpcotool.c](#).

```

01540 {
01541     unsigned int i, j, k, s[calibrate->nbest];
01542     double e[calibrate->nbest];
01543     #if DEBUG
01544     fprintf (stderr, "calibrate_merge: start\n");
01545     #endif
01546     i = j = k = 0;
01547     do
01548     {
01549         if (i == calibrate->nsaveds)
01550         {
01551             s[k] = simulation_best[j];
01552             e[k] = error_best[j];
01553             ++j;
01554             ++k;
01555             if (j == nsaveds)
01556                 break;
01557         }
01558         else if (j == nsaveds)
01559         {
01560             s[k] = calibrate->simulation_best[i];
01561             e[k] = calibrate->error_best[i];
01562             ++i;
01563             ++k;
01564             if (i == calibrate->nsaveds)
01565                 break;
01566         }
01567         else if (calibrate->error_best[i] > error_best[j])
01568         {
01569             s[k] = simulation_best[j];
01570             e[k] = error_best[j];
01571             ++j;
01572             ++k;
01573         }
01574         else
01575         {
01576             s[k] = calibrate->simulation_best[i];

```

```

01577         e[k] = calibrate->error_best[i];
01578         ++i;
01579         ++k;
01580     }
01581 }
01582 while (k < calibrate->nbest);
01583 calibrate->nsaveds = k;
01584 memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01585 memcpy (calibrate->error_best, e, k * sizeof (double));
01586 #if DEBUG
01587 fprintf (stderr, "calibrate_merge: end\n");
01588 #endif
01589 }

```

### 5.5.2.10 double calibrate\_parse ( unsigned int *simulation*, unsigned int *experiment* )

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

#### Parameters

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

#### Returns

Objective function value.

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

```

01261 {
01262     unsigned int i;
01263     double e;
01264     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01265         *buffer3, *buffer4;
01266     FILE *file_result;
01267
01268 #if DEBUG
01269     fprintf (stderr, "calibrate_parse: start\n");
01270     fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01271             experiment);
01272 #endif
01273
01274     // Opening input files
01275     for (i = 0; i < calibrate->ninputs; ++i)
01276     {
01277         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01278 #if DEBUG
01279         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01280 #endif
01281         calibrate_input (simulation, &input[i][0],
01282                         calibrate->file[i][experiment]);
01283     }
01284     for (; i < MAX_NINPUTS; ++i)
01285         strcpy (&input[i][0], "");
01286 #if DEBUG
01287     fprintf (stderr, "calibrate_parse: parsing end\n");
01288 #endif
01289
01290     // Performing the simulation
01291     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01292     buffer2 = g_path_get_dirname (calibrate->simulator);
01293     buffer3 = g_path_get_basename (calibrate->simulator);
01294     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01295     snprintf (buffer, 512, "%s" "%s %s %s %s %s %s %s %s %s",
01296             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01297             input[6], input[7], output);
01298     g_free (buffer4);
01299     g_free (buffer3);
01300     g_free (buffer2);
01301 #if DEBUG
01302     fprintf (stderr, "calibrate_parse: %s\n", buffer);
01303 #endif
01304     system (buffer);
01305
01306     // Checking the objective value function
01307     if (calibrate->evaluator)
01308     {
01309         snprintf (result, 32, "result-%u-%u", simulation, experiment);

```

```

01310     buffer2 = g_path_get_dirname (calibrate->evaluator);
01311     buffer3 = g_path_get_basename (calibrate->evaluator);
01312     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01313     snprintf (buffer, 512, "\"%s\" %s %s %s",
01314             buffer4, output, calibrate->experiment[experiment], result);
01315     g_free (buffer4);
01316     g_free (buffer3);
01317     g_free (buffer2);
01318     #if DEBUG
01319     fprintf (stderr, "calibrate_parse: %s\n", buffer);
01320     #endif
01321     system (buffer);
01322     file_result = g_fopen (result, "r");
01323     e = atof (fgets (buffer, 512, file_result));
01324     fclose (file_result);
01325 }
01326 else
01327 {
01328     strcpy (result, "");
01329     file_result = g_fopen (output, "r");
01330     e = atof (fgets (buffer, 512, file_result));
01331     fclose (file_result);
01332 }
01333
01334 // Removing files
01335 #if !DEBUG
01336 for (i = 0; i < calibrate->ninputs; ++i)
01337 {
01338     if (calibrate->file[i][0])
01339     {
01340         snprintf (buffer, 512, RM " %s", &input[i][0]);
01341         system (buffer);
01342     }
01343 }
01344     snprintf (buffer, 512, RM " %s %s", output, result);
01345     system (buffer);
01346 #endif
01347
01348 #if DEBUG
01349 fprintf (stderr, "calibrate_parse: end\n");
01350 #endif
01351
01352 // Returning the objective function
01353 return e * calibrate->weight[experiment];
01354 }

```

Here is the call graph for this function:



#### 5.5.2.11 void calibrate\_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 1392 of file [mpcotool.c](#).

```

01393 {
01394     unsigned int i;
01395     char buffer[64];
01396     #if DEBUG

```

```

01397 fprintf (stderr, "calibrate_save_variables: start\n");
01398 #endif
01399 for (i = 0; i < calibrate->nvariables; ++i)
01400 {
01401     snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01402     fprintf (calibrate->file_variables, buffer,
01403             calibrate->value[simulation * calibrate->
01404                     nvariables + i]);
01405     fprintf (calibrate->file_variables, "%.14le\n", error);
01406     #if DEBUG
01407     fprintf (stderr, "calibrate_save_variables: end\n");
01408     #endif
01409 }

```

#### 5.5.2.12 void calibrate\_step\_gradient ( unsigned int *simulation* )

Function to do a step of the gradient based method.

##### Parameters

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

Definition at line 1900 of file `mpcotool.c`.

```

01901 {
01902     GThread *thread[nthreads_gradient];
01903     ParallelData data[nthreads_gradient];
01904     unsigned int i, j, k, b;
01905     #if DEBUG
01906     fprintf (stderr, "calibrate_step_gradient: start\n");
01907     #endif
01908     for (i = 0; i < calibrate->nestimates; ++i)
01909     {
01910         k = (simulation + i) * calibrate->nvariables;
01911         b = calibrate->simulation_best[0] * calibrate->
01912             nvariables;
01913         #if DEBUG
01914         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01915                 simulation + i, calibrate->simulation_best[0]);
01916         #endif
01917         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01918         {
01919             #if DEBUG
01920             fprintf (stderr,
01921                     "calibrate_step_gradient: estimate=%u best=%u=%.14le\n",
01922                     i, j, calibrate->value[b]);
01923             #endif
01924             calibrate->value[k]
01925                 = calibrate->value[b] + calibrate_estimate_gradient (j
01926                 , i);
01927             calibrate->value[k] = fmin (fmax (calibrate->
01928                 value[k],
01929                                     calibrate->rangeminabs[j]),
01930                                     calibrate->rangemaxabs[j]);
01931             #if DEBUG
01932             fprintf (stderr,
01933                     "calibrate_step_gradient: estimate=%u variable=%u=%.14le\n",
01934                     i, j, calibrate->value[k]);
01935             #endif
01936         }
01937     }
01938     if (nthreads_gradient == 1)
01939         calibrate_gradient_sequential (simulation);
01940     else
01941     {
01942         for (i = 0; i <= nthreads_gradient; ++i)
01943         {
01944             calibrate->thread_gradient[i]
01945                 = simulation + calibrate->nstart_gradient
01946                 + i * (calibrate->nend_gradient - calibrate->
01947                     nstart_gradient)
01948                 / nthreads_gradient;
01949             #if DEBUG
01950             fprintf (stderr,
01951                     "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01952                     i, calibrate->thread_gradient[i]);
01953             #endif
01954         }
01955         for (i = 0; i < nthreads_gradient; ++i)
01956         {

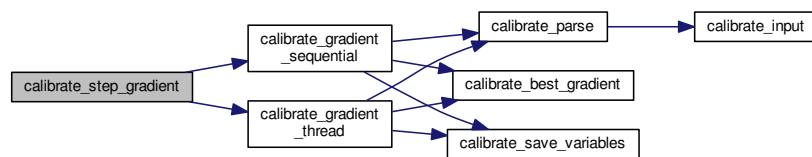
```

```

01953         data[i].thread = i;
01954         thread[i] = g_thread_new
01955             (NULL, (void (*)(void*)) calibrate_gradient_thread, &data[i]);
01956     }
01957     for (i = 0; i < nthreads_gradient; ++i)
01958         g_thread_join (thread[i]);
01959 }
01960 #if DEBUG
01961 fprintf (stderr, "calibrate_step_gradient: end\n");
01962 #endif
01963 }

```

Here is the call graph for this function:



#### 5.5.2.13 void \*calibrate\_thread ( ParallelData \*data )

Function to calibrate on a thread.

##### Parameters

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

##### Returns

NULL

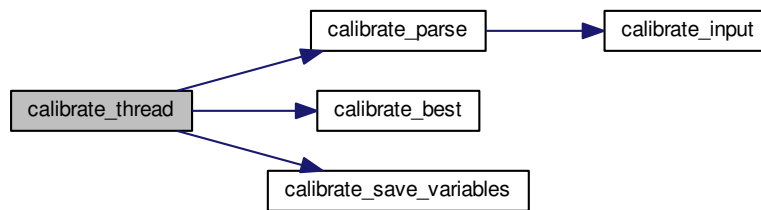
Definition at line 1494 of file [mpcotool.c](#).

```

01495 {
01496     unsigned int i, j, thread;
01497     double e;
01498     #if DEBUG
01499         fprintf (stderr, "calibrate_thread: start\n");
01500     #endif
01501     thread = data->thread;
01502     #if DEBUG
01503         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01504             calibrate->thread[thread], calibrate->thread[thread + 1]);
01505     #endif
01506     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01507     {
01508         e = 0.;
01509         for (j = 0; j < calibrate->nexperiments; ++j)
01510             e += calibrate_parse (i, j);
01511         g_mutex_lock (mutex);
01512         calibrate_best (i, e);
01513         calibrate_save_variables (i, e);
01514         g_mutex_unlock (mutex);
01515     #if DEBUG
01516         fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01517     #endif
01518     }
01519     #if DEBUG
01520         fprintf (stderr, "calibrate_thread: end\n");
01521     #endif
01522     g_thread_exit (NULL);
01523     return NULL;
01524 }

```

Here is the call graph for this function:



#### 5.5.2.14 int cores\_number ( )

Function to obtain the cores number.

##### Returns

Cores number.

Definition at line 4707 of file [mpcotool.c](#).

```

04708 {
04709     #ifdef G_OS_WIN32
04710         SYSTEM_INFO sysinfo;
04711         GetSystemInfo (&sysinfo);
04712         return sysinfo.dwNumberOfProcessors;
04713     #else
04714         return (int) sysconf ( _SC_NPROCESSORS_ONLN );
04715     #endif
04716 }
  
```

#### 5.5.2.15 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 488 of file [mpcotool.c](#).

```

00489 {
00490     char buffer2[64];
00491     char *buffert[MAX_NINPUTS] =
00492         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493     xmlDoc *doc;
00494     xmlNode *node, *child;
00495     xmlChar *buffer;
00496     char *msg;
00497     int error_code;
00498     unsigned int i;
00499
00500     #if DEBUG
00501         fprintf (stderr, "input_open: start\n");
00502     #endif
  
```

```

00503
00504 // Resetting input data
00505 buffer = NULL;
00506 input_new ();
00507
00508 // Parsing the input file
00509 #if DEBUG
00510 fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00511 #endif
00512 doc = xmlParseFile (filename);
00513 if (!doc)
00514 {
00515     msg = gettext ("Unable to parse the input file");
00516     goto exit_on_error;
00517 }
00518
00519 // Getting the root node
00520 #if DEBUG
00521 fprintf (stderr, "input_open: getting the root node\n");
00522 #endif
00523 node = xmlDocGetRootElement (doc);
00524 if (xmlStrcmp (node->name, XML_CALIBRATE))
00525 {
00526     msg = gettext ("Bad root XML node");
00527     goto exit_on_error;
00528 }
00529
00530 // Getting results file names
00531 input->result = (char *) xmlGetProp (node, XML_RESULT);
00532 if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535 if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538 // Opening simulator program name
00539 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540 if (!input->simulator)
00541 {
00542     msg = gettext ("Bad simulator program");
00543     goto exit_on_error;
00544 }
00545
00546 // Opening evaluator program name
00547 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549 // Obtaining pseudo-random numbers generator seed
00550 if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552 else
00553 {
00554     input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555     if (error_code)
00556     {
00557         msg = gettext ("Bad pseudo-random numbers generator seed");
00558         goto exit_on_error;
00559     }
00560 }
00561
00562 // Opening algorithm
00563 buffer = xmlGetProp (node, XML_ALGORITHM);
00564 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565 {
00566     input->algorithm = ALGORITHM_MONTE_CARLO;
00567
00568     // Obtaining simulations number
00569     input->nsimulations
00570     = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571     if (error_code)
00572     {
00573         msg = gettext ("Bad simulations number");
00574         goto exit_on_error;
00575     }
00576 }
00577 else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     input->algorithm = ALGORITHM_SWEEP;
00579 else if (!xmlStrcmp (buffer, XML_GENETIC))
00580 {
00581     input->algorithm = ALGORITHM_GENETIC;
00582
00583     // Obtaining population
00584     if (xmlHasProp (node, XML_NPOPULATION))
00585     {
00586         input->nsimulations
00587         = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00588         if (error_code || input->nsimulations < 3)
00589         {

```

```

00590         msg = gettext ("Invalid population number");
00591         goto exit_on_error;
00592     }
00593 }
00594 else
00595 {
00596     msg = gettext ("No population number");
00597     goto exit_on_error;
00598 }
00599
00600 // Obtaining generations
00601 if (xmlHasProp (node, XML_NGENERATIONS))
00602 {
00603     input->niterations
00604     = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00605     if (error_code || !input->niterations)
00606     {
00607         msg = gettext ("Invalid generations number");
00608         goto exit_on_error;
00609     }
00610 }
00611 else
00612 {
00613     msg = gettext ("No generations number");
00614     goto exit_on_error;
00615 }
00616
00617 // Obtaining mutation probability
00618 if (xmlHasProp (node, XML_MUTATION))
00619 {
00620     input->mutation_ratio
00621     = xml_node_get_float (node, XML_MUTATION, &error_code);
00622     if (error_code || input->mutation_ratio < 0.
00623         || input->mutation_ratio >= 1.)
00624     {
00625         msg = gettext ("Invalid mutation probability");
00626         goto exit_on_error;
00627     }
00628 }
00629 else
00630 {
00631     msg = gettext ("No mutation probability");
00632     goto exit_on_error;
00633 }
00634
00635 // Obtaining reproduction probability
00636 if (xmlHasProp (node, XML_REPRODUCTION))
00637 {
00638     input->reproduction_ratio
00639     = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00640     if (error_code || input->reproduction_ratio < 0.
00641         || input->reproduction_ratio >= 1.0)
00642     {
00643         msg = gettext ("Invalid reproduction probability");
00644         goto exit_on_error;
00645     }
00646 }
00647 else
00648 {
00649     msg = gettext ("No reproduction probability");
00650     goto exit_on_error;
00651 }
00652
00653 // Obtaining adaptation probability
00654 if (xmlHasProp (node, XML_ADAPTATION))
00655 {
00656     input->adaptation_ratio
00657     = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00658     if (error_code || input->adaptation_ratio < 0.
00659         || input->adaptation_ratio >= 1.)
00660     {
00661         msg = gettext ("Invalid adaptation probability");
00662         goto exit_on_error;
00663     }
00664 }
00665 else
00666 {
00667     msg = gettext ("No adaptation probability");
00668     goto exit_on_error;
00669 }
00670
00671 // Checking survivals
00672 i = input->mutation_ratio * input->nsimulations;
00673 i += input->reproduction_ratio * input->
nsimulations;
00674 i += input->adaptation_ratio * input->
nsimulations;

```



```

00675     if (i > input->nsimulations - 2)
00676     {
00677         msg = gettext
00678             ("No enough survival entities to reproduce the population");
00679         goto exit_on_error;
00680     }
00681 }
00682 else
00683 {
00684     msg = gettext ("Unknown algorithm");
00685     goto exit_on_error;
00686 }
00687 xmlFree (buffer);
00688 buffer = NULL;
00689
00690 if (input->algorithm == ALGORITHM_MONTE_CARLO
00691     || input->algorithm == ALGORITHM_SWEEP)
00692 {
00693     // Obtaining iterations number
00694     input->niterations
00695         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00696     if (error_code == 1)
00697         input->niterations = 1;
00698     else if (error_code)
00699     {
00700         msg = gettext ("Bad iterations number");
00701         goto exit_on_error;
00702     }
00703 }
00704
00705 // Obtaining best number
00706 if (xmlHasProp (node, XML_NBEST))
00707 {
00708     input->nbest = xml_node_get_uint (node,
00709 XML_NBEST, &error_code);
00710     if (error_code || !input->nbest)
00711     {
00712         msg = gettext ("Invalid best number");
00713         goto exit_on_error;
00714     }
00715 }
00716 else
00717     input->nbest = 1;
00718
00719 // Obtaining tolerance
00720 if (xmlHasProp (node, XML_TOLERANCE))
00721 {
00722     input->tolerance
00723         = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00724     if (error_code || input->tolerance < 0.)
00725     {
00726         msg = gettext ("Invalid tolerance");
00727         goto exit_on_error;
00728     }
00729 }
00730 else
00731     input->tolerance = 0.;
00732
00733 // Getting gradient method parameters
00734 if (xmlHasProp (node, XML_NSTEPS))
00735 {
00736     input->nsteps = xml_node_get_uint (node,
00737 XML_NSTEPS, &error_code);
00738     if (error_code || !input->nsteps)
00739     {
00740         msg = gettext ("Invalid steps number");
00741         goto exit_on_error;
00742     }
00743     buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00744     if (!xmlStrcmp (buffer, XML_COORDINATES))
00745         input->gradient_method =
00746 GRADIENT_METHOD_COORDINATES;
00747     else if (!xmlStrcmp (buffer, XML_RANDOM))
00748     {
00749         input->gradient_method =
00750 GRADIENT_METHOD_RANDOM;
00751         input->nestimates
00752             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00753         if (error_code || !input->nestimates)
00754         {
00755             msg = gettext ("Invalid estimates number");
00756             goto exit_on_error;
00757         }
00758     }
00759 }
00760 else
00761 {
00762     msg = gettext ("Unknown method to estimate the gradient");

```

```

00758         goto exit_on_error;
00759     }
00760     xmlFree (buffer);
00761     buffer = NULL;
00762     if (xmlHasProp (node, XML_RELAXATION))
00763     {
00764         input->relaxation
00765         = xml_node_get_float (node, XML_RELAXATION, &error_code);
00766         if (error_code || input->relaxation < 0.
00767             || input->relaxation > 2.)
00768         {
00769             msg = gettext ("Invalid relaxation parameter");
00770             goto exit_on_error;
00771         }
00772     }
00773     else
00774         input->relaxation = DEFAULT_RELAXATION;
00775 }
00776 else
00777     input->nsteps = 0;
00778 }
00779
00780 // Reading the experimental data
00781 for (child = node->children; child; child = child->next)
00782 {
00783     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00784         break;
00785 #if DEBUG
00786     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00787 #endif
00788     if (xmlHasProp (child, XML_NAME))
00789         buffer = xmlGetProp (child, XML_NAME);
00790     else
00791     {
00792         snprintf (buffer2, 64, "%s %u: %s",
00793                 gettext ("Experiment"),
00794                 input->nexperiments + 1, gettext ("no data file name"));
00795         msg = buffer2;
00796         goto exit_on_error;
00797     }
00798 #if DEBUG
00799     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00800 #endif
00801     input->weight = g_realloc (input->weight,
00802                               (1 + input->nexperiments) * sizeof (double));
00803     if (xmlHasProp (child, XML_WEIGHT))
00804     {
00805         input->weight[input->nexperiments]
00806         = xml_node_get_float (child, XML_WEIGHT, &error_code);
00807         if (error_code)
00808         {
00809             snprintf (buffer2, 64, "%s %s: %s",
00810                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00811             msg = buffer2;
00812             goto exit_on_error;
00813         }
00814     }
00815     else
00816         input->weight[input->nexperiments] = 1.;
00817 #if DEBUG
00818     fprintf (stderr, "input_open: weight=%lg\n",
00819             input->weight[input->nexperiments]);
00820 #endif
00821     if (!input->nexperiments)
00822         input->ninputs = 0;
00823 #if DEBUG
00824     fprintf (stderr, "input_open: template[0]\n");
00825 #endif
00826     if (xmlHasProp (child, XML_TEMPLATE1))
00827     {
00828         input->template[0]
00829         = (char **) g_realloc (input->template[0],
00830                               (1 + input->nexperiments) * sizeof (char *));
00831         buffert[0] = (char *) xmlGetProp (child, template[0]);
00832 #if DEBUG
00833         fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00834                 input->nexperiments, buffert[0]);
00835 #endif
00836         if (!input->nexperiments)
00837             ++input->ninputs;
00838 #if DEBUG
00839         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00840 #endif
00841     }
00842     else
00843     {
00844         snprintf (buffer2, 64, "%s %s: %s",

```

```

00845         gettext ("Experiment"), buffer, gettext ("no template"));
00846         msg = buffer2;
00847         goto exit_on_error;
00848     }
00849     for (i = 1; i < MAX_NINPUTS; ++i)
00850     {
00851     #if DEBUG
00852         fprintf (stderr, "input_open: template%u\n", i + 1);
00853     #endif
00854         if (xmlHasProp (child, template[i]))
00855         {
00856             if (input->nexperiments && input->ninputs <= i)
00857             {
00858                 snprintf (buffer2, 64, "%s %s: %s",
00859                     gettext ("Experiment"),
00860                     buffer, gettext ("bad templates number"));
00861                 msg = buffer2;
00862                 while (i-- > 0)
00863                     xmlFree (buffert[i]);
00864                 goto exit_on_error;
00865             }
00866             input->template[i] = (char **)
00867                 g_realloc (input->template[i],
00868                     (1 + input->nexperiments) * sizeof (char *));
00869             buffert[i] = (char *) xmlGetProp (child, template[i]);
00870     #if DEBUG
00871             fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00872                 input->nexperiments, i + 1,
00873                 input->template[i][input->nexperiments]);
00874     #endif
00875             if (!input->nexperiments)
00876                 ++input->ninputs;
00877     #if DEBUG
00878             fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00879     #endif
00880         }
00881         else if (input->nexperiments && input->ninputs >= i)
00882         {
00883             snprintf (buffer2, 64, "%s %s: %s%u",
00884                 gettext ("Experiment"),
00885                 buffer, gettext ("no template"), i + 1);
00886             msg = buffer2;
00887             while (i-- > 0)
00888                 xmlFree (buffert[i]);
00889             goto exit_on_error;
00890         }
00891         else
00892             break;
00893     }
00894     input->experiment
00895     = g_realloc (input->experiment,
00896         (1 + input->nexperiments) * sizeof (char *));
00897     input->experiment[input->nexperiments] = (char *) buffer;
00898     for (i = 0; i < input->ninputs; ++i)
00899         input->template[i][input->nexperiments] = buffert[i];
00900     ++input->nexperiments;
00901     #if DEBUG
00902     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00903     #endif
00904     }
00905     if (!input->nexperiments)
00906     {
00907         msg = gettext ("No calibration experiments");
00908         goto exit_on_error;
00909     }
00910     buffer = NULL;
00911
00912     // Reading the variables data
00913     for (; child; child = child->next)
00914     {
00915         if (xmlStrcmp (child->name, XML_VARIABLE))
00916         {
00917             snprintf (buffer2, 64, "%s %u: %s",
00918                 gettext ("Variable"),
00919                 input->nvariables + 1, gettext ("bad XML node"));
00920             msg = buffer2;
00921             goto exit_on_error;
00922         }
00923         if (xmlHasProp (child, XML_NAME))
00924             buffer = xmlGetProp (child, XML_NAME);
00925         else
00926         {
00927             snprintf (buffer2, 64, "%s %u: %s",
00928                 gettext ("Variable"),
00929                 input->nvariables + 1, gettext ("no name"));
00930             msg = buffer2;
00931             goto exit_on_error;

```

```

00932     }
00933     if (xmlHasProp (child, XML_MINIMUM))
00934     {
00935         input->rangemin = g_realloc
00936             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00937         input->rangeminabs = g_realloc
00938             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00939         input->rangemin[input->nvariables]
00940             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00941         if (error_code)
00942         {
00943             snprintf (buffer2, 64, "%s %s: %s",
00944                 gettext ("Variable"), buffer, gettext ("bad minimum"));
00945             msg = buffer2;
00946             goto exit_on_error;
00947         }
00948         if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00949         {
00950             input->rangeminabs[input->nvariables]
00951                 = xml_node_get_float (child,
XML_ABSOLUTE_MINIMUM, &error_code);
00952             if (error_code)
00953             {
00954                 snprintf (buffer2, 64, "%s %s: %s",
00955                     gettext ("Variable"),
00956                     buffer, gettext ("bad absolute minimum"));
00957                 msg = buffer2;
00958                 goto exit_on_error;
00959             }
00960         }
00961         else
00962             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00963         if (input->rangemin[input->nvariables]
00964             < input->rangeminabs[input->nvariables])
00965         {
00966             snprintf (buffer2, 64, "%s %s: %s",
00967                 gettext ("Variable"),
00968                 buffer, gettext ("minimum range not allowed"));
00969             msg = buffer2;
00970             goto exit_on_error;
00971         }
00972     }
00973     else
00974     {
00975         snprintf (buffer2, 64, "%s %s: %s",
00976             gettext ("Variable"), buffer, gettext ("no minimum range"));
00977         msg = buffer2;
00978         goto exit_on_error;
00979     }
00980     if (xmlHasProp (child, XML_MAXIMUM))
00981     {
00982         input->rangemax = g_realloc
00983             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00984         input->rangemaxabs = g_realloc
00985             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00986         input->rangemax[input->nvariables]
00987             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00988         if (error_code)
00989         {
00990             snprintf (buffer2, 64, "%s %s: %s",
00991                 gettext ("Variable"), buffer, gettext ("bad maximum"));
00992             msg = buffer2;
00993             goto exit_on_error;
00994         }
00995         if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00996         {
00997             input->rangemaxabs[input->nvariables]
00998                 = xml_node_get_float (child,
XML_ABSOLUTE_MAXIMUM, &error_code);
00999             if (error_code)
01000             {
01001                 snprintf (buffer2, 64, "%s %s: %s",
01002                     gettext ("Variable"),
01003                     buffer, gettext ("bad absolute maximum"));
01004                 msg = buffer2;
01005                 goto exit_on_error;
01006             }
01007         }
01008         else
01009             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01010         if (input->rangemax[input->nvariables]
01011             > input->rangemaxabs[input->nvariables])
01012         {
01013             snprintf (buffer2, 64, "%s %s: %s",
01014                 gettext ("Variable"),
01015                 buffer, gettext ("maximum range not allowed"));
01016             msg = buffer2;

```

```

01017         goto exit_on_error;
01018     }
01019 }
01020 else
01021 {
01022     snprintf (buffer2, 64, "%s %s: %s",
01023             gettext ("Variable"), buffer, gettext ("no maximum range"));
01024     msg = buffer2;
01025     goto exit_on_error;
01026 }
01027 if (input->rangemax[input->nvariables]
01028     < input->rangemin[input->nvariables])
01029 {
01030     snprintf (buffer2, 64, "%s %s: %s",
01031             gettext ("Variable"), buffer, gettext ("bad range"));
01032     msg = buffer2;
01033     goto exit_on_error;
01034 }
01035 input->precision = g_realloc
01036 (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01037 if (xmlHasProp (child, XML_PRECISION))
01038 {
01039     input->precision[input->nvariables]
01040     = xml_node_get_uint (child, XML_PRECISION, &error_code);
01041     if (error_code || input->precision[input->
01042 nvariables] >= NPRECISIONS)
01043     {
01044         snprintf (buffer2, 64, "%s %s: %s",
01045                 gettext ("Variable"),
01046                 buffer, gettext ("bad precision"));
01047         msg = buffer2;
01048         goto exit_on_error;
01049     }
01050 }
01051 else
01052     input->precision[input->nvariables] =
01053     DEFAULT_PRECISION;
01054 if (input->algorithm == ALGORITHM_SWEEP)
01055 {
01056     if (xmlHasProp (child, XML_NSWEEPS))
01057     {
01058         input->nsweeps = (unsigned int *)
01059         g_realloc (input->nsweeps,
01060             (1 + input->nvariables) * sizeof (unsigned int));
01061         input->nsweeps[input->nvariables]
01062         = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01063         if (error_code || !input->nsweeps[input->
01064 nvariables])
01065         {
01066             snprintf (buffer2, 64, "%s %s: %s",
01067                     gettext ("Variable"),
01068                     buffer, gettext ("bad sweeps"));
01069             msg = buffer2;
01070             goto exit_on_error;
01071         }
01072     }
01073     else
01074     {
01075         snprintf (buffer2, 64, "%s %s: %s",
01076                 gettext ("Variable"),
01077                 buffer, gettext ("no sweeps number"));
01078         msg = buffer2;
01079         goto exit_on_error;
01080     }
01081 }
01082 #if DEBUG
01083 fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01084         input->nsweeps[input->nvariables],
01085         input->nsimulations);
01086 #endif
01087 }
01088 if (input->algorithm == ALGORITHM_GENETIC)
01089 {
01090     // Obtaining bits representing each variable
01091     if (xmlHasProp (child, XML_NBITS))
01092     {
01093         input->nbits = (unsigned int *)
01094         g_realloc (input->nbits,
01095             (1 + input->nvariables) * sizeof (unsigned int));
01096         i = xml_node_get_uint (child, XML_NBITS, &error_code);
01097         if (error_code || !i)
01098         {
01099             snprintf (buffer2, 64, "%s %s: %s",
01100                     gettext ("Variable"),
01101                     buffer, gettext ("invalid bits number"));
01102             msg = buffer2;
01103             goto exit_on_error;
01104         }
01105     }
01106 }

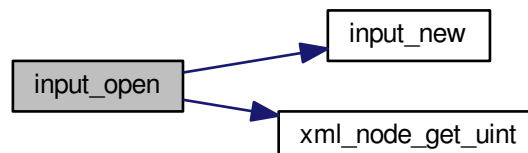
```

```

01100         input->nbits[input->nvariables] = i;
01101     }
01102     else
01103     {
01104         snprintf (buffer2, 64, "%s %s: %s",
01105                 gettext ("Variable"),
01106                 buffer, gettext ("no bits number"));
01107         msg = buffer2;
01108         goto exit_on_error;
01109     }
01110 }
01111 else if (input->nsteps)
01112 {
01113     input->step = (double *)
01114         g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01115     input->step[input->nvariables]
01116         = xml_node_get_float (child, XML_STEP, &error_code);
01117     if (error_code || input->step[input->nvariables] < 0.)
01118     {
01119         snprintf (buffer2, 64, "%s %s: %s",
01120                 gettext ("Variable"),
01121                 buffer, gettext ("bad step size"));
01122         msg = buffer2;
01123         goto exit_on_error;
01124     }
01125 }
01126 input->label = g_realloc
01127     (input->label, (1 + input->nvariables) * sizeof (char *));
01128 input->label[input->nvariables] = (char *) buffer;
01129 ++input->nvariables;
01130 }
01131 if (!input->nvariables)
01132 {
01133     msg = gettext ("No calibration variables");
01134     goto exit_on_error;
01135 }
01136 buffer = NULL;
01137
01138 // Getting the working directory
01139 input->directory = g_path_get_dirname (filename);
01140 input->name = g_path_get_basename (filename);
01141
01142 // Closing the XML document
01143 xmlFreeDoc (doc);
01144
01145 #if DEBUG
01146 fprintf (stderr, "input_open: end\n");
01147 #endif
01148 return 1;
01149
01150 exit_on_error:
01151 xmlFree (buffer);
01152 xmlFreeDoc (doc);
01153 show_error (msg);
01154 input_free ();
01155 #if DEBUG
01156 fprintf (stderr, "input_open: end\n");
01157 #endif
01158 return 0;
01159 }

```

Here is the call graph for this function:



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

Function to save the input file.

## Parameters

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

Definition at line 2670 of file `mpcotool.c`.

```

02671 {
02672     unsigned int i, j;
02673     char *buffer;
02674     xmlDoc *doc;
02675     xmlNode *node, *child;
02676     GFile *file, *file2;
02677
02678     // Getting the input file directory
02679     input->name = g_path_get_basename (filename);
02680     input->directory = g_path_get_dirname (filename);
02681     file = g_file_new_for_path (input->directory);
02682
02683     // Opening the input file
02684     doc = xmlNewDoc ((const xmlChar *) "1.0");
02685
02686     // Setting root XML node
02687     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02688     xmlDocSetRootElement (doc, node);
02689
02690     // Adding properties to the root XML node
02691     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02692         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02693     if (xmlStrcmp ((const xmlChar *) input->variables,
02694         variables_name))
02695         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02696         variables);
02697     file2 = g_file_new_for_path (input->simulator);
02698     buffer = g_file_get_relative_path (file, file2);
02699     g_object_unref (file2);
02700     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02701     g_free (buffer);
02702     if (input->evaluator)
02703     {
02704         file2 = g_file_new_for_path (input->evaluator);
02705         buffer = g_file_get_relative_path (file, file2);
02706         g_object_unref (file2);
02707         if (xmlStrlen ((xmlChar *) buffer))
02708             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02709         g_free (buffer);
02710     }
02711     if (input->seed != DEFAULT_RANDOM_SEED)
02712         xml_node_set_uint (node, XML_SEED, input->seed);
02713
02714     // Setting the algorithm
02715     buffer = (char *) g_malloc (64);
02716     switch (input->algorithm)
02717     {
02718     case ALGORITHM_MONTE_CARLO:
02719         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02720         snprintf (buffer, 64, "%u", input->nsimulations);
02721         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02722         snprintf (buffer, 64, "%u", input->niterations);
02723         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02724         snprintf (buffer, 64, "%.3lg", input->tolerance);
02725         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02726         snprintf (buffer, 64, "%u", input->nbest);
02727         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02728         input_save_gradient (node);
02729         break;
02730     case ALGORITHM_SWEEP:
02731         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02732         snprintf (buffer, 64, "%u", input->niterations);
02733         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02734         snprintf (buffer, 64, "%.3lg", input->tolerance);
02735         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02736         snprintf (buffer, 64, "%u", input->nbest);
02737         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02738         input_save_gradient (node);
02739         break;
02740     default:
02741         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02742         snprintf (buffer, 64, "%u", input->nsimulations);
02743         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02744         snprintf (buffer, 64, "%u", input->niterations);
02745         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02746         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02747         xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02748         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02749         xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02750         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);

```

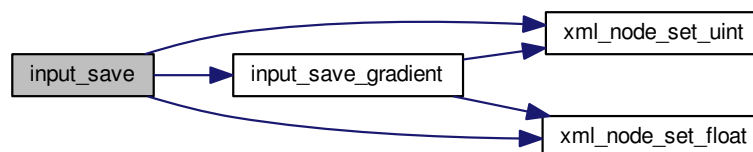


```

02749     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02750     break;
02751 }
02752 g_free (buffer);
02753
02754 // Setting the experimental data
02755 for (i = 0; i < input->nexperiments; ++i)
02756 {
02757     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02758     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02759     if (input->weight[i] != 1.)
02760         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02761     for (j = 0; j < input->ninputs; ++j)
02762         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02763 }
02764
02765 // Setting the variables data
02766 for (i = 0; i < input->nvariables; ++i)
02767 {
02768     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02769     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02770     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02771     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02772         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02773     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02774     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02775         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02776     if (input->precision[i] != DEFAULT_PRECISION)
02777         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02778     if (input->algorithm == ALGORITHM_SWEEP)
02779         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02780     else if (input->algorithm == ALGORITHM_GENETIC)
02781         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02782 }
02783
02784 // Saving the XML file
02785 xmlSaveFormatFile (filename, doc, 1);
02786
02787 // Freeing memory
02788 xmlFreeDoc (doc);
02789 }

```

Here is the call graph for this function:



#### 5.5.2.17 void input\_save\_gradient ( xmlNode \* node )

Function to save the gradient based method data in a XML node.

## Parameters

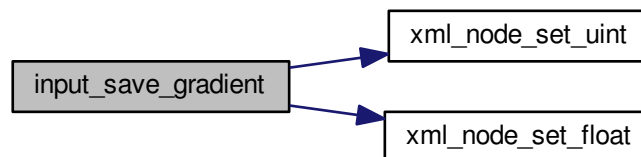
<i>node</i>	XML node.
-------------	-----------

Definition at line 2644 of file [mpcotool.c](#).

```

02645 {
02646     if (input->nsteps)
02647     {
02648         xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
02649         if (input->relaxation != DEFAULT_RELAXATION)
02650             xml_node_set_float (node, XML_RELAXATION,
input->relaxation);
02651         switch (input->gradient_method)
02652         {
02653             case GRADIENT_METHOD_COORDINATES:
02654                 xmlSetProp (node, XML_GRADIENT_METHOD,
XML_COORDINATES);
02655                 break;
02656             default:
02657                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02658                 xml_node_set_uint (node, XML_NESTIMATES,
input->nestimates);
02659         }
02660     }
02661 }
```

Here is the call graph for this function:



### 5.5.2.18 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 4728 of file [mpcotool.c](#).

```

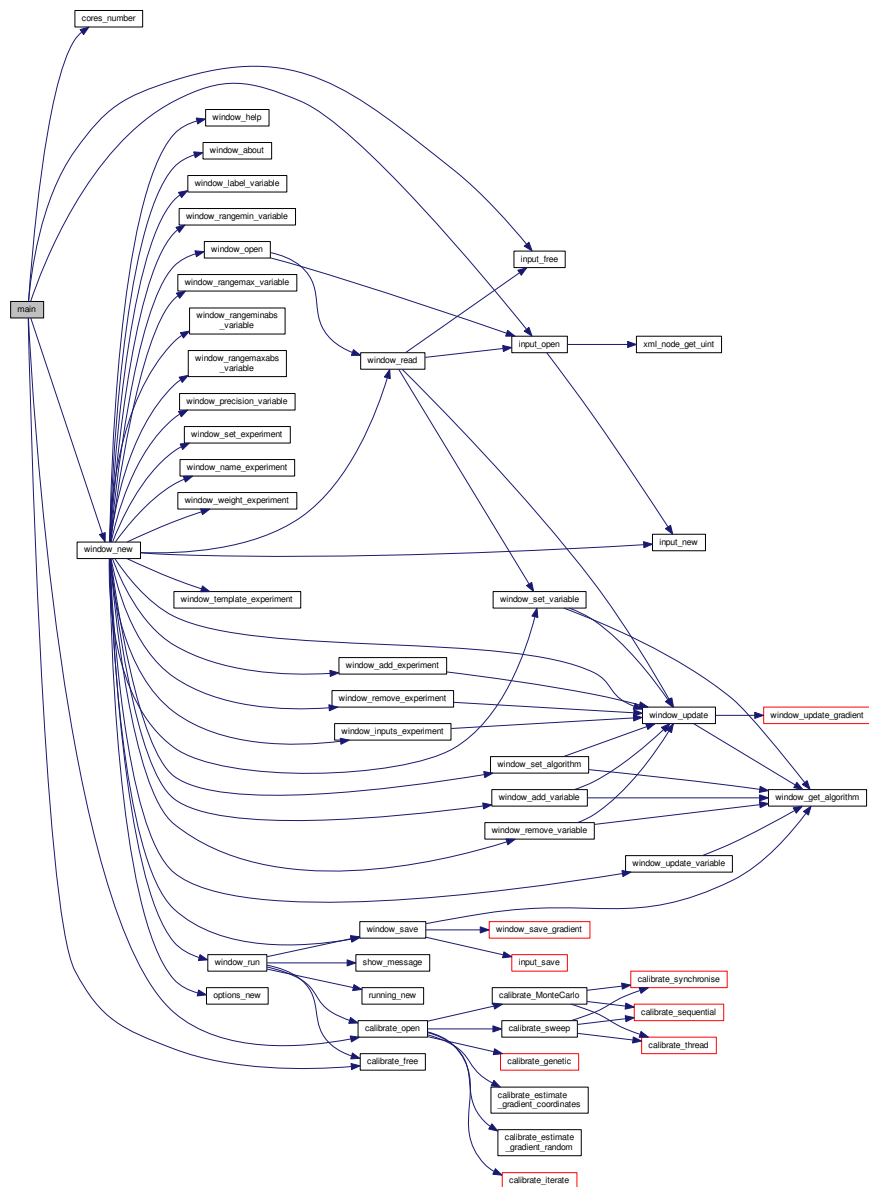
04729 {
04730     #if HAVE_GTK
04731     char *buffer;
04732     #endif
04733
04734     // Starting pseudo-random numbers generator
04735     calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04736     calibrate->seed = DEFAULT_RANDOM_SEED;
04737 }
```

```

04738 // Allowing spaces in the XML data file
04739 xmlKeepBlanksDefault (0);
04740
04741 // Starting MPI
04742 #if HAVE_MPI
04743 MPI_Init (&argn, &argc);
04744 MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04745 MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04746 printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04747 #else
04748 ntasks = 1;
04749 #endif
04750
04751 #if HAVE_GTK
04752
04753 // Getting threads number
04754 nthreads_gradient = nthreads = cores_number ();
04755
04756 // Setting local language and international floating point numbers notation
04757 setlocale (LC_ALL, "");
04758 setlocale (LC_NUMERIC, "C");
04759 window->application_directory = g_get_current_dir ();
04760 buffer = g_build_filename (window->application_directory,
    LOCALE_DIR, NULL);
04761 bindtextdomain (PROGRAM_INTERFACE, buffer);
04762 bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04763 textdomain (PROGRAM_INTERFACE);
04764
04765 // Initing GTK+
04766 gtk_disable_setlocale ();
04767 gtk_init (&argn, &argc);
04768
04769 // Opening the main window
04770 window_new ();
04771 gtk_main ();
04772
04773 // Freeing memory
04774 input_free ();
04775 g_free (buffer);
04776 gtk_widget_destroy (GTK_WIDGET (window->window));
04777 g_free (window->application_directory);
04778
04779 #else
04780
04781 // Checking syntax
04782 if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04783 {
04784     printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04785     return 1;
04786 }
04787
04788 // Getting threads number
04789 if (argn == 2)
04790     nthreads_gradient = nthreads = cores_number ();
04791 else
04792 {
04793     nthreads_gradient = nthreads = atoi (argc[2]);
04794     if (!nthreads)
04795     {
04796         printf ("Bad threads number\n");
04797         return 2;
04798     }
04799 }
04800 printf ("nthreads=%u\n", nthreads);
04801
04802 // Making calibration
04803 if (input_open (argc[argn - 1]))
04804     calibrate_open ();
04805
04806 // Freeing memory
04807 calibrate_free ();
04808
04809 #endif
04810
04811 // Closing MPI
04812 #if HAVE_MPI
04813 MPI_Finalize ();
04814 #endif
04815
04816 // Freeing memory
04817 gsl_rng_free (calibrate->rng);
04818
04819 // Closing
04820 return 0;
04821 }

```

Here is the call graph for this function:



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

Function to show a dialog with an error message.

#### Parameters

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

Definition at line 256 of file [mpcotool.c](#).

```
00257 {
00258     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00259 }
```

Here is the call graph for this function:



#### 5.5.2.20 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 226 of file [mpcotool.c](#).

```

00227 {
00228     #if HAVE_GTK
00229         GtkMessageDialog *dlg;
00230
00231         // Creating the dialog
00232         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00233             (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00234
00235         // Setting the dialog title
00236         gtk_window_set_title (GTK_WINDOW (dlg), title);
00237
00238         // Showing the dialog and waiting response
00239         gtk_dialog_run (GTK_DIALOG (dlg));
00240
00241         // Closing and freeing memory
00242         gtk_widget_destroy (GTK_WIDGET (dlg));
00243
00244     #else
00245         printf ("%s: %s\n", title, msg);
00246     #endif
00247 }
  
```

#### 5.5.2.21 int window\_get\_algorithm ( )

Function to get the stochastic algorithm number.

##### Returns

Stochastic algorithm number.

Definition at line 2887 of file [mpcotool.c](#).

```

02888 {
02889     unsigned int i;
02890     for (i = 0; i < NALGORITHMS; ++i)
02891         if (gtk_toggle_button_get_active
02892             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02893             break;
02894     return i;
02895 }
  
```

### 5.5.2.22 int window\_get\_gradient ( )

Function to get the gradient base method number.

#### Returns

Gradient base method number.

Definition at line 2903 of file [mpcotool.c](#).

```
02904 {
02905     unsigned int i;
02906     for (i = 0; i < NGRADIENTS; ++i)
02907         if (gtk_toggle_button_get_active
02908             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02909             break;
02910     return i;
02911 }
```

### 5.5.2.23 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 3908 of file [mpcotool.c](#).

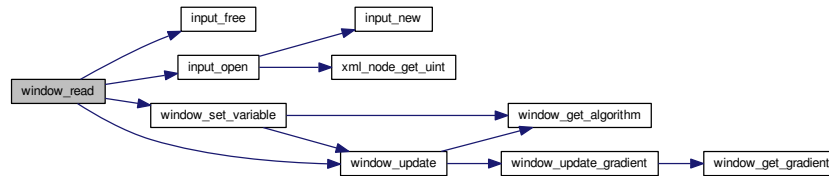
```
03909 {
03910     unsigned int i;
03911     char *buffer;
03912     #if DEBUG
03913     fprintf (stderr, "window_read: start\n");
03914     #endif
03915
03916     // Reading new input file
03917     input_free ();
03918     if (!input_open (filename))
03919         return 0;
03920
03921     // Setting GTK+ widgets data
03922     gtk_entry_set_text (window->entry_result, input->result);
03923     gtk_entry_set_text (window->entry_variables, input->
variables);
03924     buffer = g_build_filename (input->directory, input->
simulator, NULL);
03925     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
03926     g_free (buffer);
03927     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
03928     if (input->evaluator)
03929     {
03930         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
03931         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
03932         g_free (buffer);
03933     }
03934     gtk_toggle_button_set_active
03935     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
03936     switch (input->algorithm)
03937     {
03938     case ALGORITHM_MONTE_CARLO:
03939         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
03940     case ALGORITHM_SWEEP:
```

```

03945     gtk_spin_button_set_value (window->spin_iterations,
03946                               (gdouble) input->niterations);
03947     gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
03948     gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
03949     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
                                input->nsteps);
03950
03951     if (input->nsteps)
03952     {
03953         gtk_toggle_button_set_active
03954             (GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
03955         gtk_spin_button_set_value (window->spin_steps,
03956                                   (gdouble) input->nsteps);
03957         gtk_spin_button_set_value (window->spin_relaxation,
03958                                   (gdouble) input->relaxation);
03959         switch (input->gradient_method)
03960         {
03961             case GRADIENT_METHOD_RANDOM:
03962                 gtk_spin_button_set_value (window->spin_estimates,
03963                                             (gdouble) input->nestimates);
03964             }
03965         }
03966     }
03967     break;
03968 default:
03969     gtk_spin_button_set_value (window->spin_population,
03970                               (gdouble) input->nsimulations);
03971     gtk_spin_button_set_value (window->spin_generations,
03972                               (gdouble) input->niterations);
03973     gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
03974     gtk_spin_button_set_value (window->spin_reproduction,
03975                               input->reproduction_ratio);
03976     gtk_spin_button_set_value (window->spin_adaptation,
03977                               input->adaptation_ratio);
03978 }
03979 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03980 g_signal_handler_block (window->button_experiment,
03981                         window->id_experiment_name);
03982 gtk_combo_box_text_remove_all (window->combo_experiment);
03983 for (i = 0; i < input->nexperiments; ++i)
03984     gtk_combo_box_text_append_text (window->combo_experiment,
03985                                     input->experiment[i]);
03986 g_signal_handler_unblock
03987     (window->button_experiment, window->
id_experiment_name);
03988 g_signal_handler_unblock (window->combo_experiment,
03989                             window->id_experiment);
03989 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03990 g_signal_handler_block (window->combo_variable, window->
id_variable);
03991 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03992 gtk_combo_box_text_remove_all (window->combo_variable);
03993 for (i = 0; i < input->nvariables; ++i)
03994     gtk_combo_box_text_append_text (window->combo_variable,
03995                                     input->label[i]);
03995 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03996 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03997 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03998 window_set_variable ();
03999 window_update ();
04000
04001 #if DEBUG
04002 fprintf (stderr, "window_read: end\n");
04003 #endif
04004 return 1;
04005 }

```

Here is the call graph for this function:



### 5.5.2.24 int window\_save ( )

Function to save the input file.

#### Returns

1 on OK, 0 on Cancel.

Definition at line 2951 of file `mpcotool.c`.

```

02952 {
02953     char *buffer;
02954     GtkFileChooserDialog *dlg;
02955
02956     #if DEBUG
02957         fprintf (stderr, "window_save: start\n");
02958     #endif
02959
02960     // Opening the saving dialog
02961     dlg = (GtkFileChooserDialog *)
02962         gtk_file_chooser_dialog_new (gettext ("Save file"),
02963                                     window->window,
02964                                     GTK_FILE_CHOOSER_ACTION_SAVE,
02965                                     gettext ("_Cancel"),
02966                                     GTK_RESPONSE_CANCEL,
02967                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02968     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02969     buffer = g_build_filename (input->directory, input->name, NULL);
02970     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02971     g_free (buffer);
02972
02973     // If OK response then saving
02974     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02975     {
02976
02977         // Adding properties to the root XML node
02978         input->simulator = gtk_file_chooser_get_filename
02979             (GTK_FILE_CHOOSER (window->button_simulator));
02980         if (gtk_toggle_button_get_active
02981             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02982             input->evaluator = gtk_file_chooser_get_filename
02983                 (GTK_FILE_CHOOSER (window->button_evaluator));
02984         else
02985             input->evaluator = NULL;
02986         input->result
02987             = (char *) xmlStrdup ((const xmlChar *)
02988                                   gtk_entry_get_text (window->entry_result));
02989         input->variables
02990             = (char *) xmlStrdup ((const xmlChar *)
02991                                   gtk_entry_get_text (window->entry_variables));
02992
02993         // Setting the algorithm
02994         switch (window_get_algorithm ())
02995         {
02996             case ALGORITHM_MONTE_CARLO:
02997                 input->algorithm = ALGORITHM_MONTE_CARLO;
02998                 input->nsimulations
02999                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
03000                 input->niterations
03001                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002                 input->tolerance = gtk_spin_button_get_value (window->

```

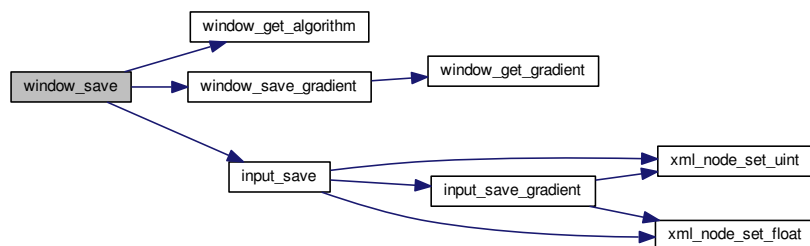


```

    spin_tolerance);
03003     input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
03004     window_save_gradient ();
03005     break;
03006     case ALGORITHM_SWEEP:
03007         input->algorithm = ALGORITHM_SWEEP;
03008         input->niterations
= gtk_spin_button_get_value_as_int (window->spin_iterations);
03009         input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
03011         input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
03012         window_save_gradient ();
03013         break;
03014     default:
03015         input->algorithm = ALGORITHM_GENETIC;
03016         input->nsimulations
= gtk_spin_button_get_value_as_int (window->spin_population);
03017         input->niterations
= gtk_spin_button_get_value_as_int (window->spin_generations);
03018         input->mutation_ratio
= gtk_spin_button_get_value (window->spin_mutation);
03019         input->reproduction_ratio
= gtk_spin_button_get_value (window->spin_reproduction);
03020         input->adaptation_ratio
= gtk_spin_button_get_value (window->spin_adaptation);
03021         break;
03022     }
03023     // Saving the XML file
03024     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03025     input_save (buffer);
03026     // Closing and freeing memory
03027     g_free (buffer);
03028     gtk_widget_destroy (GTK_WIDGET (dlg));
03029     #if DEBUG
03030     fprintf (stderr, "window_save: end\n");
03031     #endif
03032     return 1;
03033 }
03034 // Closing and freeing memory
03035 gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037 fprintf (stderr, "window_save: end\n");
03038 #endif
03039 return 0;
03040 }

```

Here is the call graph for this function:



#### 5.5.2.25 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 3544 of file [mpcotool.c](#).

```

03545 {
03546     unsigned int i, j;
03547     char *buffer;
03548     GFile *file1, *file2;
03549     #if DEBUG
03550     fprintf (stderr, "window_template_experiment: start\n");
03551     #endif
03552     i = (size_t) data;
03553     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03554     file1
03555     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03556     file2 = g_file_new_for_path (input->directory);
03557     buffer = g_file_get_relative_path (file2, file1);
03558     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03559     g_free (buffer);
03560     g_object_unref (file2);
03561     g_object_unref (file1);
03562     #if DEBUG
03563     fprintf (stderr, "window_template_experiment: end\n");
03564     #endif
03565 }
```

#### 5.5.2.26 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 336 of file [mpcotool.c](#).

```

00337 {
00338     double x = 0.;
00339     xmlChar *buffer;
00340     buffer = xmlGetProp (node, prop);
00341     if (!buffer)
00342         *error_code = 1;
00343     else
00344     {
00345         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346             *error_code = 2;
00347         else
00348             *error_code = 0;
00349         xmlFree (buffer);
00350     }
00351     return x;
00352 }
```

#### 5.5.2.27 int xml\_node\_get\_int ( xmlNode \* node, const xmlChar \* prop, int \* error\_code )

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

## Parameters

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

## Returns

Integer number value.

Definition at line 274 of file [mpcotool.c](#).

```
00275 {
00276     int i = 0;
00277     xmlChar *buffer;
00278     buffer = xmlGetProp (node, prop);
00279     if (!buffer)
00280         *error_code = 1;
00281     else
00282     {
00283         if (sscanf ((char *) buffer, "%d", &i) != 1)
00284             *error_code = 2;
00285         else
00286             *error_code = 0;
00287         xmlFree (buffer);
00288     }
00289     return i;
00290 }
```

**5.5.2.28** `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 305 of file [mpcotool.c](#).

```
00306 {
00307     unsigned int i = 0;
00308     xmlChar *buffer;
00309     buffer = xmlGetProp (node, prop);
00310     if (!buffer)
00311         *error_code = 1;
00312     else
00313     {
00314         if (sscanf ((char *) buffer, "%u", &i) != 1)
00315             *error_code = 2;
00316         else
00317             *error_code = 0;
00318         xmlFree (buffer);
00319     }
00320     return i;
00321 }
```

**5.5.2.29** `void xml_node_set_float ( xmlDoc * 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 403 of file [mpcotool.c](#).

```
00404 {
00405     xmlChar buffer[64];
00406     snprintf ((char *) buffer, 64, "%.14lg", value);
00407     xmlSetProp (node, prop, buffer);
00408 }
```

**5.5.2.30 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 365 of file [mpcotool.c](#).

```
00366 {
00367     xmlChar buffer[64];
00368     snprintf ((char *) buffer, 64, "%d", value);
00369     xmlSetProp (node, prop, buffer);
00370 }
```

**5.5.2.31 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 384 of file [mpcotool.c](#).

```
00385 {
00386     xmlChar buffer[64];
00387     snprintf ((char *) buffer, 64, "%u", value);
00388     xmlSetProp (node, prop, buffer);
00389 }
```

**5.5.3 Variable Documentation****5.5.3.1 const char\* format[NPRECISIONS]****Initial value:**

```
= {
    "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
    "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
}
```

Array of C-strings with variable formats.

Definition at line 117 of file [mpcotool.c](#).

### 5.5.3.2 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 122 of file [mpcotool.c](#).

### 5.5.3.3 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 110 of file [mpcotool.c](#).

## 5.6 mpcotool.c

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00030 #define _GNU_SOURCE
00031 #include "config.h"
00032 #include <stdio.h>
00033 #include <stdlib.h>
00034 #include <string.h>
00035 #include <math.h>
00036 #include <unistd.h>
00037 #include <locale.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>
```

```

00051 #elif (!__BSD_VISIBLE)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "mpcotool.h"
00059 #if HAVE_GTK
00060 #include <gio/gio.h>
00061 #include <gtk/gtk.h>
00062 #include "interface.h"
00063 #endif
00064
00065 #define DEBUG 0
00066
00067 #if HAVE_GTK
00077 #define ERROR_TYPE GTK_MESSAGE_ERROR
00078 #define INFO_TYPE GTK_MESSAGE_INFO
00079 #else
00080 #define ERROR_TYPE 0
00081 #define INFO_TYPE 0
00082 #endif
00083 #ifdef G_OS_WIN32
00084 #define INPUT_FILE "test-ga-win.xml"
00085 #define RM "del"
00086 #else
00087 #define INPUT_FILE "test-ga.xml"
00088 #define RM "rm"
00089 #endif
00090
00091 int ntasks;
00092 unsigned int nthreads;
00093 unsigned int nthreads_gradient;
00095 GMutex mutex[1];
00096 void (*calibrate_algorithm) ();
00098 double (*calibrate_estimate_gradient) (unsigned int variable,
00099                                       unsigned int estimate);
00101 Input input[1];
00103 Calibrate calibrate[1];
00104
00105 const xmlChar *result_name = (xmlChar *) "result";
00107 const xmlChar *variables_name = (xmlChar *) "variables";
00109
00110 const xmlChar *template[MAX_NINPUTS] = {
00111     XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
00112     XML_TEMPLATE4,
00113     XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
00114     XML_TEMPLATE8
00115 };
00116
00117 const char *format[NPRECISIONS] = {
00118     "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00119     "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00120 };
00121
00122 const double precision[NPRECISIONS] = {
00123     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,
00124     1e-13, 1e-14
00125 };
00126
00127 const char *logo[] = {
00128     "32 32 3 1",
00129     "      c None",
00130     ".      c #0000FF",
00131     "+      c #FF0000",
00132     "      ",
00133     "      ",
00134     "      ",
00135     "      .      .      .      .      ",
00136     "      .      .      .      .      ",
00137     "      .      .      .      .      ",
00138     "      .      .      .      .      ",
00139     "      .      .      + + +      .      ",
00140     "      .      .      + + + + +      .      ",
00141     "      .      .      + + + + +      .      ",
00142     "      .      .      + + + + +      .      ",
00143     "      + + +      .      + + +      + + +      ",
00144     "      + + + + +      .      .      + + + + +      ",
00145     "      + + + + +      .      .      + + + + +      ",
00146     "      + + + + +      .      .      + + + + +      ",
00147     "      + + +      .      .      + + +      ",
00148     "      .      .      .      .      ",
00149     "      .      + + +      .      .      ",
00150     "      .      + + + + +      .      .      ",
00151     "      .      + + + + +      .      .      ",

```

```
00152 " . . +++++ . . " ,
00153 " . . +++ . . " ,
00154 " . . . . . " ,
00155 " . . . . . " ,
00156 " . . . . . " ,
00157 " . . . . . " ,
00158 " . . . . . " ,
00159 " . . . . . " ,
00160 " . . . . . " ,
00161 " . . . . . " ,
00162 " . . . . . " ,
00163 " . . . . . "
00164 };
00165
00166 /*
00167 const char * logo[] = {
00168 "32 32 3 1",
00169 " c #FFFFFFFFFFFFFFF",
00170 ". c #00000000FFFF",
00171 "X c #FFF00000000",
00172 " ",
00173 " ",
00174 " ",
00175 " . . . . . " ,
00176 " . . . . . " ,
00177 " . . . . . " ,
00178 " . . . . . " ,
00179 " . . XXX . " ,
00180 " . . XXXXX . " ,
00181 " . . XXXXX . " ,
00182 " . . XXXXX . " ,
00183 " XXX . XXX XXX " ,
00184 " XXXXX . XXXXX " ,
00185 " XXXXX . XXXXX " ,
00186 " XXXXX . XXXXX " ,
00187 " XXX . XXX " ,
00188 " . . . . . " ,
00189 " . XXX . . " ,
00190 " . XXXXX . . " ,
00191 " . XXXXX . . " ,
00192 " . XXXXX . . " ,
00193 " . XXX . . " ,
00194 " . . . . . " ,
00195 " . . . . . " ,
00196 " . . . . . " ,
00197 " . . . . . " ,
00198 " . . . . . " ,
00199 " . . . . . " ,
00200 " . . . . . " ,
00201 " . . . . . " ,
00202 " . . . . . " ,
00203 " . . . . . " ;
00204 */
00205
00206 #if HAVE_GTK
00207 Options options[1];
00209 Running running[1];
00211 Window window[1];
00213 #endif
00214
00225 void
00226 show_message (char *title, char *msg, int type)
00227 {
00228 #if HAVE_GTK
00229 GtkWidgetDialog *dlg;
00230
00231 // Creating the dialog
00232 dlg = (GtkWidgetDialog *) gtk_message_dialog_new
00233 (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00234
00235 // Setting the dialog title
00236 gtk_window_set_title (GTK_WINDOW (dlg), title);
00237
00238 // Showing the dialog and waiting response
00239 gtk_dialog_run (GTK_DIALOG (dlg));
00240
00241 // Closing and freeing memory
00242 gtk_widget_destroy (GTK_WIDGET (dlg));
00243
00244 #else
00245 printf ("%s: %s\n", title, msg);
00246 #endif
00247 }
00248
00255 void
00256 show_error (char *msg)
00257 {
```

```

00258     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00259 }
00260
00273 int
00274 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00275 {
00276     int i = 0;
00277     xmlChar *buffer;
00278     buffer = xmlGetProp (node, prop);
00279     if (!buffer)
00280         *error_code = 1;
00281     else
00282     {
00283         if (sscanf ((char *) buffer, "%d", &i) != 1)
00284             *error_code = 2;
00285         else
00286             *error_code = 0;
00287         xmlFree (buffer);
00288     }
00289     return i;
00290 }
00291
00304 unsigned int
00305 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00306 {
00307     unsigned int i = 0;
00308     xmlChar *buffer;
00309     buffer = xmlGetProp (node, prop);
00310     if (!buffer)
00311         *error_code = 1;
00312     else
00313     {
00314         if (sscanf ((char *) buffer, "%u", &i) != 1)
00315             *error_code = 2;
00316         else
00317             *error_code = 0;
00318         xmlFree (buffer);
00319     }
00320     return i;
00321 }
00322
00335 double
00336 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00337 {
00338     double x = 0.;
00339     xmlChar *buffer;
00340     buffer = xmlGetProp (node, prop);
00341     if (!buffer)
00342         *error_code = 1;
00343     else
00344     {
00345         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346             *error_code = 2;
00347         else
00348             *error_code = 0;
00349         xmlFree (buffer);
00350     }
00351     return x;
00352 }
00353
00364 void
00365 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00366 {
00367     xmlChar buffer[64];
00368     snprintf ((char *) buffer, 64, "%d", value);
00369     xmlSetProp (node, prop, buffer);
00370 }
00371
00383 void
00384 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00385 {
00386     xmlChar buffer[64];
00387     snprintf ((char *) buffer, 64, "%u", value);
00388     xmlSetProp (node, prop, buffer);
00389 }
00390
00402 void
00403 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00404 {
00405     xmlChar buffer[64];
00406     snprintf ((char *) buffer, 64, "%.14lg", value);
00407     xmlSetProp (node, prop, buffer);
00408 }
00409
00414 void
00415 input_new ()
00416 {

```



```

00417     unsigned int i;
00418     #if DEBUG
00419     fprintf (stderr, "input_new: start\n");
00420     #endif
00421     input->nvariables = input->nexperiments = input->ninputs = input->
nsteps = 0;
00422     input->simulator = input->evaluator = input->directory = input->
name
00423     = input->result = input->variables = NULL;
00424     input->experiment = input->label = NULL;
00425     input->precision = input->nsweeps = input->nbits = NULL;
00426     input->rangemin = input->rangemax = input->rangeminabs = input->
rangemaxabs
00427     = input->weight = input->step = NULL;
00428     for (i = 0; i < MAX_NINPUTS; ++i)
00429         input->template[i] = NULL;
00430     #if DEBUG
00431     fprintf (stderr, "input_new: end\n");
00432     #endif
00433 }
00434
00439 void
00440 input_free ()
00441 {
00442     unsigned int i, j;
00443     #if DEBUG
00444     fprintf (stderr, "input_free: start\n");
00445     #endif
00446     g_free (input->name);
00447     g_free (input->directory);
00448     for (i = 0; i < input->nexperiments; ++i)
00449     {
00450         xmlFree (input->experiment[i]);
00451         for (j = 0; j < input->ninputs; ++j)
00452             xmlFree (input->template[j][i]);
00453         g_free (input->template[j]);
00454     }
00455     g_free (input->experiment);
00456     for (i = 0; i < input->ninputs; ++i)
00457         g_free (input->template[i]);
00458     for (i = 0; i < input->nvariables; ++i)
00459         xmlFree (input->label[i]);
00460     g_free (input->label);
00461     g_free (input->precision);
00462     g_free (input->rangemin);
00463     g_free (input->rangemax);
00464     g_free (input->rangeminabs);
00465     g_free (input->rangemaxabs);
00466     g_free (input->weight);
00467     g_free (input->step);
00468     g_free (input->nsweeps);
00469     g_free (input->nbits);
00470     xmlFree (input->evaluator);
00471     xmlFree (input->simulator);
00472     xmlFree (input->result);
00473     xmlFree (input->variables);
00474     input->nexperiments = input->ninputs = input->nvariables = input->
nsteps = 0;
00475     #if DEBUG
00476     fprintf (stderr, "input_free: end\n");
00477     #endif
00478 }
00479
00487 int
00488 input_open (char *filename)
00489 {
00490     char buffer2[64];
00491     char *buffert[MAX_NINPUTS] =
00492     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493     xmlDoc *doc;
00494     xmlNode *node, *child;
00495     xmlChar *buffer;
00496     char *msg;
00497     int error_code;
00498     unsigned int i;
00499
00500     #if DEBUG
00501     fprintf (stderr, "input_open: start\n");
00502     #endif
00503
00504     // Resetting input data
00505     buffer = NULL;
00506     input_new ();
00507
00508     // Parsing the input file
00509     #if DEBUG
00510     fprintf (stderr, "input_open: parsing the input file %s\n", filename);

```

```

00511 #endif
00512 doc = xmlParseFile (filename);
00513 if (!doc)
00514 {
00515     msg = gettext ("Unable to parse the input file");
00516     goto exit_on_error;
00517 }
00518
00519 // Getting the root node
00520 #if DEBUG
00521 fprintf (stderr, "input_open: getting the root node\n");
00522 #endif
00523 node = xmlDocGetRootElement (doc);
00524 if (xmlStrcmp (node->name, XML_CALIBRATE))
00525 {
00526     msg = gettext ("Bad root XML node");
00527     goto exit_on_error;
00528 }
00529
00530 // Getting results file names
00531 input->result = (char *) xmlGetProp (node, XML_RESULT);
00532 if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535 if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538 // Opening simulator program name
00539 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540 if (!input->simulator)
00541 {
00542     msg = gettext ("Bad simulator program");
00543     goto exit_on_error;
00544 }
00545
00546 // Opening evaluator program name
00547 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549 // Obtaining pseudo-random numbers generator seed
00550 if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552 else
00553 {
00554     input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555     if (error_code)
00556     {
00557         msg = gettext ("Bad pseudo-random numbers generator seed");
00558         goto exit_on_error;
00559     }
00560 }
00561
00562 // Opening algorithm
00563 buffer = xmlGetProp (node, XML_ALGORITHM);
00564 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565 {
00566     input->algorithm = ALGORITHM_MONTE_CARLO;
00567
00568     // Obtaining simulations number
00569     input->nsimulations
00570     = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571     if (error_code)
00572     {
00573         msg = gettext ("Bad simulations number");
00574         goto exit_on_error;
00575     }
00576 }
00577 else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     input->algorithm = ALGORITHM_SWEEP;
00579 else if (!xmlStrcmp (buffer, XML_GENETIC))
00580 {
00581     input->algorithm = ALGORITHM_GENETIC;
00582
00583     // Obtaining population
00584     if (xmlHasProp (node, XML_NPOPULATION))
00585     {
00586         input->nsimulations
00587         = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00588         if (error_code || input->nsimulations < 3)
00589         {
00590             msg = gettext ("Invalid population number");
00591             goto exit_on_error;
00592         }
00593     }
00594 else
00595 {
00596     msg = gettext ("No population number");
00597     goto exit_on_error;

```

```

00598     }
00599
00600     // Obtaining generations
00601     if (xmlHasProp (node, XML_NGENERATIONS))
00602     {
00603         input->niterations
00604         = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00605         if (error_code || !input->niterations)
00606         {
00607             msg = gettext ("Invalid generations number");
00608             goto exit_on_error;
00609         }
00610     }
00611     else
00612     {
00613         msg = gettext ("No generations number");
00614         goto exit_on_error;
00615     }
00616
00617     // Obtaining mutation probability
00618     if (xmlHasProp (node, XML_MUTATION))
00619     {
00620         input->mutation_ratio
00621         = xml_node_get_float (node, XML_MUTATION, &error_code);
00622         if (error_code || input->mutation_ratio < 0.
00623             || input->mutation_ratio >= 1.)
00624         {
00625             msg = gettext ("Invalid mutation probability");
00626             goto exit_on_error;
00627         }
00628     }
00629     else
00630     {
00631         msg = gettext ("No mutation probability");
00632         goto exit_on_error;
00633     }
00634
00635     // Obtaining reproduction probability
00636     if (xmlHasProp (node, XML_REPRODUCTION))
00637     {
00638         input->reproduction_ratio
00639         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00640         if (error_code || input->reproduction_ratio < 0.
00641             || input->reproduction_ratio >= 1.0)
00642         {
00643             msg = gettext ("Invalid reproduction probability");
00644             goto exit_on_error;
00645         }
00646     }
00647     else
00648     {
00649         msg = gettext ("No reproduction probability");
00650         goto exit_on_error;
00651     }
00652
00653     // Obtaining adaptation probability
00654     if (xmlHasProp (node, XML_ADAPTATION))
00655     {
00656         input->adaptation_ratio
00657         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00658         if (error_code || input->adaptation_ratio < 0.
00659             || input->adaptation_ratio >= 1.)
00660         {
00661             msg = gettext ("Invalid adaptation probability");
00662             goto exit_on_error;
00663         }
00664     }
00665     else
00666     {
00667         msg = gettext ("No adaptation probability");
00668         goto exit_on_error;
00669     }
00670
00671     // Checking survivals
00672     i = input->mutation_ratio * input->nsimulations;
00673     i += input->reproduction_ratio * input->nsimulations;
00674     i += input->adaptation_ratio * input->nsimulations;
00675     if (i > input->nsimulations - 2)
00676     {
00677         msg = gettext
00678             ("No enough survival entities to reproduce the population");
00679         goto exit_on_error;
00680     }
00681 }
00682 else
00683 {
00684     msg = gettext ("Unknown algorithm");

```

```

00685         goto exit_on_error;
00686     }
00687     xmlFree (buffer);
00688     buffer = NULL;
00689
00690     if (input->algorithm == ALGORITHM_MONTE_CARLO
00691         || input->algorithm == ALGORITHM_SWEEP)
00692     {
00693
00694         // Obtaining iterations number
00695         input->niterations
00696             = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00697         if (error_code == 1)
00698             input->niterations = 1;
00699         else if (error_code)
00700         {
00701             msg = gettext ("Bad iterations number");
00702             goto exit_on_error;
00703         }
00704
00705         // Obtaining best number
00706         if (xmlHasProp (node, XML_NBEST))
00707         {
00708             input->nbest = xml_node_get_uint (node,
XML_NBEST, &error_code);
00709             if (error_code || !input->nbest)
00710             {
00711                 msg = gettext ("Invalid best number");
00712                 goto exit_on_error;
00713             }
00714         }
00715         else
00716             input->nbest = 1;
00717
00718         // Obtaining tolerance
00719         if (xmlHasProp (node, XML_TOLERANCE))
00720         {
00721             input->tolerance
00722                 = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00723             if (error_code || input->tolerance < 0.)
00724             {
00725                 msg = gettext ("Invalid tolerance");
00726                 goto exit_on_error;
00727             }
00728         }
00729         else
00730             input->tolerance = 0.;
00731
00732         // Getting gradient method parameters
00733         if (xmlHasProp (node, XML_NSTEPS))
00734         {
00735             input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
00736             if (error_code || !input->nsteps)
00737             {
00738                 msg = gettext ("Invalid steps number");
00739                 goto exit_on_error;
00740             }
00741             buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00742             if (!xmlStrcmp (buffer, XML_COORDINATES))
00743                 input->gradient_method = GRADIENT_METHOD_COORDINATES;
00744             else if (!xmlStrcmp (buffer, XML_RANDOM))
00745             {
00746                 input->gradient_method = GRADIENT_METHOD_RANDOM;
00747                 input->nestimates
00748                     = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00749                 if (error_code || !input->nestimates)
00750                 {
00751                     msg = gettext ("Invalid estimates number");
00752                     goto exit_on_error;
00753                 }
00754             }
00755             else
00756             {
00757                 msg = gettext ("Unknown method to estimate the gradient");
00758                 goto exit_on_error;
00759             }
00760             xmlFree (buffer);
00761             buffer = NULL;
00762             if (xmlHasProp (node, XML_RELAXATION))
00763             {
00764                 input->relaxation
00765                     = xml_node_get_float (node, XML_RELAXATION, &error_code);
00766                 if (error_code || input->relaxation < 0.
00767                     || input->relaxation > 2.)
00768                 {
00769                     msg = gettext ("Invalid relaxation parameter");

```

```

00770             goto exit_on_error;
00771         }
00772     }
00773     else
00774         input->relaxation = DEFAULT_RELAXATION;
00775 }
00776 else
00777     input->nsteps = 0;
00778 }
00779
00780 // Reading the experimental data
00781 for (child = node->children; child; child = child->next)
00782 {
00783     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00784         break;
00785 #if DEBUG
00786     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00787 #endif
00788     if (xmlHasProp (child, XML_NAME))
00789         buffer = xmlGetProp (child, XML_NAME);
00790     else
00791     {
00792         snprintf (buffer2, 64, "%s %u: %s",
00793             gettext ("Experiment"),
00794             input->nexperiments + 1, gettext ("no data file name"));
00795         msg = buffer2;
00796         goto exit_on_error;
00797     }
00798 #if DEBUG
00799     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00800 #endif
00801     input->weight = g_realloc (input->weight,
00802         (1 + input->nexperiments) * sizeof (double));
00803     if (xmlHasProp (child, XML_WEIGHT))
00804     {
00805         input->weight[input->nexperiments]
00806             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00807         if (error_code)
00808         {
00809             snprintf (buffer2, 64, "%s %s: %s",
00810                 gettext ("Experiment"), buffer, gettext ("bad weight"));
00811             msg = buffer2;
00812             goto exit_on_error;
00813         }
00814     }
00815     else
00816         input->weight[input->nexperiments] = 1.;
00817 #if DEBUG
00818     fprintf (stderr, "input_open: weight=%lg\n",
00819         input->weight[input->nexperiments]);
00820 #endif
00821     if (!input->nexperiments)
00822         input->ninputs = 0;
00823 #if DEBUG
00824     fprintf (stderr, "input_open: template[0]\n");
00825 #endif
00826     if (xmlHasProp (child, XML_TEMPLATE1))
00827     {
00828         input->template[0]
00829             = (char **) g_realloc (input->template[0],
00830                 (1 + input->nexperiments) * sizeof (char *));
00831         buffert[0] = (char *) xmlGetProp (child, template[0]);
00832 #if DEBUG
00833         fprintf (stderr, "input_open: experiment=%u templatel=%s\n",
00834             input->nexperiments, buffert[0]);
00835 #endif
00836         if (!input->nexperiments)
00837             ++input->ninputs;
00838 #if DEBUG
00839         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00840 #endif
00841     }
00842     else
00843     {
00844         snprintf (buffer2, 64, "%s %s: %s",
00845             gettext ("Experiment"), buffer, gettext ("no template"));
00846         msg = buffer2;
00847         goto exit_on_error;
00848     }
00849     for (i = 1; i < MAX_NINPUTS; ++i)
00850     {
00851 #if DEBUG
00852         fprintf (stderr, "input_open: template%u\n", i + 1);
00853 #endif
00854         if (xmlHasProp (child, template[i]))
00855         {
00856             if (input->nexperiments && input->ninputs <= i)

```

```

00857         {
00858             snprintf (buffer2, 64, "%s %s: %s",
00859                     gettext ("Experiment"),
00860                     buffer, gettext ("bad templates number"));
00861             msg = buffer2;
00862             while (i-- > 0)
00863                 xmlFree (buffert[i]);
00864             goto exit_on_error;
00865         }
00866         input->template[i] = (char **)
00867             g_realloc (input->template[i],
00868                     (1 + input->nexperiments) * sizeof (char *));
00869         buffert[i] = (char *) xmlGetProp (child, template[i]);
00870 #if DEBUG
00871         fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00872                 input->nexperiments, i + 1,
00873                 input->template[i][input->nexperiments]);
00874 #endif
00875         if (!input->nexperiments)
00876             ++input->ninputs;
00877 #if DEBUG
00878         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00879 #endif
00880     }
00881     else if (input->nexperiments && input->ninputs >= i)
00882     {
00883         snprintf (buffer2, 64, "%s %s: %s",
00884                 gettext ("Experiment"),
00885                 buffer, gettext ("no template"), i + 1);
00886         msg = buffer2;
00887         while (i-- > 0)
00888             xmlFree (buffert[i]);
00889         goto exit_on_error;
00890     }
00891     else
00892         break;
00893 }
00894 input->experiment
00895     = g_realloc (input->experiment,
00896                 (1 + input->nexperiments) * sizeof (char *));
00897 input->experiment[input->nexperiments] = (char *) buffer;
00898 for (i = 0; i < input->ninputs; ++i)
00899     input->template[i][input->nexperiments] = buffert[i];
00900 ++input->nexperiments;
00901 #if DEBUG
00902     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00903 #endif
00904 }
00905 if (!input->nexperiments)
00906 {
00907     msg = gettext ("No calibration experiments");
00908     goto exit_on_error;
00909 }
00910 buffer = NULL;
00911
00912 // Reading the variables data
00913 for (; child; child = child->next)
00914 {
00915     if (xmlStrcmp (child->name, XML_VARIABLE))
00916     {
00917         snprintf (buffer2, 64, "%s %u: %s",
00918                 gettext ("Variable"),
00919                 input->nvariables + 1, gettext ("bad XML node"));
00920         msg = buffer2;
00921         goto exit_on_error;
00922     }
00923     if (xmlHasProp (child, XML_NAME))
00924         buffer = xmlGetProp (child, XML_NAME);
00925     else
00926     {
00927         snprintf (buffer2, 64, "%s %u: %s",
00928                 gettext ("Variable"),
00929                 input->nvariables + 1, gettext ("no name"));
00930         msg = buffer2;
00931         goto exit_on_error;
00932     }
00933     if (xmlHasProp (child, XML_MINIMUM))
00934     {
00935         input->rangemin = g_realloc
00936             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00937         input->rangeminabs = g_realloc
00938             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00939         input->rangemin[input->nvariables]
00940             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00941         if (error_code)
00942         {
00943             snprintf (buffer2, 64, "%s %s: %s",

```

```

00944         gettext ("Variable"), buffer, gettext ("bad minimum"));
00945     msg = buffer2;
00946     goto exit_on_error;
00947 }
00948 if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00949 {
00950     input->rangeminabs[input->nvariables]
00951     = xml_node_get_float (child,
XML_ABSOLUTE_MINIMUM, &error_code);
00952     if (error_code)
00953     {
00954         snprintf (buffer2, 64, "%s %s: %s",
00955             gettext ("Variable"),
00956             buffer, gettext ("bad absolute minimum"));
00957         msg = buffer2;
00958         goto exit_on_error;
00959     }
00960 }
00961 else
00962     input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00963 if (input->rangemin[input->nvariables]
00964     < input->rangeminabs[input->nvariables])
00965 {
00966     snprintf (buffer2, 64, "%s %s: %s",
00967         gettext ("Variable"),
00968         buffer, gettext ("minimum range not allowed"));
00969     msg = buffer2;
00970     goto exit_on_error;
00971 }
00972 }
00973 else
00974 {
00975     snprintf (buffer2, 64, "%s %s: %s",
00976         gettext ("Variable"), buffer, gettext ("no minimum range"));
00977     msg = buffer2;
00978     goto exit_on_error;
00979 }
00980 if (xmlHasProp (child, XML_MAXIMUM))
00981 {
00982     input->rangemax = g_realloc
00983     (input->rangemax, (1 + input->nvariables) * sizeof (double));
00984     input->rangemaxabs = g_realloc
00985     (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00986     input->rangemax[input->nvariables]
00987     = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00988     if (error_code)
00989     {
00990         snprintf (buffer2, 64, "%s %s: %s",
00991             gettext ("Variable"), buffer, gettext ("bad maximum"));
00992         msg = buffer2;
00993         goto exit_on_error;
00994     }
00995     if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00996     {
00997         input->rangemaxabs[input->nvariables]
00998         = xml_node_get_float (child,
XML_ABSOLUTE_MAXIMUM, &error_code);
00999         if (error_code)
01000         {
01001             snprintf (buffer2, 64, "%s %s: %s",
01002                 gettext ("Variable"),
01003                 buffer, gettext ("bad absolute maximum"));
01004             msg = buffer2;
01005             goto exit_on_error;
01006         }
01007     }
01008     else
01009         input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01010     if (input->rangemax[input->nvariables]
01011         > input->rangemaxabs[input->nvariables])
01012     {
01013         snprintf (buffer2, 64, "%s %s: %s",
01014             gettext ("Variable"),
01015             buffer, gettext ("maximum range not allowed"));
01016         msg = buffer2;
01017         goto exit_on_error;
01018     }
01019 }
01020 else
01021 {
01022     snprintf (buffer2, 64, "%s %s: %s",
01023         gettext ("Variable"), buffer, gettext ("no maximum range"));
01024     msg = buffer2;
01025     goto exit_on_error;
01026 }
01027 if (input->rangemax[input->nvariables]
01028     < input->rangemin[input->nvariables])

```

```

01029     {
01030         snprintf (buffer2, 64, "%s %s: %s",
01031                 gettext ("Variable"), buffer, gettext ("bad range"));
01032         msg = buffer2;
01033         goto exit_on_error;
01034     }
01035     input->precision = g_realloc
01036     (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01037     if (xmlHasProp (child, XML_PRECISION))
01038     {
01039         input->precision[input->nvariables]
01040         = xml_node_get_uint (child, XML_PRECISION, &error_code);
01041         if (error_code || input->precision[input->nvariables] >=
NPRECSIONS)
01042         {
01043             snprintf (buffer2, 64, "%s %s: %s",
01044                     gettext ("Variable"),
01045                     buffer, gettext ("bad precision"));
01046             msg = buffer2;
01047             goto exit_on_error;
01048         }
01049     }
01050     else
01051         input->precision[input->nvariables] =
DEFAULT_PRECISION;
01052     if (input->algorithm == ALGORITHM_SWEEP)
01053     {
01054         if (xmlHasProp (child, XML_NSWEEPS))
01055         {
01056             input->nsweeps = (unsigned int *)
01057             g_realloc (input->nsweeps,
01058                     (1 + input->nvariables) * sizeof (unsigned int));
01059             input->nsweeps[input->nvariables]
01060             = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01061             if (error_code || !input->nsweeps[input->nvariables])
01062             {
01063                 snprintf (buffer2, 64, "%s %s: %s",
01064                         gettext ("Variable"),
01065                         buffer, gettext ("bad sweeps"));
01066                 msg = buffer2;
01067                 goto exit_on_error;
01068             }
01069         }
01070         else
01071         {
01072             snprintf (buffer2, 64, "%s %s: %s",
01073                     gettext ("Variable"),
01074                     buffer, gettext ("no sweeps number"));
01075             msg = buffer2;
01076             goto exit_on_error;
01077         }
01078     }
01079     #if DEBUG
01080     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01081             input->nsweeps[input->nvariables], input->
nsimulations);
01082     #endif
01083     if (input->algorithm == ALGORITHM_GENETIC)
01084     {
01085         // Obtaining bits representing each variable
01086         if (xmlHasProp (child, XML_NBITS))
01087         {
01088             input->nbits = (unsigned int *)
01089             g_realloc (input->nbits,
01090                     (1 + input->nvariables) * sizeof (unsigned int));
01091             i = xml_node_get_uint (child, XML_NBITS, &error_code);
01092             if (error_code || !i)
01093             {
01094                 snprintf (buffer2, 64, "%s %s: %s",
01095                         gettext ("Variable"),
01096                         buffer, gettext ("invalid bits number"));
01097                 msg = buffer2;
01098                 goto exit_on_error;
01099             }
01100             input->nbits[input->nvariables] = i;
01101         }
01102         else
01103         {
01104             snprintf (buffer2, 64, "%s %s: %s",
01105                     gettext ("Variable"),
01106                     buffer, gettext ("no bits number"));
01107             msg = buffer2;
01108             goto exit_on_error;
01109         }
01110     }
01111     else if (input->nsteps)
01112     {

```



```

01113         input->step = (double *)
01114             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01115         input->step[input->nvariables]
01116             = xml_node_get_float (child, XML_STEP, &error_code);
01117         if (error_code || input->step[input->nvariables] < 0.)
01118             {
01119                 snprintf (buffer2, 64, "%s %s: %s",
01120                     gettext ("Variable"),
01121                     buffer, gettext ("bad step size"));
01122                 msg = buffer2;
01123                 goto exit_on_error;
01124             }
01125     }
01126     input->label = g_realloc
01127         (input->label, (1 + input->nvariables) * sizeof (char *));
01128     input->label[input->nvariables] = (char *) buffer;
01129     ++input->nvariables;
01130 }
01131 if (!input->nvariables)
01132 {
01133     msg = gettext ("No calibration variables");
01134     goto exit_on_error;
01135 }
01136 buffer = NULL;
01137
01138 // Getting the working directory
01139 input->directory = g_path_get_dirname (filename);
01140 input->name = g_path_get_basename (filename);
01141
01142 // Closing the XML document
01143 xmlFreeDoc (doc);
01144
01145 #if DEBUG
01146 fprintf (stderr, "input_open: end\n");
01147 #endif
01148 return 1;
01149
01150 exit_on_error:
01151 xmlFree (buffer);
01152 xmlFreeDoc (doc);
01153 show_error (msg);
01154 input_free ();
01155 #if DEBUG
01156 fprintf (stderr, "input_open: end\n");
01157 #endif
01158 return 0;
01159 }
01160
01172 void
01173 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01174 {
01175     unsigned int i;
01176     char buffer[32], value[32], *buffer2, *buffer3, *content;
01177     FILE *file;
01178     gsize length;
01179     GRegex *regex;
01180
01181 #if DEBUG
01182 fprintf (stderr, "calibrate_input: start\n");
01183 #endif
01184
01185 // Checking the file
01186 if (!template)
01187     goto calibrate_input_end;
01188
01189 // Opening template
01190 content = g_mapped_file_get_contents (template);
01191 length = g_mapped_file_get_length (template);
01192 #if DEBUG
01193 fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01194     content);
01195 #endif
01196 file = g_fopen (input, "w");
01197
01198 // Parsing template
01199 for (i = 0; i < calibrate->nvariables; ++i)
01200 {
01201 #if DEBUG
01202 fprintf (stderr, "calibrate_input: variable=%u\n", i);
01203 #endif
01204     snprintf (buffer, 32, "@variable%u@", i + 1);
01205     regex = g_regex_new (buffer, 0, 0, NULL);
01206     if (i == 0)
01207     {
01208         buffer2 = g_regex_replace_literal (regex, content, length, 0,
01209             calibrate->label[i], 0, NULL);
01210 #if DEBUG

```

```

01211         fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01212 #endif
01213     }
01214     else
01215     {
01216         length = strlen (buffer3);
01217         buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01218                                           calibrate->label[i], 0, NULL);
01219         g_free (buffer3);
01220     }
01221     g_regex_unref (regex);
01222     length = strlen (buffer2);
01223     snprintf (buffer, 32, "@value%u@", i + 1);
01224     regex = g_regex_new (buffer, 0, 0, NULL);
01225     snprintf (value, 32, format[calibrate->precision[i]],
01226             calibrate->value[simulation * calibrate->nvariables + i]);
01227
01228 #if DEBUG
01229     fprintf (stderr, "calibrate_input: value=%s\n", value);
01230 #endif
01231     buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01232                                     0, NULL);
01233     g_free (buffer2);
01234     g_regex_unref (regex);
01235 }
01236
01237 // Saving input file
01238 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01239 g_free (buffer3);
01240 fclose (file);
01241
01242 calibrate_input_end:
01243 #if DEBUG
01244     fprintf (stderr, "calibrate_input: end\n");
01245 #endif
01246     return;
01247 }
01248
01249 double
01250 calibrate_parse (unsigned int simulation, unsigned int experiment)
01251 {
01252     unsigned int i;
01253     double e;
01254     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01255         *buffer3, *buffer4;
01256     FILE *file_result;
01257
01258 #if DEBUG
01259     fprintf (stderr, "calibrate_parse: start\n");
01260     fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01261             experiment);
01262 #endif
01263
01264     // Opening input files
01265     for (i = 0; i < calibrate->ninputs; ++i)
01266     {
01267         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01268 #if DEBUG
01269         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01270 #endif
01271 #endif
01272         calibrate_input (simulation, &input[i][0],
01273                         calibrate->file[i][experiment]);
01274     }
01275     for (; i < MAX_NINPUTS; ++i)
01276         strcpy (&input[i][0], "");
01277 #if DEBUG
01278     fprintf (stderr, "calibrate_parse: parsing end\n");
01279 #endif
01280
01281     // Performing the simulation
01282     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01283     buffer2 = g_path_get_dirname (calibrate->simulator);
01284     buffer3 = g_path_get_basename (calibrate->simulator);
01285     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01286     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01287             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01288             input[6], input[7], output);
01289     g_free (buffer4);
01290     g_free (buffer3);
01291     g_free (buffer2);
01292 #if DEBUG
01293     fprintf (stderr, "calibrate_parse: %s\n", buffer);
01294 #endif
01295     system (buffer);
01296
01297     // Checking the objective value function
01298     if (calibrate->evaluator)

```

```

01308     {
01309         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01310         buffer2 = g_path_get_dirname (calibrate->evaluator);
01311         buffer3 = g_path_get_basename (calibrate->evaluator);
01312         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01313         snprintf (buffer, 512, "%s\\\" %s %s %s",
01314                 buffer4, output, calibrate->experiment[experiment], result);
01315         g_free (buffer4);
01316         g_free (buffer3);
01317         g_free (buffer2);
01318         #if DEBUG
01319             fprintf (stderr, "calibrate_parse: %s\n", buffer);
01320         #endif
01321         system (buffer);
01322         file_result = g_fopen (result, "r");
01323         e = atof (fgets (buffer, 512, file_result));
01324         fclose (file_result);
01325     }
01326     else
01327     {
01328         strcpy (result, "");
01329         file_result = g_fopen (output, "r");
01330         e = atof (fgets (buffer, 512, file_result));
01331         fclose (file_result);
01332     }
01333
01334     // Removing files
01335     #if !DEBUG
01336         for (i = 0; i < calibrate->ninputs; ++i)
01337         {
01338             if (calibrate->file[i][0])
01339             {
01340                 snprintf (buffer, 512, RM " %s", &input[i][0]);
01341                 system (buffer);
01342             }
01343         }
01344         snprintf (buffer, 512, RM " %s %s", output, result);
01345         system (buffer);
01346     #endif
01347
01348     #if DEBUG
01349         fprintf (stderr, "calibrate_parse: end\n");
01350     #endif
01351
01352     // Returning the objective function
01353     return e * calibrate->weight[experiment];
01354 }
01355
01356 void
01357 calibrate_print ()
01358 {
01359     unsigned int i;
01360     char buffer[512];
01361     #if HAVE_MPI
01362         if (calibrate->mpi_rank)
01363             return;
01364     #endif
01365     printf ("%s\n", gettext ("Best result"));
01366     fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01367     printf ("error = %.15le\n", calibrate->error_old[0]);
01368     fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
01369             error_old[0]);
01370     for (i = 0; i < calibrate->nvariables; ++i)
01371     {
01372         snprintf (buffer, 512, "%s = %s\n",
01373                 calibrate->label[i], format[calibrate->precision[i]]);
01374         printf (buffer, calibrate->value_old[i]);
01375         fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01376     }
01377     fflush (calibrate->file_result);
01378 }
01379
01380 void
01381 calibrate_save_variables (unsigned int simulation, double error)
01382 {
01383     unsigned int i;
01384     char buffer[64];
01385     #if DEBUG
01386         fprintf (stderr, "calibrate_save_variables: start\n");
01387     #endif
01388     for (i = 0; i < calibrate->nvariables; ++i)
01389     {
01390         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01391         fprintf (calibrate->file_variables, buffer,
01392                 calibrate->value[simulation * calibrate->nvariables + i]);
01393     }
01394     fprintf (calibrate->file_variables, "%.14le\n", error);

```

```

01406 #if DEBUG
01407     fprintf (stderr, "calibrate_save_variables: end\n");
01408 #endif
01409 }
01410
01419 void
01420 calibrate_best (unsigned int simulation, double value)
01421 {
01422     unsigned int i, j;
01423     double e;
01424     #if DEBUG
01425         fprintf (stderr, "calibrate_best: start\n");
01426         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01427             calibrate->nsaveds, calibrate->nbest);
01428     #endif
01429     if (calibrate->nsaveds < calibrate->nbest
01430         || value < calibrate->error_best[calibrate->nsaveds - 1])
01431     {
01432         if (calibrate->nsaveds < calibrate->nbest)
01433             ++calibrate->nsaveds;
01434         calibrate->error_best[calibrate->nsaveds - 1] = value;
01435         calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01436         for (i = calibrate->nsaveds; --i;)
01437         {
01438             if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01439             {
01440                 j = calibrate->simulation_best[i];
01441                 e = calibrate->error_best[i];
01442                 calibrate->simulation_best[i] = calibrate->
simulation_best[i - 1];
01443                 calibrate->error_best[i] = calibrate->error_best[i - 1];
01444                 calibrate->simulation_best[i - 1] = j;
01445                 calibrate->error_best[i - 1] = e;
01446             }
01447             else
01448                 break;
01449         }
01450     }
01451     #if DEBUG
01452         fprintf (stderr, "calibrate_best: end\n");
01453     #endif
01454 }
01455
01460 void
01461 calibrate_sequential ()
01462 {
01463     unsigned int i, j;
01464     double e;
01465     #if DEBUG
01466         fprintf (stderr, "calibrate_sequential: start\n");
01467         fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01468             calibrate->nstart, calibrate->nend);
01469     #endif
01470     for (i = calibrate->nstart; i < calibrate->nend; ++i)
01471     {
01472         e = 0.;
01473         for (j = 0; j < calibrate->nexperiments; ++j)
01474             e += calibrate_parse (i, j);
01475         calibrate_best (i, e);
01476         calibrate_save_variables (i, e);
01477     #if DEBUG
01478         fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01479     #endif
01480     }
01481     #if DEBUG
01482         fprintf (stderr, "calibrate_sequential: end\n");
01483     #endif
01484 }
01485
01493 void *
01494 calibrate_thread (ParallelData * data)
01495 {
01496     unsigned int i, j, thread;
01497     double e;
01498     #if DEBUG
01499         fprintf (stderr, "calibrate_thread: start\n");
01500     #endif
01501     thread = data->thread;
01502     #if DEBUG
01503         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01504             calibrate->thread[thread], calibrate->thread[thread + 1]);
01505     #endif
01506     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01507     {
01508         e = 0.;
01509         for (j = 0; j < calibrate->nexperiments; ++j)
01510             e += calibrate_parse (i, j);

```

```

01511     g_mutex_lock (mutex);
01512     calibrate_best (i, e);
01513     calibrate_save_variables (i, e);
01514     g_mutex_unlock (mutex);
01515     #if DEBUG
01516     fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01517     #endif
01518     }
01519     #if DEBUG
01520     fprintf (stderr, "calibrate_thread: end\n");
01521     #endif
01522     g_thread_exit (NULL);
01523     return NULL;
01524 }
01525
01537 void
01538 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01539                 double *error_best)
01540 {
01541     unsigned int i, j, k, s[calibrate->nbest];
01542     double e[calibrate->nbest];
01543     #if DEBUG
01544     fprintf (stderr, "calibrate_merge: start\n");
01545     #endif
01546     i = j = k = 0;
01547     do
01548     {
01549         if (i == calibrate->nsaveds)
01550         {
01551             s[k] = simulation_best[j];
01552             e[k] = error_best[j];
01553             ++j;
01554             ++k;
01555             if (j == nsaveds)
01556                 break;
01557         }
01558         else if (j == nsaveds)
01559         {
01560             s[k] = calibrate->simulation_best[i];
01561             e[k] = calibrate->error_best[i];
01562             ++i;
01563             ++k;
01564             if (i == calibrate->nsaveds)
01565                 break;
01566         }
01567         else if (calibrate->error_best[i] > error_best[j])
01568         {
01569             s[k] = simulation_best[j];
01570             e[k] = error_best[j];
01571             ++j;
01572             ++k;
01573         }
01574         else
01575         {
01576             s[k] = calibrate->simulation_best[i];
01577             e[k] = calibrate->error_best[i];
01578             ++i;
01579             ++k;
01580         }
01581     }
01582     while (k < calibrate->nbest);
01583     calibrate->nsaveds = k;
01584     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01585     memcpy (calibrate->error_best, e, k * sizeof (double));
01586     #if DEBUG
01587     fprintf (stderr, "calibrate_merge: end\n");
01588     #endif
01589 }
01590
01595 #if HAVE_MPI
01596 void
01597 calibrate_synchronise ()
01598 {
01599     unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01600     double error_best[calibrate->nbest];
01601     MPI_Status mpi_stat;
01602     #if DEBUG
01603     fprintf (stderr, "calibrate_synchronise: start\n");
01604     #endif
01605     if (calibrate->mpi_rank == 0)
01606     {
01607         for (i = 1; i < ntasks; ++i)
01608         {
01609             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01610             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01611                      MPI_COMM_WORLD, &mpi_stat);
01612             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,

```

```

01613             MPI_COMM_WORLD, &mpi_stat);
01614         calibrate_merge (nsaveds, simulation_best, error_best);
01615     }
01616 }
01617 else
01618 {
01619     MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01620     MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01621             MPI_COMM_WORLD);
01622     MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01623             MPI_COMM_WORLD);
01624 }
01625 #if DEBUG
01626 fprintf (stderr, "calibrate_synchronise: end\n");
01627 #endif
01628 }
01629 #endif
01630
01631 void
01632 calibrate_sweep ()
01633 {
01634     unsigned int i, j, k, l;
01635     double e;
01636     GThread *thread[nthreads];
01637     ParallelData data[nthreads];
01638     #if DEBUG
01639     fprintf (stderr, "calibrate_sweep: start\n");
01640     #endif
01641     for (i = 0; i < calibrate->nsimulations; ++i)
01642     {
01643         k = i;
01644         for (j = 0; j < calibrate->nvariables; ++j)
01645         {
01646             l = k % calibrate->nsweeps[j];
01647             k /= calibrate->nsweeps[j];
01648             e = calibrate->rangemin[j];
01649             if (calibrate->nsweeps[j] > 1)
01650                 e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01651                     / (calibrate->nsweeps[j] - 1);
01652             calibrate->value[i * calibrate->nvariables + j] = e;
01653         }
01654     }
01655     calibrate->nsaveds = 0;
01656     if (nthreads <= 1)
01657         calibrate_sequential ();
01658     else
01659     {
01660         for (i = 0; i < nthreads; ++i)
01661         {
01662             data[i].thread = i;
01663             thread[i]
01664                 = g_thread_new (NULL, (void (*) ) calibrate_thread, &data[i]);
01665         }
01666         for (i = 0; i < nthreads; ++i)
01667             g_thread_join (thread[i]);
01668     }
01669     #if HAVE_MPI
01670     // Communicating tasks results
01671     calibrate_synchronise ();
01672     #endif
01673     #if DEBUG
01674     fprintf (stderr, "calibrate_sweep: end\n");
01675     #endif
01676 }
01677
01678 void
01679 calibrate_MonteCarlo ()
01680 {
01681     unsigned int i, j;
01682     GThread *thread[nthreads];
01683     ParallelData data[nthreads];
01684     #if DEBUG
01685     fprintf (stderr, "calibrate_MonteCarlo: start\n");
01686     #endif
01687     for (i = 0; i < calibrate->nsimulations; ++i)
01688     {
01689         for (j = 0; j < calibrate->nvariables; ++j)
01690             calibrate->value[i * calibrate->nvariables + j]
01691                 = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01692                     * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01693         calibrate->nsaveds = 0;
01694         if (nthreads <= 1)
01695             calibrate_sequential ();
01696         else
01697         {
01698             for (i = 0; i < nthreads; ++i)
01699             {
01700                 data[i].thread = i;

```

```

01708         thread[i]
01709         = g_thread_new (NULL, (void (*)(void)) calibrate_thread, &data[i]);
01710     }
01711     for (i = 0; i < nthreads; ++i)
01712         g_thread_join (thread[i]);
01713 }
01714 #if HAVE_MPI
01715 // Communicating tasks results
01716 calibrate_synchronise ();
01717 #endif
01718 #if DEBUG
01719 fprintf (stderr, "calibrate_MonteCarlo: end\n");
01720 #endif
01721 }
01722
01723 void
01724 calibrate_best_gradient (unsigned int simulation, double value)
01725 {
01726     #if DEBUG
01727     fprintf (stderr, "calibrate_best_gradient: start\n");
01728     fprintf (stderr,
01729             "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01730             simulation, value, calibrate->error_best[0]);
01731     #endif
01732     if (value < calibrate->error_best[0])
01733     {
01734         calibrate->error_best[0] = value;
01735         calibrate->simulation_best[0] = simulation;
01736     }
01737     #if DEBUG
01738     fprintf (stderr,
01739             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01740             simulation, value);
01741     #endif
01742 }
01743 #if DEBUG
01744 fprintf (stderr, "calibrate_best_gradient: end\n");
01745 #endif
01746
01747 void
01748 calibrate_gradient_sequential (unsigned int simulation)
01749 {
01750     unsigned int i, j, k;
01751     double e;
01752     #if DEBUG
01753     fprintf (stderr, "calibrate_gradient_sequential: start\n");
01754     fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01755             "nend_gradient=%u\n",
01756             calibrate->nstart_gradient, calibrate->nend_gradient);
01757     #endif
01758     for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01759     {
01760         k = simulation + i;
01761         e = 0.;
01762         for (j = 0; j < calibrate->nexperiments; ++j)
01763             e += calibrate_parse (k, j);
01764         calibrate_best_gradient (k, e);
01765         calibrate_save_variables (k, e);
01766     }
01767     #if DEBUG
01768     fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01769     #endif
01770 }
01771 #if DEBUG
01772 fprintf (stderr, "calibrate_gradient_sequential: end\n");
01773 #endif
01774
01775 void *
01776 calibrate_gradient_thread (ParallelData * data)
01777 {
01778     unsigned int i, j, thread;
01779     double e;
01780     #if DEBUG
01781     fprintf (stderr, "calibrate_gradient_thread: start\n");
01782     #endif
01783     thread = data->thread;
01784     #if DEBUG
01785     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01786             thread,
01787             calibrate->thread_gradient[thread],
01788             calibrate->thread_gradient[thread + 1]);
01789     #endif
01790     for (i = calibrate->thread_gradient[thread];
01791          i < calibrate->thread_gradient[thread + 1]; ++i)
01792     {
01793         e = 0.;
01794         for (j = 0; j < calibrate->nexperiments; ++j)

```

```

01817         e += calibrate_parse (i, j);
01818         g_mutex_lock (mutex);
01819         calibrate_best_gradient (i, e);
01820         calibrate_save_variables (i, e);
01821         g_mutex_unlock (mutex);
01822 #if DEBUG
01823         fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01824 #endif
01825     }
01826 #if DEBUG
01827     fprintf (stderr, "calibrate_gradient_thread: end\n");
01828 #endif
01829     g_thread_exit (NULL);
01830     return NULL;
01831 }
01832
01833 double
01834 calibrate_estimate_gradient_random (unsigned int variable,
01835                                     unsigned int estimate)
01836 {
01837     double x;
01838 #if DEBUG
01839     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01840 #endif
01841     x = calibrate->gradient [variable]
01842         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step [variable];
01843 #if DEBUG
01844     fprintf (stderr, "calibrate_estimate_gradient_random: gradient=%lg\n",
01845             variable, x);
01846     fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01847 #endif
01848     return x;
01849 }
01850
01851 double
01852 calibrate_estimate_gradient_coordinates (unsigned int variable,
01853                                         unsigned int estimate)
01854 {
01855     double x;
01856 #if DEBUG
01857     fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01858 #endif
01859     x = calibrate->gradient [variable];
01860     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01861     {
01862         if (estimate & 1)
01863             x += calibrate->step [variable];
01864         else
01865             x -= calibrate->step [variable];
01866     }
01867 #if DEBUG
01868     fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient=%lg\n",
01869             variable, x);
01870     fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01871 #endif
01872     return x;
01873 }
01874
01875 void
01876 calibrate_step_gradient (unsigned int simulation)
01877 {
01878     GThread *thread[nthreads_gradient];
01879     ParallelData data[nthreads_gradient];
01880     unsigned int i, j, k, b;
01881 #if DEBUG
01882     fprintf (stderr, "calibrate_step_gradient: start\n");
01883 #endif
01884     for (i = 0; i < calibrate->nestimates; ++i)
01885     {
01886         k = (simulation + i) * calibrate->nvariables;
01887         b = calibrate->simulation_best[0] * calibrate->nvariables;
01888 #if DEBUG
01889         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01890                 simulation + i, calibrate->simulation_best[0]);
01891 #endif
01892         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01893         {
01894 #if DEBUG
01895             fprintf (stderr,
01896                     "calibrate_step_gradient: estimate=%u best=%u%.14le\n",
01897                     i, j, calibrate->value[b]);
01898 #endif
01899             calibrate->value[k]
01900                 = calibrate->value[b] + calibrate_estimate_gradient (j, i);
01901             calibrate->value[k] = fmin (fmax (calibrate->value[k],
01902                                             calibrate->rangeminabs[j]),
01903                                         calibrate->rangemaxabs[j]);
01904         }
01905     }
01906 }

```



```

01928 #if DEBUG
01929     fprintf (stderr,
01930             "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01931             i, j, calibrate->value[k]);
01932 #endif
01933     }
01934 }
01935 if (nthreads_gradient == 1)
01936     calibrate_gradient_sequential (simulation);
01937 else
01938     {
01939         for (i = 0; i <= nthreads_gradient; ++i)
01940         {
01941             calibrate->thread_gradient[i]
01942             = simulation + calibrate->nstart_gradient
01943             + i * (calibrate->nend_gradient - calibrate->
01944                 nstart_gradient)
01945             / nthreads_gradient;
01946 #if DEBUG
01947             fprintf (stderr,
01948                     "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01949                     i, calibrate->thread_gradient[i]);
01950 #endif
01951             for (i = 0; i < nthreads_gradient; ++i)
01952             {
01953                 data[i].thread = i;
01954                 thread[i] = g_thread_new
01955                     (NULL, (void (*) ) calibrate_gradient_thread, &data[i]);
01956             }
01957             for (i = 0; i < nthreads_gradient; ++i)
01958                 g_thread_join (thread[i]);
01959         }
01960 #if DEBUG
01961         fprintf (stderr, "calibrate_step_gradient: end\n");
01962 #endif
01963 }
01964
01965 void
01970 calibrate_gradient ()
01971 {
01972     unsigned int i, j, k, b, s, adjust;
01973 #if DEBUG
01974     fprintf (stderr, "calibrate_gradient: start\n");
01975 #endif
01976     for (i = 0; i < calibrate->nvariables; ++i)
01977         calibrate->gradient[i] = 0.;
01978     b = calibrate->simulation_best[0] * calibrate->nvariables;
01979     s = calibrate->nsimulations;
01980     adjust = 1;
01981     for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
01982     {
01983 #if DEBUG
01984         fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
01985                 i, calibrate->simulation_best[0]);
01986 #endif
01987         calibrate_step_gradient (s);
01988         k = calibrate->simulation_best[0] * calibrate->nvariables;
01989 #if DEBUG
01990         fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
01991                 i, calibrate->simulation_best[0]);
01992 #endif
01993         if (k == b)
01994         {
01995             if (adjust)
01996                 for (j = 0; j < calibrate->nvariables; ++j)
01997                     calibrate->step[j] *= 0.5;
01998             for (j = 0; j < calibrate->nvariables; ++j)
01999                 calibrate->gradient[j] = 0.;
02000             adjust = 1;
02001         }
02002         else
02003         {
02004             for (j = 0; j < calibrate->nvariables; ++j)
02005             {
02006 #if DEBUG
02007                 fprintf (stderr,
02008                         "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02009                         j, calibrate->value[k + j], j, calibrate->value[b + j]);
02010 #endif
02011                 calibrate->gradient[j]
02012                 = (1. - calibrate->relaxation) * calibrate->gradient[j]
02013                 + calibrate->relaxation
02014                 * (calibrate->value[k + j] - calibrate->value[b + j]);
02015 #if DEBUG
02016                 fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02017                         j, calibrate->gradient[j]);

```

```

02018 #endif
02019     }
02020     adjust = 0;
02021 }
02022 }
02023 #if DEBUG
02024 fprintf (stderr, "calibrate_gradient: end\n");
02025 #endif
02026 }
02027
02028 double
02029 calibrate_genetic_objective (Entity * entity)
02030 {
02031     unsigned int j;
02032     double objective;
02033     char buffer[64];
02034 #if DEBUG
02035     fprintf (stderr, "calibrate_genetic_objective: start\n");
02036 #endif
02037     for (j = 0; j < calibrate->nvariables; ++j)
02038     {
02039         calibrate->value[entity->id * calibrate->nvariables + j]
02040             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02041     }
02042     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02043         objective += calibrate_parse (entity->id, j);
02044     g_mutex_lock (mutex);
02045     for (j = 0; j < calibrate->nvariables; ++j)
02046     {
02047         snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02048         fprintf (calibrate->file_variables, buffer,
02049                 genetic_get_variable (entity, calibrate->genetic_variable + j));
02050     }
02051     fprintf (calibrate->file_variables, "%.14le\n", objective);
02052     g_mutex_unlock (mutex);
02053 #if DEBUG
02054     fprintf (stderr, "calibrate_genetic_objective: end\n");
02055 #endif
02056     return objective;
02057 }
02058
02059 void
02060 calibrate_genetic ()
02061 {
02062     char *best_genome;
02063     double best_objective, *best_variable;
02064 #if DEBUG
02065     fprintf (stderr, "calibrate_genetic: start\n");
02066     fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02067             nthreads);
02068     fprintf (stderr,
02069             "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02070             calibrate->nvariables, calibrate->nsimulations,
02071             calibrate->niterations);
02072     fprintf (stderr,
02073             "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02074             calibrate->mutation_ratio, calibrate->
02075             reproduction_ratio,
02076             calibrate->adaptation_ratio);
02077 #endif
02078     genetic_algorithm_default (calibrate->nvariables,
02079                               calibrate->genetic_variable,
02080                               calibrate->nsimulations,
02081                               calibrate->niterations,
02082                               calibrate->mutation_ratio,
02083                               calibrate->reproduction_ratio,
02084                               calibrate->adaptation_ratio,
02085                               &calibrate_genetic_objective,
02086                               &best_genome, &best_variable, &best_objective);
02087 #if DEBUG
02088     fprintf (stderr, "calibrate_genetic: the best\n");
02089 #endif
02090     calibrate->error_old = (double *) g_malloc (sizeof (double));
02091     calibrate->value_old
02092         = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02093     calibrate->error_old[0] = best_objective;
02094     memcpy (calibrate->value_old, best_variable,
02095            calibrate->nvariables * sizeof (double));
02096     g_free (best_genome);
02097     g_free (best_variable);
02098     calibrate_print ();
02099 #if DEBUG
02100     fprintf (stderr, "calibrate_genetic: end\n");
02101 #endif
02102 }
02103
02104 void

```

```

02119 calibrate_save_old ()
02120 {
02121     unsigned int i, j;
02122     #if DEBUG
02123         fprintf (stderr, "calibrate_save_old: start\n");
02124         fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02125     #endif
02126     memcpy (calibrate->error_old, calibrate->error_best,
02127             calibrate->nbest * sizeof (double));
02128     for (i = 0; i < calibrate->nbest; ++i)
02129     {
02130         j = calibrate->simulation_best[i];
02131         #if DEBUG
02132             fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02133         #endif
02134         memcpy (calibrate->value_old + i * calibrate->nvariables,
02135                 calibrate->value + j * calibrate->nvariables,
02136                 calibrate->nvariables * sizeof (double));
02137     }
02138     #if DEBUG
02139         for (i = 0; i < calibrate->nvariables; ++i)
02140             fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02141                     i, calibrate->value_old[i]);
02142         fprintf (stderr, "calibrate_save_old: end\n");
02143     #endif
02144 }
02145
02151 void
02152 calibrate_merge_old ()
02153 {
02154     unsigned int i, j, k;
02155     double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
02156             nbest],
02157            *enew, *eold;
02158     #if DEBUG
02159         fprintf (stderr, "calibrate_merge_old: start\n");
02160     #endif
02161     enew = calibrate->error_best;
02162     eold = calibrate->error_old;
02163     i = j = k = 0;
02164     do
02165     {
02166         if (*enew < *eold)
02167         {
02168             memcpy (v + k * calibrate->nvariables,
02169                     calibrate->value
02170                     + calibrate->simulation_best[i] * calibrate->
02171                     nvariables,
02172                     calibrate->nvariables * sizeof (double));
02173             e[k] = *enew;
02174             ++k;
02175             ++enew;
02176             ++i;
02177         }
02178         else
02179         {
02180             memcpy (v + k * calibrate->nvariables,
02181                     calibrate->value_old + j * calibrate->nvariables,
02182                     calibrate->nvariables * sizeof (double));
02183             e[k] = *eold;
02184             ++k;
02185             ++eold;
02186             ++j;
02187         }
02188     } while (k < calibrate->nbest);
02189     memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02190     memcpy (calibrate->error_old, e, k * sizeof (double));
02191     #if DEBUG
02192         fprintf (stderr, "calibrate_merge_old: end\n");
02193     #endif
02194 }
02195
02200 void
02201 calibrate_refine ()
02202 {
02203     unsigned int i, j;
02204     double d;
02205     #if HAVE_MPI
02206         MPI_Status mpi_stat;
02207     #endif
02208     #if DEBUG
02209         fprintf (stderr, "calibrate_refine: start\n");
02210     #endif
02211     #if HAVE_MPI
02212         if (!calibrate->mpi_rank)
02213         {

```

```

02214 #endif
02215     for (j = 0; j < calibrate->nvariables; ++j)
02216     {
02217         calibrate->rangemin[j] = calibrate->rangemax[j]
02218         = calibrate->value_old[j];
02219     }
02220     for (i = 0; ++i < calibrate->nbest;)
02221     {
02222         for (j = 0; j < calibrate->nvariables; ++j)
02223         {
02224             calibrate->rangemin[j]
02225             = fmin (calibrate->rangemin[j],
02226                     calibrate->value_old[i * calibrate->nvariables + j]);
02227             calibrate->rangemax[j]
02228             = fmax (calibrate->rangemax[j],
02229                     calibrate->value_old[i * calibrate->nvariables + j]);
02230         }
02231     }
02232     for (j = 0; j < calibrate->nvariables; ++j)
02233     {
02234         d = calibrate->tolerance
02235         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02236         switch (calibrate->algorithm)
02237         {
02238             case ALGORITHM_MONTE_CARLO:
02239                 d *= 0.5;
02240                 break;
02241             default:
02242                 if (calibrate->nsweeps[j] > 1)
02243                     d /= calibrate->nsweeps[j] - 1;
02244                 else
02245                     d = 0.;
02246         }
02247         calibrate->rangemin[j] -= d;
02248         calibrate->rangemin[j]
02249         = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02250         calibrate->rangemax[j] += d;
02251         calibrate->rangemax[j]
02252         = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02253         printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02254                 calibrate->rangemin[j], calibrate->rangemax[j]);
02255         fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02256                 calibrate->label[j], calibrate->rangemin[j],
02257                 calibrate->rangemax[j]);
02258     }
02259 #if HAVE_MPI
02260     for (i = 1; i < ntasks; ++i)
02261     {
02262         MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02263                  1, MPI_COMM_WORLD);
02264         MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02265                  1, MPI_COMM_WORLD);
02266     }
02267 }
02268 else
02269 {
02270     MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02271              MPI_COMM_WORLD, &mpi_stat);
02272     MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02273              MPI_COMM_WORLD, &mpi_stat);
02274 }
02275 #endif
02276 #if DEBUG
02277     fprintf (stderr, "calibrate_refine: end\n");
02278 #endif
02279 }
02280
02285 void
02286 calibrate_step ()
02287 {
02288     #if DEBUG
02289         fprintf (stderr, "calibrate_step: start\n");
02290     #endif
02291     calibrate_algorithm ();
02292     if (calibrate->nsteps)
02293         calibrate_gradient ();
02294     #if DEBUG
02295         fprintf (stderr, "calibrate_step: end\n");
02296     #endif
02297 }
02298
02303 void
02304 calibrate_iterate ()
02305 {
02306     unsigned int i;
02307     #if DEBUG
02308         fprintf (stderr, "calibrate_iterate: start\n");

```

```

02309 #endif
02310     calibrate->error_old
02311     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02312     calibrate->value_old = (double *)
02313     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02314     calibrate_step ();
02315     calibrate_save_old ();
02316     calibrate_refine ();
02317     calibrate_print ();
02318     for (i = 1; i < calibrate->niterations; ++i)
02319     {
02320         calibrate_step ();
02321         calibrate_merge_old ();
02322         calibrate_refine ();
02323         calibrate_print ();
02324     }
02325     #if DEBUG
02326     fprintf (stderr, "calibrate_iterate: end\n");
02327 #endif
02328 }
02329
02334 void
02335 calibrate_free ()
02336 {
02337     unsigned int i, j;
02338     #if DEBUG
02339     fprintf (stderr, "calibrate_free: start\n");
02340 #endif
02341     for (j = 0; j < calibrate->ninputs; ++j)
02342     {
02343         for (i = 0; i < calibrate->nexperiments; ++i)
02344             g_mapped_file_unref (calibrate->file[j][i]);
02345         g_free (calibrate->file[j]);
02346     }
02347     g_free (calibrate->error_old);
02348     g_free (calibrate->value_old);
02349     g_free (calibrate->value);
02350     g_free (calibrate->genetic_variable);
02351     g_free (calibrate->rangemax);
02352     g_free (calibrate->rangemin);
02353     #if DEBUG
02354     fprintf (stderr, "calibrate_free: end\n");
02355 #endif
02356 }
02357
02362 void
02363 calibrate_open ()
02364 {
02365     GTimeZone *tz;
02366     GDateTime *t0, *t;
02367     unsigned int i, j, *nbits;
02368
02369     #if DEBUG
02370     char *buffer;
02371     fprintf (stderr, "calibrate_open: start\n");
02372 #endif
02373
02374     // Getting initial time
02375     #if DEBUG
02376     fprintf (stderr, "calibrate_open: getting initial time\n");
02377 #endif
02378     tz = g_time_zone_new_utc ();
02379     t0 = g_date_time_new_now (tz);
02380
02381     // Obtaining and initing the pseudo-random numbers generator seed
02382     #if DEBUG
02383     fprintf (stderr, "calibrate_open: getting initial seed\n");
02384 #endif
02385     calibrate->seed = input->seed;
02386     gsl_rng_set (calibrate->rng, calibrate->seed);
02387
02388     // Replacing the working directory
02389     #if DEBUG
02390     fprintf (stderr, "calibrate_open: replacing the working directory\n");
02391 #endif
02392     g_chdir (input->directory);
02393
02394     // Getting results file names
02395     calibrate->result = input->result;
02396     calibrate->variables = input->variables;
02397
02398     // Obtaining the simulator file
02399     calibrate->simulator = input->simulator;
02400
02401     // Obtaining the evaluator file
02402     calibrate->evaluator = input->evaluator;
02403

```

```

02404 // Reading the algorithm
02405 calibrate->algorithm = input->algorithm;
02406 switch (calibrate->algorithm)
02407 {
02408     case ALGORITHM_MONTE_CARLO:
02409         calibrate_algorithm = calibrate_MonteCarlo;
02410         break;
02411     case ALGORITHM_SWEEP:
02412         calibrate_algorithm = calibrate_sweep;
02413         break;
02414     default:
02415         calibrate_algorithm = calibrate_genetic;
02416         calibrate->mutation_ratio = input->mutation_ratio;
02417         calibrate->reproduction_ratio = input->
reproduction_ratio;
02418         calibrate->adaptation_ratio = input->adaptation_ratio;
02419     }
02420 calibrate->nvariables = input->nvariables;
02421 calibrate->nsimulations = input->nsimulations;
02422 calibrate->niterations = input->niterations;
02423 calibrate->nbest = input->nbest;
02424 calibrate->tolerance = input->tolerance;
02425 calibrate->nsteps = input->nsteps;
02426 calibrate->nestimates = 0;
02427 if (input->nsteps)
02428 {
02429     calibrate->gradient_method = input->gradient_method;
02430     calibrate->relaxation = input->relaxation;
02431     switch (input->gradient_method)
02432     {
02433         case GRADIENT_METHOD_COORDINATES:
02434             calibrate->nestimates = 2 * calibrate->nvariables;
02435             calibrate_estimate_gradient =
calibrate_estimate_gradient_coordinates;
02436             break;
02437         default:
02438             calibrate->nestimates = input->nestimates;
02439             calibrate_estimate_gradient =
calibrate_estimate_gradient_random;
02440     }
02441 }
02442
02443 #if DEBUG
02444 fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02445 #endif
02446 calibrate->simulation_best
02447 = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02448 calibrate->error_best
02449 = (double *) alloca (calibrate->nbest * sizeof (double));
02450
02451 // Reading the experimental data
02452 #if DEBUG
02453 buffer = g_get_current_dir ();
02454 fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02455 g_free (buffer);
02456 #endif
02457 calibrate->nexperiments = input->nexperiments;
02458 calibrate->ninputs = input->ninputs;
02459 calibrate->experiment = input->experiment;
02460 calibrate->weight = input->weight;
02461 for (i = 0; i < input->ninputs; ++i)
02462 {
02463     calibrate->template[i] = input->template[i];
02464     calibrate->file[i]
02465     = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02466 }
02467 for (i = 0; i < input->nexperiments; ++i)
02468 {
02469     #if DEBUG
02470         fprintf (stderr, "calibrate_open: i=%u\n", i);
02471         fprintf (stderr, "calibrate_open: experiment=%s\n",
calibrate->experiment[i]);
02472         fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
02473     #endif
02474     for (j = 0; j < input->ninputs; ++j)
02475     {
02476         #if DEBUG
02477             fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02478             fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
i, j + 1, calibrate->template[j][i]);
02479         #endif
02480         calibrate->file[j][i]
02481         = g_mapped_file_new (input->template[j][i], 0, NULL);
02482     }
02483 }
02484 }
02485 }
02486
02487 // Reading the variables data

```

```

02488 #if DEBUG
02489     fprintf (stderr, "calibrate_open: reading variables\n");
02490 #endif
02491     calibrate->label = input->label;
02492     j = input->nvariables * sizeof (double);
02493     calibrate->rangemin = (double *) g_malloc (j);
02494     calibrate->rangemax = (double *) g_malloc (j);
02495     memcpy (calibrate->rangemin, input->rangemin, j);
02496     memcpy (calibrate->rangemax, input->rangemax, j);
02497     calibrate->rangeminabs = input->rangeminabs;
02498     calibrate->rangemaxabs = input->rangemaxabs;
02499     calibrate->precision = input->precision;
02500     calibrate->nsweeps = input->nsweeps;
02501     calibrate->step = input->step;
02502     nbits = input->nbits;
02503     if (input->algorithm == ALGORITHM_SWEEP)
02504     {
02505         calibrate->nsimulations = 1;
02506         for (i = 0; i < input->nvariables; ++i)
02507         {
02508             if (input->algorithm == ALGORITHM_SWEEP)
02509             {
02510                 calibrate->nsimulations *= input->nsweeps[i];
02511             }
02512             #if DEBUG
02513                 fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
02514                     calibrate->nsweeps[i], calibrate->nsimulations);
02515             #endif
02516         }
02517     }
02518     if (calibrate->nsteps)
02519         calibrate->gradient
02520             = (double *) alloca (calibrate->nvariables * sizeof (double));
02521
02522     // Allocating values
02523     #if DEBUG
02524         fprintf (stderr, "calibrate_open: allocating variables\n");
02525         fprintf (stderr, "calibrate_open: nvariables=%u\n", calibrate->nvariables);
02526     #endif
02527     calibrate->genetic_variable = NULL;
02528     if (calibrate->algorithm == ALGORITHM_GENETIC)
02529     {
02530         calibrate->genetic_variable = (GeneticVariable *)
02531             g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02532         for (i = 0; i < calibrate->nvariables; ++i)
02533         {
02534             #if DEBUG
02535                 fprintf (stderr, "calibrate_open: i=%u min=%lg max=%lg nbits=%u\n",
02536                     i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02537             #endif
02538             calibrate->genetic_variable[i].minimum = calibrate->
02539                 rangemin[i];
02540             calibrate->genetic_variable[i].maximum = calibrate->
02541                 rangemax[i];
02542             calibrate->genetic_variable[i].nbits = nbits[i];
02543         }
02544         #if DEBUG
02545             fprintf (stderr, "calibrate_open: nvariables=%u nsimulations=%u\n",
02546                 calibrate->nvariables, calibrate->nsimulations);
02547         #endif
02548         calibrate->value = (double *)
02549             g_malloc ((calibrate->nsimulations
02550                 + calibrate->nestimates * calibrate->nsteps)
02551                 * calibrate->nvariables * sizeof (double));
02552
02553         // Calculating simulations to perform on each task
02554         #if HAVE_MPI
02555             #if DEBUG
02556                 fprintf (stderr, "calibrate_open: rank=%u ntasks=%u\n",
02557                     calibrate->mpi_rank, ntasks);
02558             #endif
02559             calibrate->nstart = calibrate->mpi_rank * calibrate->
02560                 nsimulations / ntasks;
02561             calibrate->nend
02562                 = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
02563                 ntasks;
02564             if (calibrate->nsteps)
02565             {
02566                 calibrate->nstart_gradient
02567                     = calibrate->mpi_rank * calibrate->nestimates / ntasks;
02568                 calibrate->nend_gradient
02569                     = (1 + calibrate->mpi_rank) * calibrate->nestimates /
02570                     ntasks;
02571             }
02572         #else
02573             calibrate->nstart = 0;
02574         #endif
02575     }

```

```

02570     calibrate->nend = calibrate->nsimulations;
02571     if (calibrate->nsteps)
02572     {
02573         calibrate->nstart_gradient = 0;
02574         calibrate->nend_gradient = calibrate->nestimates;
02575     }
02576 #endif
02577 #if DEBUG
02578     fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02579             calibrate->nend);
02580 #endif
02581
02582     // Calculating simulations to perform for each thread
02583     calibrate->thread
02584     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02585     for (i = 0; i <= nthreads; ++i)
02586     {
02587         calibrate->thread[i] = calibrate->nstart
02588             + i * (calibrate->nend - calibrate->nstart) / nthreads;
02589 #if DEBUG
02590         fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02591                 calibrate->thread[i]);
02592 #endif
02593     }
02594     if (calibrate->nsteps)
02595         calibrate->thread_gradient = (unsigned int *)
02596             alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02597
02598     // Opening result files
02599     calibrate->file_result = g_fopen (calibrate->result, "w");
02600     calibrate->file_variables = g_fopen (calibrate->variables, "w");
02601
02602     // Performing the algorithm
02603     switch (calibrate->algorithm)
02604     {
02605         // Genetic algorithm
02606         case ALGORITHM_GENETIC:
02607             calibrate_genetic ();
02608             break;
02609
02610         // Iterative algorithm
02611         default:
02612             calibrate_iterate ();
02613     }
02614
02615     // Getting calculation time
02616     t = g_date_time_new_now (tz);
02617     calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02618     g_date_time_unref (t);
02619     g_date_time_unref (t0);
02620     g_time_zone_unref (tz);
02621     printf ("%s = %.6lg s\n",
02622             gettext ("Calculation time"), calibrate->calculation_time);
02623     fprintf (calibrate->file_result, "%s = %.6lg s\n",
02624             gettext ("Calculation time"), calibrate->calculation_time);
02625
02626     // Closing result files
02627     fclose (calibrate->file_variables);
02628     fclose (calibrate->file_result);
02629
02630 #if DEBUG
02631     fprintf (stderr, "calibrate_open: end\n");
02632 #endif
02633 }
02634
02635 #if HAVE_GTK
02636
02643 void
02644 input_save_gradient (xmlNode * node)
02645 {
02646     if (input->nsteps)
02647     {
02648         xml_node_set_uint (node, XML_NSTEPS, input->
02649 nsteps);
02649         if (input->relaxation != DEFAULT_RELAXATION)
02650             xml_node_set_float (node, XML_RELAXATION, input->
02651 relaxation);
02651         switch (input->gradient_method)
02652         {
02653             case GRADIENT_METHOD_COORDINATES:
02654                 xmlSetProp (node, XML_GRADIENT_METHOD,
02655 XML_COORDINATES);
02655                 break;
02656             default:
02657                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02658                 xml_node_set_uint (node, XML_NESTIMATES, input->
02659 nestimates);

```



```

02659     }
02660 }
02661 }
02662
02663 void
02670 input_save (char *filename)
02671 {
02672     unsigned int i, j;
02673     char *buffer;
02674     xmlDoc *doc;
02675     xmlNode *node, *child;
02676     GFile *file, *file2;
02677
02678     // Getting the input file directory
02679     input->name = g_path_get_basename (filename);
02680     input->directory = g_path_get_dirname (filename);
02681     file = g_file_new_for_path (input->directory);
02682
02683     // Opening the input file
02684     doc = xmlNewDoc ((const xmlChar *) "1.0");
02685
02686     // Setting root XML node
02687     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02688     xmlDocSetRootElement (doc, node);
02689
02690     // Adding properties to the root XML node
02691     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02692         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02693     if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02694         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02695     file2 = g_file_new_for_path (input->simulator);
02696     buffer = g_file_get_relative_path (file, file2);
02697     g_object_unref (file2);
02698     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02699     g_free (buffer);
02700     if (input->evaluator)
02701     {
02702         file2 = g_file_new_for_path (input->evaluator);
02703         buffer = g_file_get_relative_path (file, file2);
02704         g_object_unref (file2);
02705         if (xmlStrlen ((xmlChar *) buffer))
02706             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02707         g_free (buffer);
02708     }
02709     if (input->seed != DEFAULT_RANDOM_SEED)
02710         xml_node_set_uint (node, XML_SEED, input->seed);
02711
02712     // Setting the algorithm
02713     buffer = (char *) g_malloc (64);
02714     switch (input->algorithm)
02715     {
02716     case ALGORITHM_MONTE_CARLO:
02717         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02718         snprintf (buffer, 64, "%u", input->nsimulations);
02719         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02720         snprintf (buffer, 64, "%u", input->niterations);
02721         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02722         snprintf (buffer, 64, "%.3lg", input->tolerance);
02723         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02724         snprintf (buffer, 64, "%u", input->nbest);
02725         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02726         input_save_gradient (node);
02727         break;
02728     case ALGORITHM_SWEEP:
02729         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02730         snprintf (buffer, 64, "%u", input->niterations);
02731         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02732         snprintf (buffer, 64, "%.3lg", input->tolerance);
02733         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02734         snprintf (buffer, 64, "%u", input->nbest);
02735         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02736         input_save_gradient (node);
02737         break;
02738     default:
02739         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02740         snprintf (buffer, 64, "%u", input->nsimulations);
02741         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02742         snprintf (buffer, 64, "%u", input->niterations);
02743         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02744         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02745         xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02746         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02747         xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02748         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02749         xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02750         break;
02751     }

```

```

02752 g_free (buffer);
02753
02754 // Setting the experimental data
02755 for (i = 0; i < input->nexperiments; ++i)
02756 {
02757     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02758     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02759     if (input->weight[i] != 1.)
02760         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02761     for (j = 0; j < input->ninputs; ++j)
02762         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02763 }
02764
02765 // Setting the variables data
02766 for (i = 0; i < input->nvariables; ++i)
02767 {
02768     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02769     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02770     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02771     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02772         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
rangeminabs[i]);
02773     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02774     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02775         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
rangemaxabs[i]);
02776     if (input->precision[i] != DEFAULT_PRECISION)
02777         xml_node_set_uint (child, XML_PRECISION, input->
precision[i]);
02778     if (input->algorithm == ALGORITHM_SWEEP)
02779         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02780     else if (input->algorithm == ALGORITHM_GENETIC)
02781         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02782 }
02783
02784 // Saving the XML file
02785 xmlSaveFormatFile (filename, doc, 1);
02786
02787 // Freeing memory
02788 xmlFreeDoc (doc);
02789 }
02790
02795 void
02796 options_new ()
02797 {
02798     options->label_seed = (GtkLabel *)
02799         gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02800     options->spin_seed = (GtkSpinButton *)
02801         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02802     gtk_widget_set_tooltip_text
02803         (GTK_WIDGET (options->spin_seed),
02804          gettext ("Seed to init the pseudo-random numbers generator"));
02805     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02806     options->label_threads = (GtkLabel *)
02807         gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02808     options->spin_threads
02809         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02810     gtk_widget_set_tooltip_text
02811         (GTK_WIDGET (options->spin_threads),
02812          gettext ("Number of threads to perform the calibration/optimization for "
02813                  "the stochastic algorithm"));
02814     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
02815     options->label_gradient = (GtkLabel *)
02816         gtk_label_new (gettext ("Threads number for the gradient based method"));
02817     options->spin_gradient
02818         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02819     gtk_widget_set_tooltip_text
02820         (GTK_WIDGET (options->spin_gradient),
02821          gettext ("Number of threads to perform the calibration/optimization for "
02822                  "the gradient based method"));
02823     gtk_spin_button_set_value (options->spin_gradient,
02824                                (gdouble) nthreads_gradient);
02825     options->grid = (GtkGrid *) gtk_grid_new ();
02826     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
02827     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
02828     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
02829                      0, 1, 1, 1);
02830     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
02831                      1, 1, 1, 1);
02832     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
02833                      0, 2, 1, 1);

```

```

02834     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
02835                      1, 2, 1, 1);
02836     gtk_widget_show_all (GTK_WIDGET (options->grid));
02837     options->dialog = (GtkDialog *)
02838         gtk_dialog_new_with_buttons (gettext ("Options"),
02839                                     window->window,
02840                                     GTK_DIALOG_MODAL,
02841                                     gettext ("_OK"), GTK_RESPONSE_OK,
02842                                     gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02843                                     NULL);
02844     gtk_container_add
02845         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02846          GTK_WIDGET (options->grid));
02847     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02848     {
02849         input->seed
02850             = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02851         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
02852         nthreads_gradient
02853             = gtk_spin_button_get_value_as_int (options->spin_gradient);
02854     }
02855     gtk_widget_destroy (GTK_WIDGET (options->dialog));
02856 }
02857
02862 void
02863 running_new ()
02864 {
02865     #if DEBUG
02866         fprintf (stderr, "running_new: start\n");
02867     #endif
02868     running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02869     running->dialog = (GtkDialog *)
02870         gtk_dialog_new_with_buttons (gettext ("Calculating"),
02871                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
02872     gtk_container_add
02873         (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02874          GTK_WIDGET (running->label));
02875     gtk_widget_show_all (GTK_WIDGET (running->dialog));
02876     #if DEBUG
02877         fprintf (stderr, "running_new: end\n");
02878     #endif
02879 }
02880
02886 int
02887 window_get_algorithm ()
02888 {
02889     unsigned int i;
02890     for (i = 0; i < NALGORITHMS; ++i)
02891         if (gtk_toggle_button_get_active
02892             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02893             break;
02894     return i;
02895 }
02896
02902 int
02903 window_get_gradient ()
02904 {
02905     unsigned int i;
02906     for (i = 0; i < NGRADIENTS; ++i)
02907         if (gtk_toggle_button_get_active
02908             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02909             break;
02910     return i;
02911 }
02912
02917 void
02918 window_save_gradient ()
02919 {
02920     #if DEBUG
02921         fprintf (stderr, "window_save_gradient: start\n");
02922     #endif
02923     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02924     {
02925         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02926         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
02927         switch (window_get_gradient ())
02928         {
02929             case GRADIENT_METHOD_COORDINATES:
02930                 input->gradient_method = GRADIENT_METHOD_COORDINATES;
02931                 break;
02932             default:
02933                 input->gradient_method = GRADIENT_METHOD_RANDOM;
02934                 input->nestimates
02935                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
02936         }
02937     }

```

```

02938     else
02939         input->nsteps = 0;
02940 #if DEBUG
02941     fprintf (stderr, "window_save_gradient: end\n");
02942 #endif
02943 }
02944
02945 int
02951 window_save ()
02952 {
02953     char *buffer;
02954     GtkFileChooserDialog *dlg;
02955
02956 #if DEBUG
02957     fprintf (stderr, "window_save: start\n");
02958 #endif
02959
02960     // Opening the saving dialog
02961     dlg = (GtkFileChooserDialog *)
02962         gtk_file_chooser_dialog_new (gettext ("Save file"),
02963                                     window->window,
02964                                     GTK_FILE_CHOOSER_ACTION_SAVE,
02965                                     gettext ("Cancel"),
02966                                     GTK_RESPONSE_CANCEL,
02967                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
02968     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02969     buffer = g_build_filename (input->directory, input->name, NULL);
02970     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02971     g_free (buffer);
02972
02973     // If OK response then saving
02974     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02975     {
02976
02977         // Adding properties to the root XML node
02978         input->simulator = gtk_file_chooser_get_filename
02979             (GTK_FILE_CHOOSER (window->button_simulator));
02980         if (gtk_toggle_button_get_active
02981             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02982             input->evaluator = gtk_file_chooser_get_filename
02983                 (GTK_FILE_CHOOSER (window->button_evaluator));
02984         else
02985             input->evaluator = NULL;
02986         input->result
02987             = (char *) xmlStrdup ((const xmlChar *)
02988                                   gtk_entry_get_text (window->entry_result));
02989         input->variables
02990             = (char *) xmlStrdup ((const xmlChar *)
02991                                   gtk_entry_get_text (window->entry_variables));
02992
02993         // Setting the algorithm
02994         switch (window_get_algorithm ())
02995         {
02996             case ALGORITHM_MONTE_CARLO:
02997                 input->algorithm = ALGORITHM_MONTE_CARLO;
02998                 input->nsimulations
02999                     = gtk_spin_button_get_value_as_int (window->spin_simulations);
03000                 input->niterations
03001                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002                 input->tolerance = gtk_spin_button_get_value (window->
03003 spin_tolerance);
03004                 input->nbest = gtk_spin_button_get_value_as_int (window->
03005 spin_bests);
03006                 window_save_gradient ();
03007                 break;
03008             case ALGORITHM_SWEEP:
03009                 input->algorithm = ALGORITHM_SWEEP;
03010                 input->niterations
03011                     = gtk_spin_button_get_value_as_int (window->spin_iterations);
03012                 input->tolerance = gtk_spin_button_get_value (window->
03013 spin_tolerance);
03014                 input->nbest = gtk_spin_button_get_value_as_int (window->
03015 spin_bests);
03016                 window_save_gradient ();
03017                 break;
03018             default:
03019                 input->algorithm = ALGORITHM_GENETIC;
03020                 input->nsimulations
03021                     = gtk_spin_button_get_value_as_int (window->spin_population);
03022                 input->niterations
03023                     = gtk_spin_button_get_value_as_int (window->spin_generations);
03024                 input->mutation_ratio
03025                     = gtk_spin_button_get_value (window->spin_mutation);
03026                 input->reproduction_ratio
03027                     = gtk_spin_button_get_value (window->spin_reproduction);
03028                 input->adaptation_ratio
03029                     = gtk_spin_button_get_value (window->spin_adaptation);

```

```

03026         break;
03027     }
03028
03029     // Saving the XML file
03030     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03031     input_save (buffer);
03032
03033     // Closing and freeing memory
03034     g_free (buffer);
03035     gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037     fprintf (stderr, "window_save: end\n");
03038 #endif
03039     return 1;
03040 }
03041
03042 // Closing and freeing memory
03043 gtk_widget_destroy (GTK_WIDGET (dlg));
03044 #if DEBUG
03045     fprintf (stderr, "window_save: end\n");
03046 #endif
03047     return 0;
03048 }
03049
03050 void
03051 window_run ()
03052 {
03053     unsigned int i;
03054     char *msg, *msg2, buffer[64], buffer2[64];
03055 #if DEBUG
03056     fprintf (stderr, "window_run: start\n");
03057 #endif
03058     if (!window_save ())
03059     {
03060         #if DEBUG
03061             fprintf (stderr, "window_run: end\n");
03062         #endif
03063         return;
03064     }
03065     running_new ();
03066     while (gtk_events_pending ())
03067         gtk_main_iteration ();
03068     calibrate_open ();
03069     gtk_widget_destroy (GTK_WIDGET (running->dialog));
03070     snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03071     msg2 = g_strdup (buffer);
03072     for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03073     {
03074         snprintf (buffer, 64, "%s = %s\n",
03075                 calibrate->label[i], format[calibrate->precision[i]]);
03076         snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03077         msg = g_strconcat (msg2, buffer2, NULL);
03078         g_free (msg2);
03079     }
03080     snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03081             calibrate->calculation_time);
03082     msg = g_strconcat (msg2, buffer, NULL);
03083     g_free (msg2);
03084     show_message (gettext ("Best result"), msg, INFO_TYPE);
03085     g_free (msg);
03086     calibrate_free ();
03087 #if DEBUG
03088     fprintf (stderr, "window_run: end\n");
03089 #endif
03090 }
03091
03092 void
03093 window_help ()
03094 {
03095     char *buffer, *buffer2;
03096     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03097                               gettext ("user-manual.pdf"), NULL);
03098     buffer = g_filename_to_uri (buffer2, NULL, NULL);
03099     g_free (buffer2);
03100     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03101     g_free (buffer);
03102 }
03103
03104 void
03105 window_about ()
03106 {
03107     static const gchar *authors[] = {
03108         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03109         "Borja Latorre Garcés <borja.latorre@csic.es>",
03110         NULL
03111     };
03112     gtk_show_about_dialog

```

```

03125     (window->window,
03126      "program_name", "MPCOTool",
03127      "comments",
03128      gettext ("A software to perform calibrations/optimizations of empirical "
03129              "parameters"),
03130      "authors", authors,
03131      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03132      "version", "1.2.1",
03133      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03134      "logo", window->logo,
03135      "website", "https://github.com/jburguete/mpcotool",
03136      "license-type", GTK_LICENSE_BSD, NULL);
03137 }
03138
03144 void
03145 window_update_gradient ()
03146 {
03147     gtk_widget_show (GTK_WIDGET (window->check_gradient));
03148     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03149         gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03150     switch (window_get_gradient ())
03151     {
03152     case GRADIENT_METHOD_COORDINATES:
03153         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03154         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03155         break;
03156     default:
03157         gtk_widget_show (GTK_WIDGET (window->label_estimates));
03158         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03159     }
03160 }
03161
03166 void
03167 window_update ()
03168 {
03169     unsigned int i;
03170     gtk_widget_set_sensitive
03171         (GTK_WIDGET (window->button_evaluator),
03172          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03173              (window->check_evaluator)));
03174     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03175     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03176     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03177     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03178     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03179     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03180     gtk_widget_hide (GTK_WIDGET (window->label_bests));
03181     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03182     gtk_widget_hide (GTK_WIDGET (window->label_population));
03183     gtk_widget_hide (GTK_WIDGET (window->spin_population));
03184     gtk_widget_hide (GTK_WIDGET (window->label_generations));
03185     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03186     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03187     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03188     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03189     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03190     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03191     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03192     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03193     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03194     gtk_widget_hide (GTK_WIDGET (window->label_bits));
03195     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03196     gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03197     gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03198     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03199     switch (window_get_algorithm ())
03200     {
03201     case ALGORITHM_MONTE_CARLO:
03202         gtk_widget_show (GTK_WIDGET (window->label_simulations));
03203         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03204         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03205         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03206         if (i > 1)
03207         {
03208             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03209             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03210             gtk_widget_show (GTK_WIDGET (window->label_bests));
03211             gtk_widget_show (GTK_WIDGET (window->spin_bests));
03212         }
03213         window_update_gradient ();
03214         break;
03215     case ALGORITHM_SWEEP:
03216         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03217         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03218         if (i > 1)
03219         {
03220             gtk_widget_show (GTK_WIDGET (window->label_tolerance));

```

```

03221         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03222         gtk_widget_show (GTK_WIDGET (window->label_bests));
03223         gtk_widget_show (GTK_WIDGET (window->spin_bests));
03224     }
03225     gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03226     gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03227     gtk_widget_show (GTK_WIDGET (window->check_gradient));
03228     window_update_gradient ();
03229     break;
03230 default:
03231     gtk_widget_show (GTK_WIDGET (window->label_population));
03232     gtk_widget_show (GTK_WIDGET (window->spin_population));
03233     gtk_widget_show (GTK_WIDGET (window->label_generations));
03234     gtk_widget_show (GTK_WIDGET (window->spin_generations));
03235     gtk_widget_show (GTK_WIDGET (window->label_mutation));
03236     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03237     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03238     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03239     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03240     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03241     gtk_widget_show (GTK_WIDGET (window->label_bits));
03242     gtk_widget_show (GTK_WIDGET (window->spin_bits));
03243 }
03244 gtk_widget_set_sensitive
03245 (GTK_WIDGET (window->button_remove_experiment), input->
n experiments > 1);
03246 gtk_widget_set_sensitive
03247 (GTK_WIDGET (window->button_remove_variable), input->
n variables > 1);
03248 for (i = 0; i < input->ninputs; ++i)
03249 {
03250     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03251     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03252     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03253     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03254     g_signal_handler_block
03255 (window->check_template[i], window->id_template[i]);
03256     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03257     gtk_toggle_button_set_active
03258 (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03259     g_signal_handler_unblock
03260 (window->button_template[i], window->id_input[i]);
03261     g_signal_handler_unblock
03262 (window->check_template[i], window->id_template[i]);
03263 }
03264 if (i > 0)
03265 {
03266     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03267     gtk_widget_set_sensitive
03268 (GTK_WIDGET (window->button_template[i - 1]),
03269      gtk_toggle_button_get_active
03270      (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
03271 }
03272 if (i < MAX_NINPUTS)
03273 {
03274     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03275     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03276     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03277     gtk_widget_set_sensitive
03278 (GTK_WIDGET (window->button_template[i]),
03279      gtk_toggle_button_get_active
03280      (GTK_TOGGLE_BUTTON (window->check_template[i])));
03281     g_signal_handler_block
03282 (window->check_template[i], window->id_template[i]);
03283     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03284     gtk_toggle_button_set_active
03285 (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03286     g_signal_handler_unblock
03287 (window->button_template[i], window->id_input[i]);
03288     g_signal_handler_unblock
03289 (window->check_template[i], window->id_template[i]);
03290 }
03291 while (++i < MAX_NINPUTS)
03292 {
03293     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03294     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03295 }
03296 gtk_widget_set_sensitive
03297 (GTK_WIDGET (window->spin_minabs),
03298  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03299 gtk_widget_set_sensitive
03300 (GTK_WIDGET (window->spin_maxabs),
03301  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03302 }
03303

```

```

03308 void
03309 window_set_algorithm ()
03310 {
03311     int i;
03312     #if DEBUG
03313     fprintf (stderr, "window_set_algorithm: start\n");
03314     #endif
03315     i = window_get_algorithm ();
03316     switch (i)
03317     {
03318     case ALGORITHM_SWEEP:
03319         input->nsweeps = (unsigned int *) g_realloc
03320             (input->nsweeps, input->nvariables * sizeof (unsigned int));
03321         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03322         if (i < 0)
03323             i = 0;
03324         gtk_spin_button_set_value (window->spin_sweeps,
03325             (gdouble) input->nsweeps[i]);
03326         break;
03327     case ALGORITHM_GENETIC:
03328         input->nbits = (unsigned int *) g_realloc
03329             (input->nbits, input->nvariables * sizeof (unsigned int));
03330         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03331         if (i < 0)
03332             i = 0;
03333         gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
03334             nbits[i]);
03335         window_update ();
03336         #if DEBUG
03337         fprintf (stderr, "window_set_algorithm: end\n");
03338         #endif
03339     }
03340 }
03341 void
03342 window_set_experiment ()
03343 {
03344     unsigned int i, j;
03345     char *buffer1, *buffer2;
03346     #if DEBUG
03347     fprintf (stderr, "window_set_experiment: start\n");
03348     #endif
03349     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03350     gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03351     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03352     buffer2 = g_build_filename (input->directory, buffer1, NULL);
03353     g_free (buffer1);
03354     g_signal_handler_block
03355         (window->button_experiment, window->id_experiment_name);
03356     gtk_file_chooser_set_filename
03357         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03358     g_signal_handler_unblock
03359         (window->button_experiment, window->id_experiment_name);
03360     g_free (buffer2);
03361     for (j = 0; j < input->ninputs; ++j)
03362     {
03363         g_signal_handler_block (window->button_template[j], window->
03364             id_input[j]);
03365         buffer2
03366             = g_build_filename (input->directory, input->template[j][i], NULL);
03367         gtk_file_chooser_set_filename
03368             (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03369         g_free (buffer2);
03370         g_signal_handler_unblock
03371             (window->button_template[j], window->id_input[j]);
03372     }
03373     #if DEBUG
03374     fprintf (stderr, "window_set_experiment: end\n");
03375     #endif
03376 }
03377 void
03378 window_remove_experiment ()
03379 {
03380     unsigned int i, j;
03381     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03382     g_signal_handler_block (window->combo_experiment, window->
03383         id_experiment);
03384     gtk_combo_box_text_remove (window->combo_experiment, i);
03385     g_signal_handler_unblock (window->combo_experiment, window->
03386         id_experiment);
03387     xmlFree (input->experiment[i]);
03388     --input->nexperiments;
03389     for (j = i; j < input->nexperiments; ++j)
03390     {
03391         input->experiment[j] = input->experiment[j + 1];
03392         input->weight[j] = input->weight[j + 1];
03393     }

```



```

03399     }
03400     j = input->nexperiments - 1;
03401     if (i > j)
03402         i = j;
03403     for (j = 0; j < input->ninputs; ++j)
03404         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03405     g_signal_handler_block
03406         (window->button_experiment, window->id_experiment_name);
03407     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03408     g_signal_handler_unblock
03409         (window->button_experiment, window->id_experiment_name);
03410     for (j = 0; j < input->ninputs; ++j)
03411         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03412     window_update ();
03413 }
03414
03419 void
03420 window_add_experiment ()
03421 {
03422     unsigned int i, j;
03423     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03424     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03425     gtk_combo_box_text_insert_text
03426         (window->combo_experiment, i, input->experiment[i]);
03427     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03428     input->experiment = (char **) g_realloc
03429         (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03430     input->weight = (double *) g_realloc
03431         (input->weight, (input->nexperiments + 1) * sizeof (double));
03432     for (j = input->nexperiments - 1; j > i; --j)
03433     {
03434         input->experiment[j + 1] = input->experiment[j];
03435         input->weight[j + 1] = input->weight[j];
03436     }
03437     input->experiment[j + 1]
03438         = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03439     input->weight[j + 1] = input->weight[j];
03440     ++input->nexperiments;
03441     for (j = 0; j < input->ninputs; ++j)
03442         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03443     g_signal_handler_block
03444         (window->button_experiment, window->id_experiment_name);
03445     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03446     g_signal_handler_unblock
03447         (window->button_experiment, window->id_experiment_name);
03448     for (j = 0; j < input->ninputs; ++j)
03449         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03450     window_update ();
03451 }
03452
03457 void
03458 window_name_experiment ()
03459 {
03460     unsigned int i;
03461     char *buffer;
03462     GFile *file1, *file2;
03463     #if DEBUG
03464     fprintf (stderr, "window_name_experiment: start\n");
03465     #endif
03466     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03467     file1
03468         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03469     file2 = g_file_new_for_path (input->directory);
03470     buffer = g_file_get_relative_path (file2, file1);
03471     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03472     gtk_combo_box_text_remove (window->combo_experiment, i);
03473     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03474     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03475     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03476     g_free (buffer);
03477     g_object_unref (file2);
03478     g_object_unref (file1);
03479     #if DEBUG
03480     fprintf (stderr, "window_name_experiment: end\n");
03481     #endif
03482 }
03483
03488 void
03489 window_weight_experiment ()

```

```

03490 {
03491     unsigned int i;
03492     #if DEBUG
03493         fprintf (stderr, "window_weight_experiment: start\n");
03494     #endif
03495     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03496     input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03497     #if DEBUG
03498         fprintf (stderr, "window_weight_experiment: end\n");
03499     #endif
03500 }
03501
03502 void
03503 window_inputs_experiment ()
03504 {
03505     unsigned int j;
03506     #if DEBUG
03507         fprintf (stderr, "window_inputs_experiment: start\n");
03508     #endif
03509     j = input->ninputs - 1;
03510     if (j
03511         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03512                                           (window->check_template[j]))
03513         --input->ninputs;
03514     if (input->ninputs < MAX_NINPUTS
03515         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03516                                           (window->check_template[j]))
03517     {
03518         ++input->ninputs;
03519         for (j = 0; j < input->ninputs; ++j)
03520         {
03521             input->template[j] = (char **)
03522                 g_realloc (input->template[j], input->nvariables * sizeof (char *));
03523         }
03524     }
03525     window_update ();
03526     #if DEBUG
03527         fprintf (stderr, "window_inputs_experiment: end\n");
03528     #endif
03529 }
03530
03531 void
03532 window_template_experiment (void *data)
03533 {
03534     unsigned int i, j;
03535     char *buffer;
03536     GFile *file1, *file2;
03537     #if DEBUG
03538         fprintf (stderr, "window_template_experiment: start\n");
03539     #endif
03540     i = (size_t) data;
03541     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03542     file1
03543         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03544     file2 = g_file_new_for_path (input->directory);
03545     buffer = g_file_get_relative_path (file2, file1);
03546     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03547     g_free (buffer);
03548     g_object_unref (file2);
03549     g_object_unref (file1);
03550     #if DEBUG
03551         fprintf (stderr, "window_template_experiment: end\n");
03552     #endif
03553 }
03554
03555 void
03556 window_set_variable ()
03557 {
03558     unsigned int i;
03559     #if DEBUG
03560         fprintf (stderr, "window_set_variable: start\n");
03561     #endif
03562     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03563     g_signal_handler_block (window->entry_variable, window->
03564                             id_variable_label);
03565     gtk_entry_set_text (window->entry_variable, input->label[i]);
03566     g_signal_handler_unblock (window->entry_variable, window->
03567                              id_variable_label);
03568     gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03569     gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03570     if (input->rangeminabs[i] != -G_MAXDOUBLE)
03571     {
03572         gtk_spin_button_set_value (window->spin_minabs, input->
03573                                   rangeminabs[i]);
03574         gtk_toggle_button_set_active
03575             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03576     }
03577 }

```

```

03590     else
03591     {
03592         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03593         gtk_toggle_button_set_active
03594             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03595     }
03596     if (input->rangemaxabs[i] != G_MAXDOUBLE)
03597     {
03598         gtk_spin_button_set_value (window->spin_maxabs, input->
rangemaxabs[i]);
03599         gtk_toggle_button_set_active
03600             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03601     }
03602     else
03603     {
03604         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03605         gtk_toggle_button_set_active
03606             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03607     }
03608     gtk_spin_button_set_value (window->spin_precision, input->
precision[i]);
03609     #if DEBUG
03610     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03611             input->precision[i]);
03612     #endif
03613     switch (window_get_algorithm ())
03614     {
03615         case ALGORITHM_SWEEP:
03616             gtk_spin_button_set_value (window->spin_sweeps,
03617                                         (gdouble) input->nsweeps[i]);
03618             #if DEBUG
03619             fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03620                     input->nsweeps[i]);
03621             #endif
03622             break;
03623         case ALGORITHM_GENETIC:
03624             gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
nbits[i]);
03625             #if DEBUG
03626             fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03627                     input->nbits[i]);
03628             #endif
03629             break;
03630     }
03631     window_update ();
03632     #if DEBUG
03633     fprintf (stderr, "window_set_variable: end\n");
03634     #endif
03635 }
03636
03641 void
03642 window_remove_variable ()
03643 {
03644     unsigned int i, j;
03645     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03646     g_signal_handler_block (window->combo_variable, window->
id_variable);
03647     gtk_combo_box_text_remove (window->combo_variable, i);
03648     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03649     xmlFree (input->label[i]);
03650     --input->nvariables;
03651     for (j = i; j < input->nvariables; ++j)
03652     {
03653         input->label[j] = input->label[j + 1];
03654         input->rangemin[j] = input->rangemin[j + 1];
03655         input->rangemax[j] = input->rangemax[j + 1];
03656         input->rangeminabs[j] = input->rangeminabs[j + 1];
03657         input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03658         input->precision[j] = input->precision[j + 1];
03659         switch (window_get_algorithm ())
03660         {
03661             case ALGORITHM_SWEEP:
03662                 input->nsweeps[j] = input->nsweeps[j + 1];
03663                 break;
03664             case ALGORITHM_GENETIC:
03665                 input->nbits[j] = input->nbits[j + 1];
03666             }
03667     }
03668     j = input->nvariables - 1;
03669     if (i > j)
03670         i = j;
03671     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03672     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03673     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);

```

```

03674     window_update ();
03675 }
03676
03681 void
03682 window_add_variable ()
03683 {
03684     unsigned int i, j;
03685     #if DEBUG
03686     fprintf (stderr, "window_add_variable: start\n");
03687     #endif
03688     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03689     g_signal_handler_block (window->combo_variable, window->
id_variable);
03690     gtk_combo_box_text_insert_text (window->combo_variable, i, input->
label[i]);
03691     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03692     input->label = (char **) g_realloc
(input->label, (input->nvariables + 1) * sizeof (char *));
03693     input->rangemin = (double *) g_realloc
(input->rangemin, (input->nvariables + 1) * sizeof (double));
03694     input->rangemax = (double *) g_realloc
(input->rangemax, (input->nvariables + 1) * sizeof (double));
03695     input->rangeminabs = (double *) g_realloc
(input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03696     input->rangemaxabs = (double *) g_realloc
(input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03697     input->precision = (unsigned int *) g_realloc
(input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03698     for (j = input->nvariables - 1; j > i; --j)
03699     {
03700         input->label[j + 1] = input->label[j];
03701         input->rangemin[j + 1] = input->rangemin[j];
03702         input->rangemax[j + 1] = input->rangemax[j];
03703         input->rangeminabs[j + 1] = input->rangeminabs[j];
03704         input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03705         input->precision[j + 1] = input->precision[j];
03706     }
03707     input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03708     input->rangemin[j + 1] = input->rangemin[j];
03709     input->rangemax[j + 1] = input->rangemax[j];
03710     input->rangeminabs[j + 1] = input->rangeminabs[j];
03711     input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03712     input->precision[j + 1] = input->precision[j];
03713     switch (window_get_algorithm ())
03714     {
03715     case ALGORITHM_SWEEP:
03716         input->nsweeps = (unsigned int *) g_realloc
(input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03717         for (j = input->nvariables - 1; j > i; --j)
03718             input->nsweeps[j + 1] = input->nsweeps[j];
03719         input->nsweeps[j + 1] = input->nsweeps[j];
03720         break;
03721     case ALGORITHM_GENETIC:
03722         input->nbits = (unsigned int *) g_realloc
(input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03723         for (j = input->nvariables - 1; j > i; --j)
03724             input->nbits[j + 1] = input->nbits[j];
03725         input->nbits[j + 1] = input->nbits[j];
03726     }
03727     ++input->nvariables;
03728     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03729     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03730     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03731     window_update ();
03732     #if DEBUG
03733     fprintf (stderr, "window_add_variable: end\n");
03734     #endif
03735 }
03736
03737 void
03738 window_label_variable ()
03739 {
03740     unsigned int i;
03741     const char *buffer;
03742     #if DEBUG
03743     fprintf (stderr, "window_label_variable: start\n");
03744     #endif
03745     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03746     buffer = gtk_entry_get_text (window->entry_variable);
03747     g_signal_handler_block (window->combo_variable, window->
id_variable);
03748     gtk_combo_box_text_remove (window->combo_variable, i);
03749     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03750     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);

```

```

03763 g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
03764 #if DEBUG
03765 fprintf (stderr, "window_label_variable: end\n");
03766 #endif
03767 }
03768
03773 void
03774 window_precision_variable ()
03775 {
03776     unsigned int i;
03777     #if DEBUG
03778     fprintf (stderr, "window_precision_variable: start\n");
03779     #endif
03780     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03781     input->precision[i]
03782     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03783     gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03784     gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03785     gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03786     gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03787     #if DEBUG
03788     fprintf (stderr, "window_precision_variable: end\n");
03789     #endif
03790 }
03791
03796 void
03797 window_rangemin_variable ()
03798 {
03799     unsigned int i;
03800     #if DEBUG
03801     fprintf (stderr, "window_rangemin_variable: start\n");
03802     #endif
03803     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03804     input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03805     #if DEBUG
03806     fprintf (stderr, "window_rangemin_variable: end\n");
03807     #endif
03808 }
03809
03814 void
03815 window_rangemax_variable ()
03816 {
03817     unsigned int i;
03818     #if DEBUG
03819     fprintf (stderr, "window_rangemax_variable: start\n");
03820     #endif
03821     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03822     input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03823     #if DEBUG
03824     fprintf (stderr, "window_rangemax_variable: end\n");
03825     #endif
03826 }
03827
03832 void
03833 window_rangeminabs_variable ()
03834 {
03835     unsigned int i;
03836     #if DEBUG
03837     fprintf (stderr, "window_rangeminabs_variable: start\n");
03838     #endif
03839     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03840     input->rangeminabs[i] = gtk_spin_button_get_value (window->
        spin_minabs);
03841     #if DEBUG
03842     fprintf (stderr, "window_rangeminabs_variable: end\n");
03843     #endif
03844 }
03845
03850 void
03851 window_rangemaxabs_variable ()
03852 {
03853     unsigned int i;
03854     #if DEBUG
03855     fprintf (stderr, "window_rangemaxabs_variable: start\n");
03856     #endif
03857     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03858     input->rangemaxabs[i] = gtk_spin_button_get_value (window->
        spin_maxabs);
03859     #if DEBUG
03860     fprintf (stderr, "window_rangemaxabs_variable: end\n");
03861     #endif
03862 }
03863
03868 void
03869 window_update_variable ()
03870 {

```

```

03871     int i;
03872     #if DEBUG
03873     fprintf (stderr, "window_update_variable: start\n");
03874     #endif
03875     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03876     if (i < 0)
03877         i = 0;
03878     switch (window_get_algorithm ())
03879     {
03880     case ALGORITHM_SWEEP:
03881         input->nsweeps[i]
03882         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03883     #if DEBUG
03884         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03885                 input->nsweeps[i]);
03886     #endif
03887         break;
03888     case ALGORITHM_GENETIC:
03889         input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03890     #if DEBUG
03891         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
03892                 input->nbits[i]);
03893     #endif
03894     }
03895     #if DEBUG
03896     fprintf (stderr, "window_update_variable: end\n");
03897     #endif
03898 }
03899
03900 int
03901 window_read (char *filename)
03902 {
03903     unsigned int i;
03904     char *buffer;
03905     #if DEBUG
03906     fprintf (stderr, "window_read: start\n");
03907     #endif
03908
03909     // Reading new input file
03910     input_free ();
03911     if (!input_open (filename))
03912         return 0;
03913
03914     // Setting GTK+ widgets data
03915     gtk_entry_set_text (window->entry_result, input->result);
03916     gtk_entry_set_text (window->entry_variables, input->variables);
03917     buffer = g_build_filename (input->directory, input->simulator, NULL);
03918     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03919                                   (window->button_simulator), buffer);
03920     g_free (buffer);
03921     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03922                                   (size_t) input->evaluator);
03923
03924     if (input->evaluator)
03925     {
03926         buffer = g_build_filename (input->directory, input->evaluator, NULL);
03927         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03928                                       (window->button_evaluator), buffer);
03929         g_free (buffer);
03930     }
03931     gtk_toggle_button_set_active
03932     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
03933 algorithm]), TRUE);
03934     switch (input->algorithm)
03935     {
03936     case ALGORITHM_MONTE_CARLO:
03937         gtk_spin_button_set_value (window->spin_simulations,
03938                                   (gdouble) input->nsimulations);
03939     case ALGORITHM_SWEEP:
03940         gtk_spin_button_set_value (window->spin_iterations,
03941                                   (gdouble) input->niterations);
03942         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
03943 nbest);
03944         gtk_spin_button_set_value (window->spin_tolerance, input->
03945 tolerance);
03946         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
03947                                   input->nsteps);
03948     #if DEBUG
03949         if (input->nsteps)
03950         {
03951             gtk_toggle_button_set_active
03952             (GTK_TOGGLE_BUTTON (window->button_gradient
03953 [input->gradient_method]), TRUE);
03954             gtk_spin_button_set_value (window->spin_steps,
03955                                       (gdouble) input->nsteps);
03956             gtk_spin_button_set_value (window->spin_relaxation,
03957                                       (gdouble) input->relaxation);
03958         }
03959     #endif
03960     }
03961 }

```

```

03962         case GRADIENT_METHOD_RANDOM:
03963             gtk_spin_button_set_value (window->spin_estimates,
03964                                     (gdouble) input->nestimates);
03965         }
03966     }
03967     break;
03968     default:
03969         gtk_spin_button_set_value (window->spin_population,
03970                                 (gdouble) input->nsimulations);
03971         gtk_spin_button_set_value (window->spin_generations,
03972                                 (gdouble) input->niterations);
03973         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
03974         gtk_spin_button_set_value (window->spin_reproduction,
03975                                 input->reproduction_ratio);
03976         gtk_spin_button_set_value (window->spin_adaptation,
03977                                 input->adaptation_ratio);
03978     }
03979     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03980     g_signal_handler_block (window->button_experiment,
03981                             window->id_experiment_name);
03982     gtk_combo_box_text_remove_all (window->combo_experiment);
03983     for (i = 0; i < input->nexperiments; ++i)
03984         gtk_combo_box_text_append_text (window->combo_experiment,
03985                                         input->experiment[i]);
03986     g_signal_handler_unblock
03987         (window->button_experiment, window->id_experiment_name);
03988     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03989     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03990     g_signal_handler_block (window->combo_variable, window->
id_variable);
03991     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03992     gtk_combo_box_text_remove_all (window->combo_variable);
03993     for (i = 0; i < input->nvariables; ++i)
03994         gtk_combo_box_text_append_text (window->combo_variable, input->
label[i]);
03995     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03996     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03997     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03998     window_set_variable ();
03999     window_update ();
04000
04001     #if DEBUG
04002     fprintf (stderr, "window_read: end\n");
04003     #endif
04004     return 1;
04005 }
04006
04011 void
04012 window_open ()
04013 {
04014     char *buffer, *directory, *name;
04015     GtkFileChooserDialog *dlg;
04016
04017     #if DEBUG
04018     fprintf (stderr, "window_open: start\n");
04019     #endif
04020
04021     // Saving a backup of the current input file
04022     directory = g_strdup (input->directory);
04023     name = g_strdup (input->name);
04024
04025     // Opening dialog
04026     dlg = (GtkFileChooserDialog *)
04027         gtk_file_chooser_dialog_new (gettext ("Open input file"),
04028                                     window->window,
04029                                     GTK_FILE_CHOOSER_ACTION_OPEN,
04030                                     gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04031                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04032     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04033     {
04034
04035         // Traying to open the input file
04036         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04037         if (!window_read (buffer))
04038         {
04039             #if DEBUG
04040             fprintf (stderr, "window_open: error reading input file\n");
04041             #endif
04042             g_free (buffer);
04043
04044             // Reading backup file on error

```

```

04045         buffer = g_build_filename (directory, name, NULL);
04046         if (!input_open (buffer))
04047         {
04048
04049             // Closing on backup file reading error
04050 #if DEBUG
04051             fprintf (stderr, "window_read: error reading backup file\n");
04052 #endif
04053             g_free (buffer);
04054             break;
04055         }
04056         g_free (buffer);
04057     }
04058     else
04059     {
04060         g_free (buffer);
04061         break;
04062     }
04063 }
04064
04065 // Freeing and closing
04066 g_free (name);
04067 g_free (directory);
04068 gtk_widget_destroy (GTK_WIDGET (dlg));
04069 #if DEBUG
04070 fprintf (stderr, "window_open: end\n");
04071 #endif
04072 }
04073
04074 void
04075 window_new ()
04076 {
04077     unsigned int i;
04078     char *buffer, *buffer2, buffer3[64];
04079     GtkViewport *viewport;
04080     char *label_algorithm[NALGORITHMS] = {
04081         "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04082     };
04083     char *tip_algorithm[NALGORITHMS] = {
04084         gettext ("Monte-Carlo brute force algorithm"),
04085         gettext ("Sweep brute force algorithm"),
04086         gettext ("Genetic algorithm")
04087     };
04088     char *label_gradient[NGRADIENTS] = {
04089         gettext ("_Coordinates descent"), gettext ("_Random")
04090     };
04091     char *tip_gradient[NGRADIENTS] = {
04092         gettext ("Coordinates descent gradient estimate method"),
04093         gettext ("Random gradient estimate method")
04094     };
04095
04096     // Creating the window
04097     window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04098
04099     // Finish when closing the window
04100     g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04101
04102     // Setting the window title
04103     gtk_window_set_title (window->window, PROGRAM_INTERFACE);
04104
04105     // Creating the open button
04106     window->button_open = (GtkToolButton *) gtk_tool_button_new
04107         (gtk_image_new_from_icon_name ("document-open",
04108             GTK_ICON_SIZE_LARGE_TOOLBAR),
04109         gettext ("Open"));
04110     g_signal_connect (window->button_open, "clicked", window_open, NULL);
04111
04112     // Creating the save button
04113     window->button_save = (GtkToolButton *) gtk_tool_button_new
04114         (gtk_image_new_from_icon_name ("document-save",
04115             GTK_ICON_SIZE_LARGE_TOOLBAR),
04116         gettext ("Save"));
04117     g_signal_connect (window->button_save, "clicked", (void *)
04118         window_save,
04119         NULL);
04120
04121     // Creating the run button
04122     window->button_run = (GtkToolButton *) gtk_tool_button_new
04123         (gtk_image_new_from_icon_name ("system-run",
04124             GTK_ICON_SIZE_LARGE_TOOLBAR),
04125         gettext ("Run"));
04126     g_signal_connect (window->button_run, "clicked", window_run, NULL);
04127
04128     // Creating the options button
04129     window->button_options = (GtkToolButton *) gtk_tool_button_new
04130         (gtk_image_new_from_icon_name ("preferences-system",
04131             GTK_ICON_SIZE_LARGE_TOOLBAR),

```



```

04135     gettext ("Options"));
04136     g_signal_connect (window->button_options, "clicked", options_new, NULL);
04137
04138     // Creating the help button
04139     window->button_help = (GtkToolButton *) gtk_tool_button_new
04140         (gtk_image_new_from_icon_name ("help-browser",
04141             GTK_ICON_SIZE_LARGE_TOOLBAR),
04142         gettext ("Help"));
04143     g_signal_connect (window->button_help, "clicked", window_help, NULL);
04144
04145     // Creating the about button
04146     window->button_about = (GtkToolButton *) gtk_tool_button_new
04147         (gtk_image_new_from_icon_name ("help-about",
04148             GTK_ICON_SIZE_LARGE_TOOLBAR),
04149         gettext ("About"));
04150     g_signal_connect (window->button_about, "clicked", window_about, NULL);
04151
04152     // Creating the exit button
04153     window->button_exit = (GtkToolButton *) gtk_tool_button_new
04154         (gtk_image_new_from_icon_name ("application-exit",
04155             GTK_ICON_SIZE_LARGE_TOOLBAR),
04156         gettext ("Exit"));
04157     g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
04158
04159     // Creating the buttons bar
04160     window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04161     gtk_toolbar_insert
04162         (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
04163     gtk_toolbar_insert
04164         (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
04165     gtk_toolbar_insert
04166         (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
04167     gtk_toolbar_insert
04168         (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
04169     gtk_toolbar_insert
04170         (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
04171     gtk_toolbar_insert
04172         (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
04173     gtk_toolbar_insert
04174         (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
04175     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04176
04177     // Creating the simulator program label and entry
04178     window->label_simulator
04179         = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04180     window->button_simulator = (GtkFileChooserButton *)
04181         gtk_file_chooser_button_new (gettext ("Simulator program"),
04182             GTK_FILE_CHOOSER_ACTION_OPEN);
04183     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04184         gettext ("Simulator program executable file"));
04185
04186     // Creating the evaluator program label and entry
04187     window->check_evaluator = (GtkCheckButton *)
04188         gtk_check_button_new_with_mnemonic (gettext ("Evaluator program"));
04189     g_signal_connect (window->check_evaluator, "toggled",
04190         window_update, NULL);
04191     window->button_evaluator = (GtkFileChooserButton *)
04192         gtk_file_chooser_button_new (gettext ("Evaluator program"),
04193             GTK_FILE_CHOOSER_ACTION_OPEN);
04194     gtk_widget_set_tooltip_text
04195         (GTK_WIDGET (window->button_evaluator),
04196         gettext ("Optional evaluator program executable file"));
04197
04198     // Creating the results files labels and entries
04199     window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
04200     window->entry_result = (GtkEntry *) gtk_entry_new ();
04201     gtk_widget_set_tooltip_text
04202         (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
04203     window->label_variables
04204         = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
04205     window->entry_variables = (GtkEntry *) gtk_entry_new ();
04206     gtk_widget_set_tooltip_text
04207         (GTK_WIDGET (window->entry_variables),
04208         gettext ("All simulated results file"));
04209
04210     // Creating the files grid and attaching widgets
04211     window->grid_files = (GtkGrid *) gtk_grid_new ();
04212     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04213         label_simulator),
04214         0, 0, 1, 1);
04215     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04216         button_simulator),
04217         1, 0, 1, 1);
04218     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04219         check_evaluator),
04220         2, 0, 1, 1);
04221     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->

```

```

        button_evaluator),
04218         3, 0, 1, 1);
04219     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
04220         0, 1, 1, 1);
04221     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
04222         1, 1, 1, 1);
04223     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
04224         2, 1, 1, 1);
04225     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
04226         3, 1, 1, 1);
04227
04228     // Creating the algorithm properties
04229     window->label_simulations = (GtkLabel *) gtk_label_new
04230     (gettext ("Simulations number"));
04231     window->spin_simulations
04232     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04233     gtk_widget_set_tooltip_text
04234     (GTK_WIDGET (window->spin_simulations),
04235      gettext ("Number of simulations to perform for each iteration"));
04236     window->label_iterations = (GtkLabel *)
04237     gtk_label_new (gettext ("Iterations number"));
04238     window->spin_iterations
04239     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04240     gtk_widget_set_tooltip_text
04241     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04242     g_signal_connect
04243     (window->spin_iterations, "value-changed", window_update, NULL);
04244     window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04245     window->spin_tolerance
04246     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04247     gtk_widget_set_tooltip_text
04248     (GTK_WIDGET (window->spin_tolerance),
04249      gettext ("Tolerance to set the variable interval on the next iteration"));
04250     window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04251     window->spin_bests
04252     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04253     gtk_widget_set_tooltip_text
04254     (GTK_WIDGET (window->spin_bests),
04255      gettext ("Number of best simulations used to set the variable interval "
04256               "on the next iteration"));
04257     window->label_population
04258     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04259     window->spin_population
04260     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04261     gtk_widget_set_tooltip_text
04262     (GTK_WIDGET (window->spin_population),
04263      gettext ("Number of population for the genetic algorithm"));
04264     window->label_generations
04265     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04266     window->spin_generations
04267     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04268     gtk_widget_set_tooltip_text
04269     (GTK_WIDGET (window->spin_generations),
04270      gettext ("Number of generations for the genetic algorithm"));
04271     window->label_mutation
04272     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04273     window->spin_mutation
04274     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04275     gtk_widget_set_tooltip_text
04276     (GTK_WIDGET (window->spin_mutation),
04277      gettext ("Ratio of mutation for the genetic algorithm"));
04278     window->label_reproduction
04279     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04280     window->spin_reproduction
04281     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04282     gtk_widget_set_tooltip_text
04283     (GTK_WIDGET (window->spin_reproduction),
04284      gettext ("Ratio of reproduction for the genetic algorithm"));
04285     window->label_adaptation
04286     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04287     window->spin_adaptation
04288     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04289     gtk_widget_set_tooltip_text
04290     (GTK_WIDGET (window->spin_adaptation),
04291      gettext ("Ratio of adaptation for the genetic algorithm"));
04292
04293     // Creating the gradient based method properties
04294     window->check_gradient = (GtkCheckButton *)
04295     gtk_check_button_new_with_mnemonic (gettext ("Gradient based method"));
04296     g_signal_connect (window->check_gradient, "clicked",
window_update, NULL);
04297     window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04298     window->button_gradient[0] = (GtkRadioButton *)

```

```

04299     gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04300     gtk_grid_attach (window->grid_gradient,
04301                     GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04302     g_signal_connect (window->button_gradient[0], "clicked",
window_update, NULL);
04303     for (i = 0; ++i < NGRADIENTS;)
04304     {
04305         window->button_gradient[i] = (GtkRadioButton *)
04306             gtk_radio_button_new_with_mnemonic
04307             (gtk_radio_button_get_group (window->button_gradient[0]),
04308              label_gradient[i]);
04309         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04310                                     tip_gradient[i]);
04311         gtk_grid_attach (window->grid_gradient,
04312                         GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04313         g_signal_connect (window->button_gradient[i], "clicked",
04314                           window_update, NULL);
04315     }
04316     window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04317     window->spin_steps = (GtkSpinButton *)
04318         gtk_spin_button_new_with_range (1., 1.e12, 1.);
04319     window->label_estimates
04320         = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04321     window->spin_estimates = (GtkSpinButton *)
04322         gtk_spin_button_new_with_range (1., 1.e3, 1.);
04323     window->label_relaxation
04324         = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04325     window->spin_relaxation = (GtkSpinButton *)
04326         gtk_spin_button_new_with_range (0., 2., 0.001);
04327     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_steps),
04328                     0, NGRADIENTS, 1, 1);
04329     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_steps),
04330                     1, NGRADIENTS, 1, 1);
04331     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_estimates),
04332                     0, NGRADIENTS + 1, 1, 1);
04333     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_estimates),
04334                     1, NGRADIENTS + 1, 1, 1);
04335     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_relaxation),
04336                     0, NGRADIENTS + 2, 1, 1);
04337     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_relaxation),
04338                     1, NGRADIENTS + 2, 1, 1);
04339
04340     // Creating the array of algorithms
04341     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
04342     window->button_algorithm[0] = (GtkRadioButton *)
04343         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04344     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
04345                                 tip_algorithm[0]);
04346     gtk_grid_attach (window->grid_algorithm,
04347                     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
04348     g_signal_connect (window->button_algorithm[0], "clicked",
04349                       window_set_algorithm, NULL);
04350     for (i = 0; ++i < NALGORITHMS;)
04351     {
04352         window->button_algorithm[i] = (GtkRadioButton *)
04353             gtk_radio_button_new_with_mnemonic
04354             (gtk_radio_button_get_group (window->button_algorithm[0]),
04355              label_algorithm[i]);
04356         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
04357                                     tip_algorithm[i]);
04358         gtk_grid_attach (window->grid_algorithm,
04359                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
04360         g_signal_connect (window->button_algorithm[i], "clicked",
04361                           window_set_algorithm, NULL);
04362     }
04363     gtk_grid_attach (window->grid_algorithm,
04364                     GTK_WIDGET (window->label_simulations), 0,
04365                     NALGORITHMS, 1, 1);
04366     gtk_grid_attach (window->grid_algorithm,
04367                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
04368     gtk_grid_attach (window->grid_algorithm,
04369                     GTK_WIDGET (window->label_iterations), 0,
04370                     NALGORITHMS + 1, 1, 1);
04371     gtk_grid_attach (window->grid_algorithm,
04372                     GTK_WIDGET (window->spin_iterations), 1,
04373                     NALGORITHMS + 1, 1, 1);
04374     gtk_grid_attach (window->grid_algorithm,
04375                     GTK_WIDGET (window->label_tolerance), 0,
04376                     NALGORITHMS + 2, 1, 1);
04377     gtk_grid_attach (window->grid_algorithm,
04378                     GTK_WIDGET (window->spin_tolerance), 1,

```

```

04379         NALGORITHMS + 2, 1, 1);
04380     gtk_grid_attach (window->grid_algorithm,
04381         GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
04382     gtk_grid_attach (window->grid_algorithm,
04383         GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
04384     gtk_grid_attach (window->grid_algorithm,
04385         GTK_WIDGET (window->label_population), 0,
04386         NALGORITHMS + 4, 1, 1);
04387     gtk_grid_attach (window->grid_algorithm,
04388         GTK_WIDGET (window->spin_population), 1,
04389         NALGORITHMS + 4, 1, 1);
04390     gtk_grid_attach (window->grid_algorithm,
04391         GTK_WIDGET (window->label_generations), 0,
04392         NALGORITHMS + 5, 1, 1);
04393     gtk_grid_attach (window->grid_algorithm,
04394         GTK_WIDGET (window->spin_generations), 1,
04395         NALGORITHMS + 5, 1, 1);
04396     gtk_grid_attach (window->grid_algorithm,
04397         GTK_WIDGET (window->label_mutation), 0,
04398         NALGORITHMS + 6, 1, 1);
04399     gtk_grid_attach (window->grid_algorithm,
04400         GTK_WIDGET (window->spin_mutation), 1,
04401         NALGORITHMS + 6, 1, 1);
04402     gtk_grid_attach (window->grid_algorithm,
04403         GTK_WIDGET (window->label_reproduction), 0,
04404         NALGORITHMS + 7, 1, 1);
04405     gtk_grid_attach (window->grid_algorithm,
04406         GTK_WIDGET (window->spin_reproduction), 1,
04407         NALGORITHMS + 7, 1, 1);
04408     gtk_grid_attach (window->grid_algorithm,
04409         GTK_WIDGET (window->label_adaptation), 0,
04410         NALGORITHMS + 8, 1, 1);
04411     gtk_grid_attach (window->grid_algorithm,
04412         GTK_WIDGET (window->spin_adaptation), 1,
04413         NALGORITHMS + 8, 1, 1);
04414     gtk_grid_attach (window->grid_algorithm,
04415         GTK_WIDGET (window->check_gradient), 0,
04416         NALGORITHMS + 9, 2, 1);
04417     gtk_grid_attach (window->grid_algorithm,
04418         GTK_WIDGET (window->grid_gradient), 0,
04419         NALGORITHMS + 10, 2, 1);
04420     window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
04421     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04422         GTK_WIDGET (window->grid_algorithm));
04423
04424     // Creating the variable widgets
04425     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04426     gtk_widget_set_tooltip_text
04427         (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04428     window->id_variable = g_signal_connect
04429         (window->combo_variable, "changed", window_set_variable, NULL);
04430     window->button_add_variable
04431         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04432             GTK_ICON_SIZE_BUTTON);
04433     g_signal_connect
04434         (window->button_add_variable, "clicked",
04435         window_add_variable, NULL);
04436     gtk_widget_set_tooltip_text
04437         (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
04438     window->button_remove_variable
04439         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04440             GTK_ICON_SIZE_BUTTON);
04441     g_signal_connect
04442         (window->button_remove_variable, "clicked",
04443         window_remove_variable, NULL);
04444     gtk_widget_set_tooltip_text
04445         (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04446     window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04447     window->entry_variable = (GtkEntry *) gtk_entry_new ();
04448     gtk_widget_set_tooltip_text
04449         (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04450     window->id_variable_label = g_signal_connect
04451         (window->entry_variable, "changed", window_label_variable, NULL);
04452     window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04453     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04454         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04455     gtk_widget_set_tooltip_text
04456         (GTK_WIDGET (window->spin_min),
04457         gettext ("Minimum initial value of the variable"));
04458     viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04459     gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
04460     window->scrolled_min
04461         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04462     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04463         GTK_WIDGET (viewport));
04464     g_signal_connect (window->spin_min, "value-changed",
04465         window_rangemin_variable, NULL);

```

```

04464 window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04465 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04466     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04467 gtk_widget_set_tooltip_text
04468     (GTK_WIDGET (window->spin_max),
04469      gettext ("Maximum initial value of the variable"));
04470 viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04471 gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
04472 window->scrolled_max
04473     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04474 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04475     GTK_WIDGET (viewport));
04476 g_signal_connect (window->spin_max, "value-changed",
04477     window_rangemax_variable, NULL);
04478 window->check_minabs = (GtkCheckButton *)
04479     gtk_check_button_new_with_mnemonic (gettext ("Absolute minimum"));
04480 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04481 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04482     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04483 gtk_widget_set_tooltip_text
04484     (GTK_WIDGET (window->spin_minabs),
04485      gettext ("Minimum allowed value of the variable"));
04486 viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04487 gtk_container_add (GTK_CONTAINER (viewport),
04488     GTK_WIDGET (window->spin_minabs));
04489 window->scrolled_minabs
04490     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04491 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04492     GTK_WIDGET (viewport));
04493 g_signal_connect (window->spin_minabs, "value-changed",
04494     window_rangeminabs_variable, NULL);
04495 window->check_maxabs = (GtkCheckButton *)
04496     gtk_check_button_new_with_mnemonic (gettext ("Absolute maximum"));
04497 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04498 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04499     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04500 gtk_widget_set_tooltip_text
04501     (GTK_WIDGET (window->spin_maxabs),
04502      gettext ("Maximum allowed value of the variable"));
04503 viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04504 gtk_container_add (GTK_CONTAINER (viewport),
04505     GTK_WIDGET (window->spin_maxabs));
04506 window->scrolled_maxabs
04507     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04508 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04509     GTK_WIDGET (viewport));
04510 g_signal_connect (window->spin_maxabs, "value-changed",
04511     window_rangemaxabs_variable, NULL);
04512 window->label_precision
04513     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04514 window->spin_precision = (GtkSpinButton *)
04515     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04516 gtk_widget_set_tooltip_text
04517     (GTK_WIDGET (window->spin_precision),
04518      gettext ("Number of precision floating point digits\n"
04519       "0 is for integer numbers"));
04520 g_signal_connect (window->spin_precision, "value-changed",
04521     window_precision_variable, NULL);
04522 window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
04523 window->spin_sweeps
04524     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04525 gtk_widget_set_tooltip_text
04526     (GTK_WIDGET (window->spin_sweeps),
04527      gettext ("Number of steps sweeping the variable"));
04528 g_signal_connect
04529     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
04530 window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
04531 window->spin_bits
04532     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
04533 gtk_widget_set_tooltip_text
04534     (GTK_WIDGET (window->spin_bits),
04535      gettext ("Number of bits to encode the variable"));
04536 g_signal_connect
04537     (window->spin_bits, "value-changed", window_update_variable, NULL);
04538 window->grid_variable = (GtkGrid *) gtk_grid_new ();
04539 gtk_grid_attach (window->grid_variable,
04540     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
04541 gtk_grid_attach (window->grid_variable,
04542     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
04543 gtk_grid_attach (window->grid_variable,
04544     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
04545 gtk_grid_attach (window->grid_variable,
04546     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
04547 gtk_grid_attach (window->grid_variable,
04548     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
04549 gtk_grid_attach (window->grid_variable,
04550     GTK_WIDGET (window->label_min), 0, 2, 1, 1);

```

```

04551 gtk_grid_attach (window->grid_variable,
04552 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04553 gtk_grid_attach (window->grid_variable,
04554 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04555 gtk_grid_attach (window->grid_variable,
04556 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04557 gtk_grid_attach (window->grid_variable,
04558 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
04559 gtk_grid_attach (window->grid_variable,
04560 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
04561 gtk_grid_attach (window->grid_variable,
04562 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04563 gtk_grid_attach (window->grid_variable,
04564 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
04565 gtk_grid_attach (window->grid_variable,
04566 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
04567 gtk_grid_attach (window->grid_variable,
04568 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
04569 gtk_grid_attach (window->grid_variable,
04570 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04571 gtk_grid_attach (window->grid_variable,
04572 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04573 gtk_grid_attach (window->grid_variable,
04574 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04575 gtk_grid_attach (window->grid_variable,
04576 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04577 window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
04578 gtk_container_add (GTK_CONTAINER (window->frame_variable),
04579 GTK_WIDGET (window->grid_variable));
04580
04581 // Creating the experiment widgets
04582 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
04583 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
04584 gettext ("Experiment selector"));
04585 window->id_experiment = g_signal_connect
04586 (window->combo_experiment, "changed", window_set_experiment, NULL)
;
04587 window->button_add_experiment
04588 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04589 GTK_ICON_SIZE_BUTTON);
04590 g_signal_connect
04591 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
04592 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04593 gettext ("Add experiment"));
04594 window->button_remove_experiment
04595 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04596 GTK_ICON_SIZE_BUTTON);
04597 g_signal_connect (window->button_remove_experiment, "clicked",
04598 window_remove_experiment, NULL);
04599 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04600 gettext ("Remove experiment"));
04601 window->label_experiment
04602 = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04603 window->button_experiment = (GtkFileChooserButton *)
04604 gtk_file_chooser_button_new (gettext ("Experimental data file"),
04605 GTK_FILE_CHOOSER_ACTION_OPEN);
04606 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04607 gettext ("Experimental data file"));
04608 window->id_experiment_name
04609 = g_signal_connect (window->button_experiment, "selection-changed",
04610 window_name_experiment, NULL);
04611 window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
04612 window->spin_weight
04613 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04614 gtk_widget_set_tooltip_text
04615 (GTK_WIDGET (window->spin_weight),
04616 gettext ("Weight factor to build the objective function"));
04617 g_signal_connect
04618 (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
04619 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04620 gtk_grid_attach (window->grid_experiment,
04621 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
04622 gtk_grid_attach (window->grid_experiment,
04623 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
04624 gtk_grid_attach (window->grid_experiment,
04625 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
04626 gtk_grid_attach (window->grid_experiment,
04627 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
04628 gtk_grid_attach (window->grid_experiment,
04629 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
04630 gtk_grid_attach (window->grid_experiment,
04631 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04632 gtk_grid_attach (window->grid_experiment,
04633 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04634 for (i = 0; i < MAX_NINPUTS; ++i)

```



```

04635     {
04636         snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04637         window->check_template[i] = (GtkCheckButton *)
04638             gtk_check_button_new_with_label (buffer3);
04639         window->id_template[i]
04640             = g_signal_connect (window->check_template[i], "toggled",
04641                                 window_inputs_experiment, NULL);
04642         gtk_grid_attach (window->grid_experiment,
04643                         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
04644         window->button_template[i] = (GtkFileChooserButton *)
04645             gtk_file_chooser_button_new (gettext ("Input template"),
04646                                         GTK_FILE_CHOOSER_ACTION_OPEN);
04647         gtk_widget_set_tooltip_text
04648             (GTK_WIDGET (window->button_template[i]),
04649              gettext ("Experimental input template file"));
04650         window->id_input[i]
04651             = g_signal_connect_swapped (window->button_template[i],
04652                                         "selection-changed",
04653                                         (void (*)(void *)) window_template_experiment,
04654                                         (void *) (size_t) i);
04655         gtk_grid_attach (window->grid_experiment,
04656                         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
04657     }
04658     window->frame_experiment
04659         = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04660     gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04661                       GTK_WIDGET (window->grid_experiment));
04662
04663     // Creating the grid and attaching the widgets to the grid
04664     window->grid = (GtkGrid *) gtk_grid_new ();
04665     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04666     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04667     gtk_grid_attach (window->grid,
04668                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04669     gtk_grid_attach (window->grid,
04670                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04671     gtk_grid_attach (window->grid,
04672                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04673     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
04674 grid));
04675
04676     // Setting the window logo
04677     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04678     gtk_window_set_icon (window->window, window->logo);
04679
04680     // Showing the window
04681     gtk_widget_show_all (GTK_WIDGET (window->window));
04682
04683     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04684     #if GTK_MINOR_VERSION >= 16
04685     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04686     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04687     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04688     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04689     #endif
04690
04691     // Reading initial example
04692     input_new ();
04693     buffer2 = g_get_current_dir ();
04694     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04695     g_free (buffer2);
04696     window_read (buffer);
04697     g_free (buffer);
04698 }
04699 #endif
04700
04701 int
04702 cores_number ()
04703 {
04704     #ifdef G_OS_WIN32
04705     SYSTEM_INFO sysinfo;
04706     GetSystemInfo (&sysinfo);
04707     return sysinfo.dwNumberOfProcessors;
04708     #else
04709     return (int) sysconf (_SC_NPROCESSORS_ONLN);
04710     #endif
04711 }
04712
04713 int
04714 main (int argc, char **argv)
04715 {
04716     #if HAVE_GTK
04717     char *buffer;
04718     #endif
04719
04720     // Starting pseudo-random numbers generator

```

```

04735     calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04736     calibrate->seed = DEFAULT_RANDOM_SEED;
04737
04738     // Allowing spaces in the XML data file
04739     xmlKeepBlanksDefault (0);
04740
04741     // Starting MPI
04742     #if HAVE_MPI
04743         MPI_Init (&argn, &argc);
04744         MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04745         MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04746         printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04747     #else
04748         ntasks = 1;
04749     #endif
04750
04751     #if HAVE_GTK
04752
04753         // Getting threads number
04754         nthreads_gradient = nthreads = cores_number ();
04755
04756         // Setting local language and international floating point numbers notation
04757         setlocale (LC_ALL, "");
04758         setlocale (LC_NUMERIC, "C");
04759         window->application_directory = g_get_current_dir ();
04760         buffer = g_build_filename (window->application_directory,
04761             LOCALE_DIR, NULL);
04761         bindtextdomain (PROGRAM_INTERFACE, buffer);
04762         bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04763         textdomain (PROGRAM_INTERFACE);
04764
04765         // Initing GTK+
04766         gtk_disable_setlocale ();
04767         gtk_init (&argn, &argc);
04768
04769         // Opening the main window
04770         window_new ();
04771         gtk_main ();
04772
04773         // Freeing memory
04774         input_free ();
04775         g_free (buffer);
04776         gtk_widget_destroy (GTK_WIDGET (window->window));
04777         g_free (window->application_directory);
04778     #else
04779
04780         // Checking syntax
04781         if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04782         {
04783             printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04784             return 1;
04785         }
04786
04787         // Getting threads number
04788         if (argn == 2)
04789             nthreads_gradient = nthreads = cores_number ();
04790         else
04791         {
04792             nthreads_gradient = nthreads = atoi (argc[2]);
04793             if (!nthreads)
04794             {
04795                 printf ("Bad threads number\n");
04796                 return 2;
04797             }
04798         }
04799         printf ("nthreads=%u\n", nthreads);
04800
04801         // Making calibration
04802         if (input_open (argc[argn - 1]))
04803             calibrate_open ();
04804
04805         // Freeing memory
04806         calibrate_free ();
04807     #endif
04808
04809     // Closing MPI
04810     #if HAVE_MPI
04811         MPI_Finalize ();
04812     #endif
04813
04814     // Freeing memory
04815     gsl_rng_free (calibrate->rng);
04816
04817     // Closing
04818     return 0;

```

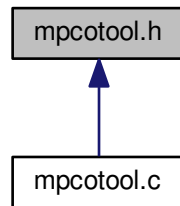


```
04821 }
```

## 5.7 mpcotool.h File Reference

Header file of the mpcotool.

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



### Data Structures

- struct [Input](#)  
*Struct to define the calibration input file.*
- struct [Calibrate](#)  
*Struct to define the calibration data.*
- struct [ParallelData](#)  
*Struct to pass to the GThreads parallelized function.*

### Enumerations

- enum [Algorithm](#) { [ALGORITHM\\_MONTE\\_CARLO](#) = 0, [ALGORITHM\\_SWEEP](#) = 1, [ALGORITHM\\_GENETIC](#) = 2 }
  - enum [GradientMethod](#) { [GRADIENT\\_METHOD\\_COORDINATES](#) = 0, [GRADIENT\\_METHOD\\_RANDOM](#) = 1 }
- Enum to define the algorithms.*
- Enum to define the methods to estimate the gradient.*

### Functions

- void [show\\_message](#) (char \*title, char \*msg, int type)  
*Function to show a dialog with a message.*
- void [show\\_error](#) (char \*msg)  
*Function to show a dialog with an error message.*
- int [xml\\_node\\_get\\_int](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an integer number of a XML node property.*
- unsigned int [xml\\_node\\_get\\_uint](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an unsigned integer number of a XML node property.*
- 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.*

  - 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.*

  - void `input_new` ()
- Function to create a new Input struct.*

  - void `input_free` ()
- Function to free the memory of the input file data.*

  - int `input_open` (char \*filename)
- Function to open the input file.*

  - void `calibrate_input` (unsigned int simulation, char \*input, GMappedFile \*template)
- Function to write the simulation input file.*

  - double `calibrate_parse` (unsigned int simulation, unsigned int experiment)
- Function to parse input files, simulating and calculating the \ objective function.*

  - void `calibrate_print` ()
- Function to print the results.*

  - void `calibrate_save_variables` (unsigned int simulation, double error)
- Function to save in a file the variables and the error.*

  - void `calibrate_best` (unsigned int simulation, double value)
- Function to save the best simulations.*

  - void `calibrate_sequential` ()
- Function to calibrate sequentially.*

  - void \* `calibrate_thread` (ParallelData \*data)
- Function to calibrate on a thread.*

  - void `calibrate_merge` (unsigned int nsaveds, unsigned int \*simulation\_best, double \*error\_best)
- Function to merge the 2 calibration results.*

  - void `calibrate_synchronise` ()
- Function to synchronise the calibration results of MPI tasks.*

  - void `calibrate_sweep` ()
- Function to calibrate with the sweep algorithm.*

  - void `calibrate_MonteCarlo` ()
- Function to calibrate with the Monte-Carlo algorithm.*

  - void `calibrate_best_gradient` (unsigned int simulation, double value)
- Function to save the best simulation in a gradient based method.*

  - void `calibrate_gradient_sequential` ()
- Function to estimate the gradient on a thread.*

  - void \* `calibrate_gradient_thread` (ParallelData \*data)
- Function to estimate the gradient on a thread.*

  - double `calibrate_variable_step_gradient` (unsigned int variable)
- Function to do a step of the gradient based method.*

  - void `calibrate_step_gradient` (unsigned int simulation)
- Function to calibrate with a gradient based method.*

  - void `calibrate_gradient` ()
- Function to calculate the objective function of an entity.*

  - double `calibrate_genetic_objective` (Entity \*entity)
- Function to calibrate with the genetic algorithm.*

  - void `calibrate_genetic` ()
- Function to save the best results on iterative methods.*

  - void `calibrate_save_old` ()

- void [calibrate\\_merge\\_old](#) ()  
*Function to merge the best results with the previous step best results on iterative methods.*
- void [calibrate\\_refine](#) ()  
*Function to refine the search ranges of the variables in iterative algorithms.*
- void [calibrate\\_step](#) ()  
*Function to do a step of the iterative algorithm.*
- void [calibrate\\_iterate](#) ()  
*Function to iterate the algorithm.*
- void [calibrate\\_open](#) ()  
*Function to open and perform a calibration.*

### 5.7.1 Detailed Description

Header file of the mpcotool.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2015, all rights reserved.

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

### 5.7.2 Enumeration Type Documentation

#### 5.7.2.1 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 43 of file [mpcotool.h](#).

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

#### 5.7.2.2 enum GradientMethod

Enum to define the methods to estimate the gradient.

##### Enumerator

**GRADIENT\_METHOD\_COORDINATES** Coordinates descent method.  
**GRADIENT\_METHOD\_RANDOM** Random method.

Definition at line 54 of file [mpcotool.h](#).

```
00055 {
00056     GRADIENT_METHOD_COORDINATES = 0,
00057     GRADIENT_METHOD_RANDOM = 1,
00058 };
```

### 5.7.3 Function Documentation

#### 5.7.3.1 void `calibrate_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 1420 of file [mpcotool.c](#).

```

01421 {
01422     unsigned int i, j;
01423     double e;
01424     #if DEBUG
01425         fprintf (stderr, "calibrate_best: start\n");
01426         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01427                 calibrate->nsaveds, calibrate->nbest);
01428     #endif
01429     if (calibrate->nsaveds < calibrate->nbest
01430         || value < calibrate->error_best[calibrate->nsaveds - 1])
01431     {
01432         if (calibrate->nsaveds < calibrate->nbest)
01433             ++calibrate->nsaveds;
01434         calibrate->error_best[calibrate->nsaveds - 1] = value;
01435         calibrate->simulation_best[calibrate->
01436             nsaveds - 1] = simulation;
01437         for (i = calibrate->nsaveds; --i;)
01438         {
01439             if (calibrate->error_best[i] < calibrate->
01440                 error_best[i - 1])
01441             {
01442                 j = calibrate->simulation_best[i];
01443                 e = calibrate->error_best[i];
01444                 calibrate->simulation_best[i] = calibrate->
01445                     simulation_best[i - 1];
01446                 calibrate->error_best[i] = calibrate->
01447                     error_best[i - 1];
01448                 calibrate->simulation_best[i - 1] = j;
01449                 calibrate->error_best[i - 1] = e;
01450             }
01451             else
01452                 break;
01453         }
01454     }
01455     #if DEBUG
01456         fprintf (stderr, "calibrate_best: end\n");
01457     #endif
01458 }

```

#### 5.7.3.2 void `calibrate_best_gradient` ( unsigned int *simulation*, double *value* )

Function to save the best simulation in a gradient based method.

##### Parameters

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

Definition at line 1733 of file [mpcotool.c](#).

```

01734 {
01735     #if DEBUG
01736         fprintf (stderr, "calibrate_best_gradient: start\n");
01737         fprintf (stderr,
01738                 "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01739                 simulation, value, calibrate->error_best[0]);
01740     #endif
01741     if (value < calibrate->error_best[0])
01742     {
01743         calibrate->error_best[0] = value;
01744         calibrate->simulation_best[0] = simulation;
01745     }
01746     #if DEBUG
01747         fprintf (stderr,

```

```

01747             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01748             simulation, value);
01749 #endif
01750     }
01751 #if DEBUG
01752     fprintf (stderr, "calibrate_best_gradient: end\n");
01753 #endif
01754 }

```

### 5.7.3.3 double calibrate\_genetic\_objective ( Entity \* entity )

Function to calculate the objective function of an entity.

#### Parameters

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

#### Returns

objective function value.

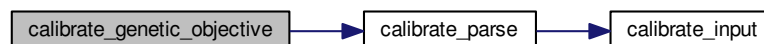
Definition at line 2036 of file [mpcotool.c](#).

```

02037 {
02038     unsigned int j;
02039     double objective;
02040     char buffer[64];
02041 #if DEBUG
02042     fprintf (stderr, "calibrate_genetic_objective: start\n");
02043 #endif
02044     for (j = 0; j < calibrate->nvariables; ++j)
02045     {
02046         calibrate->value[entity->id * calibrate->nvariables + j]
02047             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02048     }
02049     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02050         objective += calibrate_parse (entity->id, j);
02051     g_mutex_lock (mutex);
02052     for (j = 0; j < calibrate->nvariables; ++j)
02053     {
02054         snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02055         fprintf (calibrate->file_variables, buffer,
02056             genetic_get_variable (entity, calibrate->
02057                 genetic_variable + j));
02058     }
02059     fprintf (calibrate->file_variables, "%.14le\n", objective);
02060     g_mutex_unlock (mutex);
02061 #if DEBUG
02062     fprintf (stderr, "calibrate_genetic_objective: end\n");
02063 #endif
02064     return objective;
02065 }

```

Here is the call graph for this function:



### 5.7.3.4 void\* calibrate\_gradient\_thread ( ParallelData \* data )

Function to estimate the gradient on a thread.

## Parameters

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

## Returns

NULL

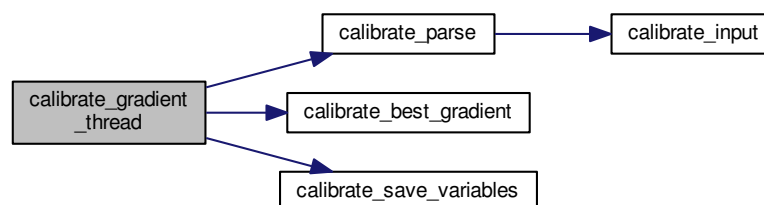
Definition at line 1798 of file [mpcotool.c](#).

```

01799 {
01800     unsigned int i, j, thread;
01801     double e;
01802     #if DEBUG
01803     fprintf (stderr, "calibrate_gradient_thread: start\n");
01804     #endif
01805     thread = data->thread;
01806     #if DEBUG
01807     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01808             thread,
01809             calibrate->thread_gradient[thread],
01810             calibrate->thread_gradient[thread + 1]);
01811     #endif
01812     for (i = calibrate->thread_gradient[thread];
01813          i < calibrate->thread_gradient[thread + 1]; ++i)
01814     {
01815         e = 0.;
01816         for (j = 0; j < calibrate->nexperiments; ++j)
01817             e += calibrate_parse (i, j);
01818         g_mutex_lock (mutex);
01819         calibrate_best_gradient (i, e);
01820         calibrate_save_variables (i, e);
01821         g_mutex_unlock (mutex);
01822     #if DEBUG
01823     fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01824     #endif
01825     }
01826     #if DEBUG
01827     fprintf (stderr, "calibrate_gradient_thread: end\n");
01828     #endif
01829     g_thread_exit (NULL);
01830     return NULL;
01831 }

```

Here is the call graph for this function:



### 5.7.3.5 void calibrate\_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 1173 of file [mpcotool.c](#).

```

01174 {
01175     unsigned int i;
01176     char buffer[32], value[32], *buffer2, *buffer3, *content;
01177     FILE *file;
01178     gsize length;
01179     GRegex *regex;
01180
01181     #if DEBUG
01182         fprintf (stderr, "calibrate_input: start\n");
01183     #endif
01184
01185     // Checking the file
01186     if (!template)
01187         goto calibrate_input_end;
01188
01189     // Opening template
01190     content = g_mapped_file_get_contents (template);
01191     length = g_mapped_file_get_length (template);
01192     #if DEBUG
01193         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01194                 content);
01195     #endif
01196     file = g_fopen (input, "w");
01197
01198     // Parsing template
01199     for (i = 0; i < calibrate->nvariables; ++i)
01200     {
01201         #if DEBUG
01202             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01203         #endif
01204         snprintf (buffer, 32, "@variable%u@", i + 1);
01205         regex = g_regex_new (buffer, 0, 0, NULL);
01206         if (i == 0)
01207         {
01208             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01209                                                calibrate->label[i], 0, NULL);
01210         #if DEBUG
01211             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01212         #endif
01213         }
01214         else
01215         {
01216             length = strlen (buffer3);
01217             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01218                                                calibrate->label[i], 0, NULL);
01219             g_free (buffer3);
01220         }
01221         g_regex_unref (regex);
01222         length = strlen (buffer2);
01223         snprintf (buffer, 32, "@value%u@", i + 1);
01224         regex = g_regex_new (buffer, 0, 0, NULL);
01225         snprintf (value, 32, format[calibrate->precision[i]],
01226                  calibrate->value[simulation * calibrate->
01227                                nvariables + i]);
01228         #if DEBUG
01229             fprintf (stderr, "calibrate_input: value=%s\n", value);
01230         #endif
01231         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01232                                           0, NULL);
01233         g_free (buffer2);
01234         g_regex_unref (regex);
01235     }
01236
01237     // Saving input file
01238     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01239     g_free (buffer3);
01240     fclose (file);
01241
01242     calibrate_input_end:
01243     #if DEBUG
01244         fprintf (stderr, "calibrate_input: end\n");
01245     #endif
01246     return;
01247 }

```

### 5.7.3.6 void `calibrate_merge` ( unsigned int *nsaveds*, unsigned int \* *simulation\_best*, double \* *error\_best* )

Function to merge the 2 calibration 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 1538 of file `mpcotool.c`.

```

01540 {
01541     unsigned int i, j, k, s[calibrate->nbest];
01542     double e[calibrate->nbest];
01543     #if DEBUG
01544     fprintf (stderr, "calibrate_merge: start\n");
01545     #endif
01546     i = j = k = 0;
01547     do
01548     {
01549         if (i == calibrate->nsaveds)
01550         {
01551             s[k] = simulation_best[j];
01552             e[k] = error_best[j];
01553             ++j;
01554             ++k;
01555             if (j == nsaveds)
01556                 break;
01557         }
01558         else if (j == nsaveds)
01559         {
01560             s[k] = calibrate->simulation_best[i];
01561             e[k] = calibrate->error_best[i];
01562             ++i;
01563             ++k;
01564             if (i == calibrate->nsaveds)
01565                 break;
01566         }
01567         else if (calibrate->error_best[i] > error_best[j])
01568         {
01569             s[k] = simulation_best[j];
01570             e[k] = error_best[j];
01571             ++j;
01572             ++k;
01573         }
01574         else
01575         {
01576             s[k] = calibrate->simulation_best[i];
01577             e[k] = calibrate->error_best[i];
01578             ++i;
01579             ++k;
01580         }
01581     }
01582     while (k < calibrate->nbest);
01583     calibrate->nsaveds = k;
01584     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01585     memcpy (calibrate->error_best, e, k * sizeof (double));
01586     #if DEBUG
01587     fprintf (stderr, "calibrate_merge: end\n");
01588     #endif
01589 }

```

### 5.7.3.7 double `calibrate_parse` ( unsigned int *simulation*, unsigned int *experiment* )

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

#### Parameters

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



## Returns

Objective function value.

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

```

01261 {
01262     unsigned int i;
01263     double e;
01264     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01265         *buffer3, *buffer4;
01266     FILE *file_result;
01267
01268     #if DEBUG
01269         fprintf (stderr, "calibrate_parse: start\n");
01270         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01271             experiment);
01272     #endif
01273
01274     // Opening input files
01275     for (i = 0; i < calibrate->ninputs; ++i)
01276     {
01277         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01278     #if DEBUG
01279         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01280     #endif
01281         calibrate_input (simulation, &input[i][0],
01282             calibrate->file[i][experiment]);
01283     }
01284     for (; i < MAX_NINPUTS; ++i)
01285         strcpy (&input[i][0], "");
01286     #if DEBUG
01287         fprintf (stderr, "calibrate_parse: parsing end\n");
01288     #endif
01289
01290     // Performing the simulation
01291     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01292     buffer2 = g_path_get_dirname (calibrate->simulator);
01293     buffer3 = g_path_get_basename (calibrate->simulator);
01294     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01295     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
01296         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01297         input[6], input[7], output);
01298     g_free (buffer4);
01299     g_free (buffer3);
01300     g_free (buffer2);
01301     #if DEBUG
01302         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01303     #endif
01304     system (buffer);
01305
01306     // Checking the objective value function
01307     if (calibrate->evaluator)
01308     {
01309         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01310         buffer2 = g_path_get_dirname (calibrate->evaluator);
01311         buffer3 = g_path_get_basename (calibrate->evaluator);
01312         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01313         snprintf (buffer, 512, "%s\ " %s %s %s",
01314             buffer4, output, calibrate->experiment[experiment], result);
01315         g_free (buffer4);
01316         g_free (buffer3);
01317         g_free (buffer2);
01318     #if DEBUG
01319         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01320     #endif
01321         system (buffer);
01322         file_result = g_fopen (result, "r");
01323         e = atof (fgets (buffer, 512, file_result));
01324         fclose (file_result);
01325     }
01326     else
01327     {
01328         strcpy (result, "");
01329         file_result = g_fopen (output, "r");
01330         e = atof (fgets (buffer, 512, file_result));
01331         fclose (file_result);
01332     }
01333
01334     // Removing files
01335     #if !DEBUG
01336     for (i = 0; i < calibrate->ninputs; ++i)
01337     {
01338         if (calibrate->file[i][0])
01339         {

```

```

01340         snprintf (buffer, 512, RM " %s", &input[i][0]);
01341         system (buffer);
01342     }
01343 }
01344     snprintf (buffer, 512, RM " %s %s", output, result);
01345     system (buffer);
01346 #endif
01347
01348 #if DEBUG
01349     fprintf (stderr, "calibrate_parse: end\n");
01350 #endif
01351
01352 // Returning the objective function
01353 return e * calibrate->weight[experiment];
01354 }

```

Here is the call graph for this function:



#### 5.7.3.8 void calibrate\_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 1392 of file [mpcotool.c](#).

```

01393 {
01394     unsigned int i;
01395     char buffer[64];
01396     #if DEBUG
01397         fprintf (stderr, "calibrate_save_variables: start\n");
01398     #endif
01399     for (i = 0; i < calibrate->nvariables; ++i)
01400     {
01401         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01402         fprintf (calibrate->file_variables, buffer,
01403                 calibrate->value[simulation * calibrate->
01404                             nvariables + i]);
01405     }
01406     fprintf (calibrate->file_variables, "%.14le\n", error);
01407     #if DEBUG
01408         fprintf (stderr, "calibrate_save_variables: end\n");
01409     #endif
01409 }

```

#### 5.7.3.9 void calibrate\_step\_gradient ( unsigned int *simulation* )

Function to do a step of the gradient based method.

## Parameters

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

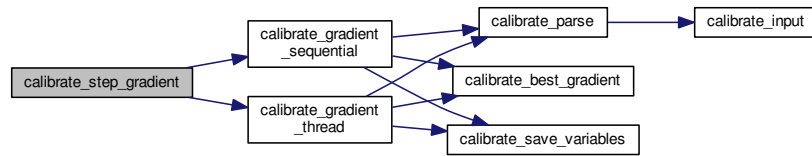
Definition at line 1900 of file `mpcotool.c`.

```

01901 {
01902     GThread *thread[nthreads_gradient];
01903     ParallelData data[nthreads_gradient];
01904     unsigned int i, j, k, b;
01905     #if DEBUG
01906         fprintf (stderr, "calibrate_step_gradient: start\n");
01907     #endif
01908     for (i = 0; i < calibrate->nestimates; ++i)
01909     {
01910         k = (simulation + i) * calibrate->nvariables;
01911         b = calibrate->simulation_best[0] * calibrate->
nvariables;
01912     #if DEBUG
01913         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01914                 simulation + i, calibrate->simulation_best[0]);
01915     #endif
01916         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01917         {
01918             #if DEBUG
01919                 fprintf (stderr,
01920                         "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01921                         i, j, calibrate->value[b]);
01922             #endif
01923             calibrate->value[k]
01924                 = calibrate->value[b] + calibrate_estimate_gradient (j
, i);
01925             calibrate->value[k] = fmin (fmax (calibrate->
value[k],
01926                                             calibrate->rangeminabs[j]),
01927                                         calibrate->rangemaxabs[j]);
01928             #if DEBUG
01929                 fprintf (stderr,
01930                         "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01931                         i, j, calibrate->value[k]);
01932             #endif
01933         }
01934     }
01935     if (nthreads_gradient == 1)
01936         calibrate_gradient_sequential (simulation);
01937     else
01938     {
01939         for (i = 0; i <= nthreads_gradient; ++i)
01940         {
01941             calibrate->thread_gradient[i]
01942                 = simulation + calibrate->nstart_gradient
01943                 + i * (calibrate->nend_gradient - calibrate->
nstart_gradient)
01944                 / nthreads_gradient;
01945             #if DEBUG
01946                 fprintf (stderr,
01947                         "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01948                         i, calibrate->thread_gradient[i]);
01949             #endif
01950         }
01951         for (i = 0; i < nthreads_gradient; ++i)
01952         {
01953             data[i].thread = i;
01954             thread[i] = g_thread_new
01955                 (NULL, (void (*) ) calibrate_gradient_thread, &data[i]);
01956         }
01957         for (i = 0; i < nthreads_gradient; ++i)
01958             g_thread_join (thread[i]);
01959     }
01960     #if DEBUG
01961         fprintf (stderr, "calibrate_step_gradient: end\n");
01962     #endif
01963 }

```

Here is the call graph for this function:



#### 5.7.3.10 void\* calibrate\_thread ( ParallelData \* data )

Function to calibrate on a thread.

##### Parameters

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

##### Returns

NULL

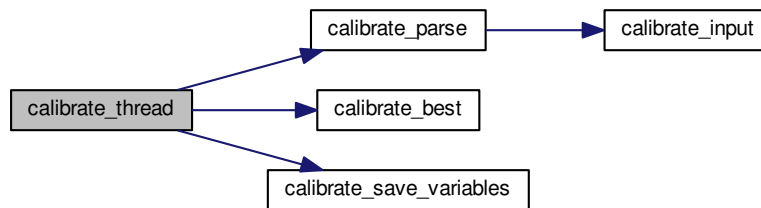
Definition at line 1494 of file [mpcotool.c](#).

```

01495 {
01496     unsigned int i, j, thread;
01497     double e;
01498     #if DEBUG
01499         fprintf (stderr, "calibrate_thread: start\n");
01500     #endif
01501     thread = data->thread;
01502     #if DEBUG
01503         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01504                 calibrate->thread[thread], calibrate->thread[thread + 1]);
01505     #endif
01506     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01507     {
01508         e = 0.;
01509         for (j = 0; j < calibrate->nexperiments; ++j)
01510             e += calibrate_parse (i, j);
01511         g_mutex_lock (mutex);
01512         calibrate_best (i, e);
01513         calibrate_save_variables (i, e);
01514         g_mutex_unlock (mutex);
01515     #if DEBUG
01516         fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01517     #endif
01518     }
01519     #if DEBUG
01520         fprintf (stderr, "calibrate_thread: end\n");
01521     #endif
01522     g_thread_exit (NULL);
01523     return NULL;
01524 }

```

Here is the call graph for this function:



### 5.7.3.11 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 488 of file `mpcotool.c`.

```

00489 {
00490     char buffer2[64];
00491     char *buffert[MAX_NINPUTS] =
00492         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493     xmlDoc *doc;
00494     xmlNode *node, *child;
00495     xmlChar *buffer;
00496     char *msg;
00497     int error_code;
00498     unsigned int i;
00499
00500     #if DEBUG
00501     fprintf (stderr, "input_open: start\n");
00502     #endif
00503
00504     // Resetting input data
00505     buffer = NULL;
00506     input_new ();
00507
00508     // Parsing the input file
00509     #if DEBUG
00510     fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00511     #endif
00512     doc = xmlParseFile (filename);
00513     if (!doc)
00514     {
00515         msg = gettext ("Unable to parse the input file");
00516         goto exit_on_error;
00517     }
00518
00519     // Getting the root node
00520     #if DEBUG
00521     fprintf (stderr, "input_open: getting the root node\n");
00522     #endif
00523     node = xmlDocGetRootElement (doc);
00524     if (xmlStrcmp (node->name, XML_CALIBRATE))
00525     {
00526         msg = gettext ("Bad root XML node");
00527         goto exit_on_error;
00528     }
00529

```

```

00530 // Getting results file names
00531 input->result = (char *) xmlGetProp (node, XML_RESULT);
00532 if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535 if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538 // Opening simulator program name
00539 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540 if (!input->simulator)
00541 {
00542     msg = gettext ("Bad simulator program");
00543     goto exit_on_error;
00544 }
00545
00546 // Opening evaluator program name
00547 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549 // Obtaining pseudo-random numbers generator seed
00550 if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552 else
00553 {
00554     input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555     if (error_code)
00556     {
00557         msg = gettext ("Bad pseudo-random numbers generator seed");
00558         goto exit_on_error;
00559     }
00560 }
00561
00562 // Opening algorithm
00563 buffer = xmlGetProp (node, XML_ALGORITHM);
00564 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565 {
00566     input->algorithm = ALGORITHM_MONTE_CARLO;
00567
00568     // Obtaining simulations number
00569     input->nsimulations
00570     = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571     if (error_code)
00572     {
00573         msg = gettext ("Bad simulations number");
00574         goto exit_on_error;
00575     }
00576 }
00577 else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     input->algorithm = ALGORITHM_SWEEP;
00579 else if (!xmlStrcmp (buffer, XML_GENETIC))
00580 {
00581     input->algorithm = ALGORITHM_GENETIC;
00582
00583     // Obtaining population
00584     if (xmlHasProp (node, XML_NPOPULATION))
00585     {
00586         input->nsimulations
00587         = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00588         if (error_code || input->nsimulations < 3)
00589         {
00590             msg = gettext ("Invalid population number");
00591             goto exit_on_error;
00592         }
00593     }
00594     else
00595     {
00596         msg = gettext ("No population number");
00597         goto exit_on_error;
00598     }
00599
00600     // Obtaining generations
00601     if (xmlHasProp (node, XML_NGENERATIONS))
00602     {
00603         input->niterations
00604         = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00605         if (error_code || !input->niterations)
00606         {
00607             msg = gettext ("Invalid generations number");
00608             goto exit_on_error;
00609         }
00610     }
00611     else
00612     {
00613         msg = gettext ("No generations number");
00614         goto exit_on_error;
00615     }
00616

```

```

00617     // Obtaining mutation probability
00618     if (xmlHasProp (node, XML_MUTATION))
00619     {
00620         input->mutation_ratio
00621         = xml_node_get_float (node, XML_MUTATION, &error_code);
00622         if (error_code || input->mutation_ratio < 0.
00623             || input->mutation_ratio >= 1.)
00624         {
00625             msg = gettext ("Invalid mutation probability");
00626             goto exit_on_error;
00627         }
00628     }
00629     else
00630     {
00631         msg = gettext ("No mutation probability");
00632         goto exit_on_error;
00633     }
00634
00635     // Obtaining reproduction probability
00636     if (xmlHasProp (node, XML_REPRODUCTION))
00637     {
00638         input->reproduction_ratio
00639         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00640         if (error_code || input->reproduction_ratio < 0.
00641             || input->reproduction_ratio >= 1.0)
00642         {
00643             msg = gettext ("Invalid reproduction probability");
00644             goto exit_on_error;
00645         }
00646     }
00647     else
00648     {
00649         msg = gettext ("No reproduction probability");
00650         goto exit_on_error;
00651     }
00652
00653     // Obtaining adaptation probability
00654     if (xmlHasProp (node, XML_ADAPTATION))
00655     {
00656         input->adaptation_ratio
00657         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00658         if (error_code || input->adaptation_ratio < 0.
00659             || input->adaptation_ratio >= 1.)
00660         {
00661             msg = gettext ("Invalid adaptation probability");
00662             goto exit_on_error;
00663         }
00664     }
00665     else
00666     {
00667         msg = gettext ("No adaptation probability");
00668         goto exit_on_error;
00669     }
00670
00671     // Checking survivals
00672     i = input->mutation_ratio * input->nsimulations;
00673     i += input->reproduction_ratio * input->
nsimulations;
00674     i += input->adaptation_ratio * input->
nsimulations;
00675     if (i > input->nsimulations - 2)
00676     {
00677         msg = gettext
00678             ("No enough survival entities to reproduce the population");
00679         goto exit_on_error;
00680     }
00681 }
00682 else
00683 {
00684     msg = gettext ("Unknown algorithm");
00685     goto exit_on_error;
00686 }
00687 xmlFree (buffer);
00688 buffer = NULL;
00689
00690 if (input->algorithm == ALGORITHM_MONTE_CARLO
00691     || input->algorithm == ALGORITHM_SWEEP)
00692 {
00693
00694     // Obtaining iterations number
00695     input->niterations
00696     = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00697     if (error_code == 1)
00698         input->niterations = 1;
00699     else if (error_code)
00700     {
00701         msg = gettext ("Bad iterations number");

```

```

00702         goto exit_on_error;
00703     }
00704
00705     // Obtaining best number
00706     if (xmlHasProp (node, XML_NBEST))
00707     {
00708         input->nbest = xml_node_get_uint (node,
XML_NBEST, &error_code);
00709         if (error_code || !input->nbest)
00710         {
00711             msg = gettext ("Invalid best number");
00712             goto exit_on_error;
00713         }
00714     }
00715     else
00716         input->nbest = 1;
00717
00718     // Obtaining tolerance
00719     if (xmlHasProp (node, XML_TOLERANCE))
00720     {
00721         input->tolerance
00722         = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00723         if (error_code || input->tolerance < 0.)
00724         {
00725             msg = gettext ("Invalid tolerance");
00726             goto exit_on_error;
00727         }
00728     }
00729     else
00730         input->tolerance = 0.;
00731
00732     // Getting gradient method parameters
00733     if (xmlHasProp (node, XML_NSTEPS))
00734     {
00735         input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
00736         if (error_code || !input->nsteps)
00737         {
00738             msg = gettext ("Invalid steps number");
00739             goto exit_on_error;
00740         }
00741         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00742         if (!xmlStrcmp (buffer, XML_COORDINATES))
00743             input->gradient_method =
GRADIENT_METHOD_COORDINATES;
00744         else if (!xmlStrcmp (buffer, XML_RANDOM))
00745         {
00746             input->gradient_method =
GRADIENT_METHOD_RANDOM;
00747             input->nestimates
00748             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00749             if (error_code || !input->nestimates)
00750             {
00751                 msg = gettext ("Invalid estimates number");
00752                 goto exit_on_error;
00753             }
00754         }
00755         else
00756         {
00757             msg = gettext ("Unknown method to estimate the gradient");
00758             goto exit_on_error;
00759         }
00760         xmlFree (buffer);
00761         buffer = NULL;
00762         if (xmlHasProp (node, XML_RELAXATION))
00763         {
00764             input->relaxation
00765             = xml_node_get_float (node, XML_RELAXATION, &error_code);
00766             if (error_code || input->relaxation < 0.
|| input->relaxation > 2.)
00767             {
00768                 msg = gettext ("Invalid relaxation parameter");
00769                 goto exit_on_error;
00770             }
00771         }
00772         else
00773             input->relaxation = DEFAULT_RELAXATION;
00774     }
00775     else
00776         input->nsteps = 0;
00777 }
00778
00779 // Reading the experimental data
00780 for (child = node->children; child; child = child->next)
00781 {
00782     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00783         break;
00784 }

```



```

00785 #if DEBUG
00786     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00787 #endif
00788     if (xmlHasProp (child, XML_NAME))
00789         buffer = xmlGetProp (child, XML_NAME);
00790     else
00791     {
00792         snprintf (buffer2, 64, "%s %u: %s",
00793             gettext ("Experiment"),
00794             input->nexperiments + 1, gettext ("no data file name"));
00795         msg = buffer2;
00796         goto exit_on_error;
00797     }
00798 #if DEBUG
00799     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00800 #endif
00801     input->weight = g_realloc (input->weight,
00802         (1 + input->nexperiments) * sizeof (double));
00803     if (xmlHasProp (child, XML_WEIGHT))
00804     {
00805         input->weight[input->nexperiments]
00806             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00807         if (error_code)
00808         {
00809             snprintf (buffer2, 64, "%s %s: %s",
00810                 gettext ("Experiment"), buffer, gettext ("bad weight"));
00811             msg = buffer2;
00812             goto exit_on_error;
00813         }
00814     }
00815     else
00816         input->weight[input->nexperiments] = 1.;
00817 #if DEBUG
00818     fprintf (stderr, "input_open: weight=%lg\n",
00819         input->weight[input->nexperiments]);
00820 #endif
00821     if (!input->nexperiments)
00822         input->ninputs = 0;
00823 #if DEBUG
00824     fprintf (stderr, "input_open: template[0]\n");
00825 #endif
00826     if (xmlHasProp (child, XML_TEMPLATE1))
00827     {
00828         input->template[0]
00829             = (char **) g_realloc (input->template[0],
00830                 (1 + input->nexperiments) * sizeof (char *));
00831         buffert[0] = (char *) xmlGetProp (child, template[0]);
00832 #if DEBUG
00833         fprintf (stderr, "input_open: experiment=%u templatel=%s\n",
00834             input->nexperiments, buffert[0]);
00835 #endif
00836         if (!input->nexperiments)
00837             ++input->ninputs;
00838 #if DEBUG
00839         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00840 #endif
00841     }
00842     else
00843     {
00844         snprintf (buffer2, 64, "%s %s: %s",
00845             gettext ("Experiment"), buffer, gettext ("no template"));
00846         msg = buffer2;
00847         goto exit_on_error;
00848     }
00849     for (i = 1; i < MAX_NINPUTS; ++i)
00850     {
00851 #if DEBUG
00852         fprintf (stderr, "input_open: template%u\n", i + 1);
00853 #endif
00854         if (xmlHasProp (child, template[i]))
00855         {
00856             if (input->nexperiments && input->ninputs <= i)
00857             {
00858                 snprintf (buffer2, 64, "%s %s: %s",
00859                     gettext ("Experiment"),
00860                     buffer, gettext ("bad templates number"));
00861                 msg = buffer2;
00862                 while (i-- > 0)
00863                     xmlFree (buffert[i]);
00864                 goto exit_on_error;
00865             }
00866             input->template[i] = (char **)
00867                 g_realloc (input->template[i],
00868                     (1 + input->nexperiments) * sizeof (char *));
00869             buffert[i] = (char *) xmlGetProp (child, template[i]);
00870 #if DEBUG
00871             fprintf (stderr, "input_open: experiment=%u template%u=%s\n",

```

```

00872             input->nexperiments, i + 1,
00873             input->template[i][input->nexperiments]);
00874 #endif
00875         if (!input->nexperiments)
00876             ++input->ninputs;
00877 #if DEBUG
00878         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00879 #endif
00880     }
00881     else if (input->nexperiments && input->ninputs >= i)
00882     {
00883         snprintf (buffer2, 64, "%s %s: %s%u",
00884                 gettext ("Experiment"),
00885                 buffer, gettext ("no template"), i + 1);
00886         msg = buffer2;
00887         while (i-- > 0)
00888             xmlFree (buffert[i]);
00889         goto exit_on_error;
00890     }
00891     else
00892         break;
00893 }
00894 input->experiment
00895 = g_realloc (input->experiment,
00896             (1 + input->nexperiments) * sizeof (char *));
00897 input->experiment[input->nexperiments] = (char *) buffer;
00898 for (i = 0; i < input->ninputs; ++i)
00899     input->template[i][input->nexperiments] = buffert[i];
00900 ++input->nexperiments;
00901 #if DEBUG
00902     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00903 #endif
00904 }
00905 if (!input->nexperiments)
00906 {
00907     msg = gettext ("No calibration experiments");
00908     goto exit_on_error;
00909 }
00910 buffer = NULL;
00911
00912 // Reading the variables data
00913 for (; child; child = child->next)
00914 {
00915     if (xmlStrcmp (child->name, XML_VARIABLE))
00916     {
00917         snprintf (buffer2, 64, "%s %u: %s",
00918                 gettext ("Variable"),
00919                 input->nvariables + 1, gettext ("bad XML node"));
00920         msg = buffer2;
00921         goto exit_on_error;
00922     }
00923     if (xmlHasProp (child, XML_NAME))
00924         buffer = xmlGetProp (child, XML_NAME);
00925     else
00926     {
00927         snprintf (buffer2, 64, "%s %u: %s",
00928                 gettext ("Variable"),
00929                 input->nvariables + 1, gettext ("no name"));
00930         msg = buffer2;
00931         goto exit_on_error;
00932     }
00933     if (xmlHasProp (child, XML_MINIMUM))
00934     {
00935         input->rangemin = g_realloc
00936             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00937         input->rangeminabs = g_realloc
00938             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00939         input->rangemin[input->nvariables]
00940             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00941         if (error_code)
00942         {
00943             snprintf (buffer2, 64, "%s %s: %s",
00944                     gettext ("Variable"), buffer, gettext ("bad minimum"));
00945             msg = buffer2;
00946             goto exit_on_error;
00947         }
00948         if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00949         {
00950             input->rangeminabs[input->nvariables]
00951                 = xml_node_get_float (child,
XML_ABSOLUTE_MINIMUM, &error_code);
00952             if (error_code)
00953             {
00954                 snprintf (buffer2, 64, "%s %s: %s",
00955                         gettext ("Variable"),
00956                         buffer, gettext ("bad absolute minimum"));
00957                 msg = buffer2;

```

```

00958         goto exit_on_error;
00959     }
00960 }
00961 else
00962     input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00963 if (input->rangemin[input->nvariables]
00964     < input->rangeminabs[input->nvariables])
00965 {
00966     snprintf (buffer2, 64, "%s %s: %s",
00967             gettext ("Variable"),
00968             buffer, gettext ("minimum range not allowed"));
00969     msg = buffer2;
00970     goto exit_on_error;
00971 }
00972 }
00973 else
00974 {
00975     snprintf (buffer2, 64, "%s %s: %s",
00976             gettext ("Variable"), buffer, gettext ("no minimum range"));
00977     msg = buffer2;
00978     goto exit_on_error;
00979 }
00980 if (xmlHasProp (child, XML_MAXIMUM))
00981 {
00982     input->rangemax = g_realloc
00983         (input->rangemax, (1 + input->nvariables) * sizeof (double));
00984     input->rangemaxabs = g_realloc
00985         (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00986     input->rangemax[input->nvariables]
00987         = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00988     if (error_code)
00989     {
00990         snprintf (buffer2, 64, "%s %s: %s",
00991             gettext ("Variable"), buffer, gettext ("bad maximum"));
00992         msg = buffer2;
00993         goto exit_on_error;
00994     }
00995     if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00996     {
00997         input->rangemaxabs[input->nvariables]
00998             = xml_node_get_float (child,
XML_ABSOLUTE_MAXIMUM, &error_code);
00999         if (error_code)
01000         {
01001             snprintf (buffer2, 64, "%s %s: %s",
01002                 gettext ("Variable"),
01003                 buffer, gettext ("bad absolute maximum"));
01004             msg = buffer2;
01005             goto exit_on_error;
01006         }
01007     }
01008     else
01009         input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01010     if (input->rangemax[input->nvariables]
01011         > input->rangemaxabs[input->nvariables])
01012     {
01013         snprintf (buffer2, 64, "%s %s: %s",
01014             gettext ("Variable"),
01015             buffer, gettext ("maximum range not allowed"));
01016         msg = buffer2;
01017         goto exit_on_error;
01018     }
01019 }
01020 else
01021 {
01022     snprintf (buffer2, 64, "%s %s: %s",
01023             gettext ("Variable"), buffer, gettext ("no maximum range"));
01024     msg = buffer2;
01025     goto exit_on_error;
01026 }
01027 if (input->rangemax[input->nvariables]
01028     < input->rangemin[input->nvariables])
01029 {
01030     snprintf (buffer2, 64, "%s %s: %s",
01031             gettext ("Variable"), buffer, gettext ("bad range"));
01032     msg = buffer2;
01033     goto exit_on_error;
01034 }
01035 input->precision = g_realloc
01036     (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01037 if (xmlHasProp (child, XML_PRECISION))
01038 {
01039     input->precision[input->nvariables]
01040         = xml_node_get_uint (child, XML_PRECISION, &error_code);
01041     if (error_code || input->precision[input->
nvariables] >= NPRECISIONS)
01042     {

```

```

01043         snprintf (buffer2, 64, "%s %s: %s",
01044                     gettext ("Variable"),
01045                     buffer, gettext ("bad precision"));
01046         msg = buffer2;
01047         goto exit_on_error;
01048     }
01049 }
01050 else
01051     input->precision[input->nvariables] =
DEFAULT_PRECISION;
01052 if (input->algorithm == ALGORITHM_SWEEP)
01053 {
01054     if (xmlHasProp (child, XML_NSWEEPS))
01055     {
01056         input->nsweeps = (unsigned int *)
01057             g_realloc (input->nsweeps,
01058                 (1 + input->nvariables) * sizeof (unsigned int));
01059         input->nsweeps[input->nvariables]
01060             = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01061         if (error_code || !input->nsweeps[input->
nvariables])
01062         {
01063             snprintf (buffer2, 64, "%s %s: %s",
01064                 gettext ("Variable"),
01065                 buffer, gettext ("bad sweeps"));
01066             msg = buffer2;
01067             goto exit_on_error;
01068         }
01069     }
01070     else
01071     {
01072         snprintf (buffer2, 64, "%s %s: %s",
01073             gettext ("Variable"),
01074             buffer, gettext ("no sweeps number"));
01075         msg = buffer2;
01076         goto exit_on_error;
01077     }
01078 #if DEBUG
01079     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01080         input->nsweeps[input->nvariables],
01081         input->nsimulations);
01082 #endif
01083 }
01084 if (input->algorithm == ALGORITHM_GENETIC)
01085 {
01086     // Obtaining bits representing each variable
01087     if (xmlHasProp (child, XML_NBITS))
01088     {
01089         input->nbits = (unsigned int *)
01090             g_realloc (input->nbits,
01091                 (1 + input->nvariables) * sizeof (unsigned int));
01092         i = xml_node_get_uint (child, XML_NBITS, &error_code);
01093         if (error_code || !i)
01094         {
01095             snprintf (buffer2, 64, "%s %s: %s",
01096                 gettext ("Variable"),
01097                 buffer, gettext ("invalid bits number"));
01098             msg = buffer2;
01099             goto exit_on_error;
01100         }
01101         input->nbits[input->nvariables] = i;
01102     }
01103     else
01104     {
01105         snprintf (buffer2, 64, "%s %s: %s",
01106             gettext ("Variable"),
01107             buffer, gettext ("no bits number"));
01108         msg = buffer2;
01109         goto exit_on_error;
01110     }
01111 }
01112 else if (input->nsteps)
01113 {
01114     input->step = (double *)
01115         g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01116     input->step[input->nvariables]
01117         = xml_node_get_float (child, XML_STEP, &error_code);
01118     if (error_code || input->step[input->nvariables] < 0.)
01119     {
01120         snprintf (buffer2, 64, "%s %s: %s",
01121             gettext ("Variable"),
01122             buffer, gettext ("bad step size"));
01123         msg = buffer2;
01124         goto exit_on_error;
01125     }
01126 }
01127 input->label = g_realloc

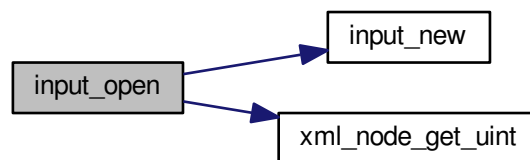
```

```

01127         (input->label, (1 + input->nvariables) * sizeof (char *));
01128     input->label[input->nvariables] = (char *) buffer;
01129     ++input->nvariables;
01130 }
01131 if (!input->nvariables)
01132 {
01133     msg = gettext ("No calibration variables");
01134     goto exit_on_error;
01135 }
01136 buffer = NULL;
01137
01138 // Getting the working directory
01139 input->directory = g_path_get_dirname (filename);
01140 input->name = g_path_get_basename (filename);
01141
01142 // Closing the XML document
01143 xmlFreeDoc (doc);
01144
01145 #if DEBUG
01146     fprintf (stderr, "input_open: end\n");
01147 #endif
01148     return 1;
01149
01150 exit_on_error:
01151     xmlFree (buffer);
01152     xmlFreeDoc (doc);
01153     show_error (msg);
01154     input_free ();
01155 #if DEBUG
01156     fprintf (stderr, "input_open: end\n");
01157 #endif
01158     return 0;
01159 }

```

Here is the call graph for this function:



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

Function to show a dialog with an error message.

#### Parameters

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

Definition at line 256 of file [mpcotool.c](#).

```

00257 {
00258     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00259 }

```

Here is the call graph for this function:



#### 5.7.3.13 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 226 of file [mpcotool.c](#).

```

00227 {
00228     #if HAVE_GTK
00229         GtkMessageDialog *dlg;
00230
00231         // Creating the dialog
00232         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00233             (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00234
00235         // Setting the dialog title
00236         gtk_window_set_title (GTK_WINDOW (dlg), title);
00237
00238         // Showing the dialog and waiting response
00239         gtk_dialog_run (GTK_DIALOG (dlg));
00240
00241         // Closing and freeing memory
00242         gtk_widget_destroy (GTK_WIDGET (dlg));
00243
00244     #else
00245         printf ("%s: %s\n", title, msg);
00246     #endif
00247 }
  
```

#### 5.7.3.14 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 336 of file [mpcotool.c](#).

```

00337 {
  
```

```

00338     double x = 0.;
00339     xmlChar *buffer;
00340     buffer = xmlGetProp (node, prop);
00341     if (!buffer)
00342         *error_code = 1;
00343     else
00344     {
00345         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346             *error_code = 2;
00347         else
00348             *error_code = 0;
00349         xmlFree (buffer);
00350     }
00351     return x;
00352 }

```

#### 5.7.3.15 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 274 of file [mpcotool.c](#).

```

00275 {
00276     int i = 0;
00277     xmlChar *buffer;
00278     buffer = xmlGetProp (node, prop);
00279     if (!buffer)
00280         *error_code = 1;
00281     else
00282     {
00283         if (sscanf ((char *) buffer, "%d", &i) != 1)
00284             *error_code = 2;
00285         else
00286             *error_code = 0;
00287         xmlFree (buffer);
00288     }
00289     return i;
00290 }

```

#### 5.7.3.16 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 305 of file [mpcotool.c](#).

```

00306 {
00307     unsigned int i = 0;
00308     xmlChar *buffer;
00309     buffer = xmlGetProp (node, prop);
00310     if (!buffer)
00311         *error_code = 1;
00312     else
00313     {
00314         if (sscanf ((char *) buffer, "%u", &i) != 1)
00315             *error_code = 2;
00316         else
00317             *error_code = 0;
00318         xmlFree (buffer);
00319     }
00320     return i;
00321 }

```

### 5.7.3.17 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 403 of file [mpcotoool.c](#).

```

00404 {
00405     xmlChar buffer[64];
00406     snprintf ((char *) buffer, 64, "%.14lg", value);
00407     xmlSetProp (node, prop, buffer);
00408 }

```

### 5.7.3.18 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 365 of file [mpcotoool.c](#).

```

00366 {
00367     xmlChar buffer[64];
00368     snprintf ((char *) buffer, 64, "%d", value);
00369     xmlSetProp (node, prop, buffer);
00370 }

```

### 5.7.3.19 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 384 of file [mpcotool.c](#).

```

00385 {
00386     xmlChar buffer[64];
00387     snprintf ((char *) buffer, 64, "%u", value);
00388     xmlSetProp (node, prop, buffer);
00389 }
```

## 5.8 mpcotool.h

```

00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef CALIBRATOR__H
00037 #define CALIBRATOR__H 1
00038
00043 enum Algorithm
00044 {
00045     ALGORITHM_MONTE_CARLO = 0,
00046     ALGORITHM_SWEEP = 1,
00047     ALGORITHM_GENETIC = 2
00048 };
00049
00054 enum GradientMethod
00055 {
00056     GRADIENT_METHOD_COORDINATES = 0,
00057     GRADIENT_METHOD_RANDOM = 1,
00058 };
00059
00064 typedef struct
00065 {
00066     char **template[MAX_NINPUTS];
00067     char **experiment;
00068     char **label;
00069     char *result;
00070     char *variables;
00071     char *simulator;
00072     char *evaluator;
00074     char *directory;
00075     char *name;
00076     double *rangemin;
00077     double *rangemax;
00078     double *rangeminabs;
00079     double *rangemaxabs;
00080     double *weight;
00081     double *step;
00082     unsigned int *precision;
00083     unsigned int *nsweeps;
00084     unsigned int *nbits;
00086     double tolerance;
00087     double mutation_ratio;
00088     double reproduction_ratio;
```

```

00089 double adaptation_ratio;
00090 double relaxation;
00091 unsigned long int seed;
00093 unsigned int nvariables;
00094 unsigned int nexperiments;
00095 unsigned int ninputs;
00096 unsigned int nsimulations;
00097 unsigned int algorithm;
00098 unsigned int nsteps;
00100 unsigned int gradient_method;
00101 unsigned int nestimates;
00103 unsigned int niterations;
00104 unsigned int nbest;
00105 } Input;
00106
00111 typedef struct
00112 {
00113     GMappedFile **file[MAX_NINPUTS];
00114     char **template[MAX_NINPUTS];
00115     char **experiment;
00116     char **label;
00117     gsl_rng *rng;
00118     GeneticVariable *genetic_variable;
00120     FILE *file_result;
00121     FILE *file_variables;
00122     char *result;
00123     char *variables;
00124     char *simulator;
00125     char *evaluator;
00127     double *value;
00128     double *rangemin;
00129     double *rangemax;
00130     double *rangeminabs;
00131     double *rangemaxabs;
00132     double *error_best;
00133     double *weight;
00134     double *step;
00135     double *gradient;
00136     double *value_old;
00138     double *error_old;
00140     unsigned int *precision;
00141     unsigned int *nsweeps;
00142     unsigned int *thread;
00144     unsigned int *thread_gradient;
00147     unsigned int *simulation_best;
00148     double tolerance;
00149     double mutation_ratio;
00150     double reproduction_ratio;
00151     double adaptation_ratio;
00152     double relaxation;
00153     double calculation_time;
00154     unsigned long int seed;
00156     unsigned int nvariables;
00157     unsigned int nexperiments;
00158     unsigned int ninputs;
00159     unsigned int nsimulations;
00160     unsigned int gradient_method;
00161     unsigned int nsteps;
00163     unsigned int nestimates;
00165     unsigned int algorithm;
00166     unsigned int nstart;
00167     unsigned int nend;
00168     unsigned int nstart_gradient;
00170     unsigned int nend_gradient;
00172     unsigned int niterations;
00173     unsigned int nbest;
00174     unsigned int nsaveds;
00175 #if HAVE_MPI
00176     int mpi_rank;
00177 #endif
00178 } Calibrate;
00179
00184 typedef struct
00185 {
00186     unsigned int thread;
00187 } ParallelData;
00188
00189 // Public functions
00190 void show_message (char *title, char *msg, int type);
00191 void show_error (char *msg);
00192 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00193 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00194                                 int *error_code);
00195 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00196                             int *error_code);
00197 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00198 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,

```

```
00199             unsigned int value);
00200 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00201 void input_new ();
00202 void input_free ();
00203 int input_open (char *filename);
00204 void calibrate_input (unsigned int simulation, char *input,
00205                      GMappedFile * template);
00206 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00207 void calibrate_print ();
00208 void calibrate_save_variables (unsigned int simulation, double error);
00209 void calibrate_best (unsigned int simulation, double value);
00210 void calibrate_sequential ();
00211 void *calibrate_thread (ParallelData * data);
00212 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00213                     double *error_best);
00214 #if HAVE_MPI
00215 void calibrate_synchronise ();
00216 #endif
00217 void calibrate_sweep ();
00218 void calibrate_MonteCarlo ();
00219 void calibrate_best_gradient (unsigned int simulation, double value);
00220 void calibrate_gradient_sequential ();
00221 void *calibrate_gradient_thread (ParallelData * data);
00222 double calibrate_variable_step_gradient (unsigned int variable);
00223 void calibrate_step_gradient (unsigned int simulation);
00224 void calibrate_gradient ();
00225 double calibrate_genetic_objective (Entity * entity);
00226 void calibrate_genetic ();
00227 void calibrate_save_old ();
00228 void calibrate_merge_old ();
00229 void calibrate_refine ();
00230 void calibrate_step ();
00231 void calibrate_iterate ();
00232 void calibrate_open ();
00233
00234 #endif
```



# Index

ALGORITHM\_GENETIC  
    mpcotoool.h, 133  
ALGORITHM\_MONTE\_CARLO  
    mpcotoool.h, 133  
ALGORITHM\_SWEEP  
    mpcotoool.h, 133  
Algorithm  
    mpcotoool.h, 133  
  
Calibrate, 13  
    thread\_gradient, 15  
calibrate\_best  
    mpcotoool.c, 45  
    mpcotoool.h, 134  
calibrate\_best\_gradient  
    mpcotoool.c, 46  
    mpcotoool.h, 134  
calibrate\_estimate\_gradient\_coordinates  
    mpcotoool.c, 46  
calibrate\_estimate\_gradient\_random  
    mpcotoool.c, 47  
calibrate\_genetic\_objective  
    mpcotoool.c, 47  
    mpcotoool.h, 135  
calibrate\_gradient\_sequential  
    mpcotoool.c, 48  
calibrate\_gradient\_thread  
    mpcotoool.c, 49  
    mpcotoool.h, 135  
calibrate\_input  
    mpcotoool.c, 50  
    mpcotoool.h, 136  
calibrate\_merge  
    mpcotoool.c, 51  
    mpcotoool.h, 137  
calibrate\_parse  
    mpcotoool.c, 52  
    mpcotoool.h, 138  
calibrate\_save\_variables  
    mpcotoool.c, 53  
    mpcotoool.h, 140  
calibrate\_step\_gradient  
    mpcotoool.c, 54  
    mpcotoool.h, 140  
calibrate\_thread  
    mpcotoool.c, 55  
    mpcotoool.h, 142  
config.h, 25  
cores\_number  
    interface.h, 31  
    mpcotoool.c, 56  
Experiment, 15  
format  
    mpcotoool.c, 78  
GRADIENT\_METHOD\_COORDINATES  
    mpcotoool.h, 133  
GRADIENT\_METHOD\_RANDOM  
    mpcotoool.h, 133  
GradientMethod  
    mpcotoool.h, 133  
  
Input, 16  
input\_open  
    mpcotoool.c, 56  
    mpcotoool.h, 143  
input\_save  
    interface.h, 31  
    mpcotoool.c, 64  
input\_save\_gradient  
    mpcotoool.c, 67  
interface.h, 29  
    cores\_number, 31  
    input\_save, 31  
    window\_get\_algorithm, 33  
    window\_get\_gradient, 34  
    window\_read, 34  
    window\_save, 36  
    window\_template\_experiment, 37  
  
main  
    mpcotoool.c, 68  
mpcotoool.c, 40  
    calibrate\_best, 45  
    calibrate\_best\_gradient, 46  
    calibrate\_estimate\_gradient\_coordinates, 46  
    calibrate\_estimate\_gradient\_random, 47  
    calibrate\_genetic\_objective, 47  
    calibrate\_gradient\_sequential, 48  
    calibrate\_gradient\_thread, 49  
    calibrate\_input, 50  
    calibrate\_merge, 51  
    calibrate\_parse, 52  
    calibrate\_save\_variables, 53  
    calibrate\_step\_gradient, 54  
    calibrate\_thread, 55  
    cores\_number, 56  
    format, 78  
    input\_open, 56

- input\_save, 64
- input\_save\_gradient, 67
- main, 68
- precision, 78
- show\_error, 70
- show\_message, 71
- template, 79
- window\_get\_algorithm, 71
- window\_get\_gradient, 71
- window\_read, 72
- window\_save, 74
- window\_template\_experiment, 75
- xml\_node\_get\_float, 76
- xml\_node\_get\_int, 76
- xml\_node\_get\_uint, 77
- xml\_node\_set\_float, 77
- xml\_node\_set\_int, 78
- xml\_node\_set\_uint, 78
- mpcotool.h, 131
  - ALGORITHM\_GENETIC, 133
  - ALGORITHM\_MONTE\_CARLO, 133
  - ALGORITHM\_SWEEP, 133
  - Algorithm, 133
  - calibrate\_best, 134
  - calibrate\_best\_gradient, 134
  - calibrate\_genetic\_objective, 135
  - calibrate\_gradient\_thread, 135
  - calibrate\_input, 136
  - calibrate\_merge, 137
  - calibrate\_parse, 138
  - calibrate\_save\_variables, 140
  - calibrate\_step\_gradient, 140
  - calibrate\_thread, 142
  - GRADIENT\_METHOD\_COORDINATES, 133
  - GRADIENT\_METHOD\_RANDOM, 133
  - GradientMethod, 133
  - input\_open, 143
  - show\_error, 151
  - show\_message, 152
  - xml\_node\_get\_float, 152
  - xml\_node\_get\_int, 153
  - xml\_node\_get\_uint, 153
  - xml\_node\_set\_float, 154
  - xml\_node\_set\_int, 154
  - xml\_node\_set\_uint, 154
- Options, 17
- ParallelData, 18
- precision
  - mpcotool.c, 78
- Running, 19
- show\_error
  - mpcotool.c, 70
  - mpcotool.h, 151
- show\_message
  - mpcotool.c, 71
- mpcotool.h, 152
- template
  - mpcotool.c, 79
- thread\_gradient
  - Calibrate, 15
- Variable, 19
- Window, 20
- window\_get\_algorithm
  - interface.h, 33
  - mpcotool.c, 71
- window\_get\_gradient
  - interface.h, 34
  - mpcotool.c, 71
- window\_read
  - interface.h, 34
  - mpcotool.c, 72
- window\_save
  - interface.h, 36
  - mpcotool.c, 74
- window\_template\_experiment
  - interface.h, 37
  - mpcotool.c, 75
- xml\_node\_get\_float
  - mpcotool.c, 76
  - mpcotool.h, 152
- xml\_node\_get\_int
  - mpcotool.c, 76
  - mpcotool.h, 153
- xml\_node\_get\_uint
  - mpcotool.c, 77
  - mpcotool.h, 153
- xml\_node\_set\_float
  - mpcotool.c, 77
  - mpcotool.h, 154
- xml\_node\_set\_int
  - mpcotool.c, 78
  - mpcotool.h, 154
- xml\_node\_set\_uint
  - mpcotool.c, 78
  - mpcotool.h, 154