

# Calibrator

1.2.5

Generated by Doxygen 1.8.9.1

Mon Jan 11 2016 05:35:03



# 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	34
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	38
5.4	interface.h	38
5.5	mpcotool.c File Reference	41
5.5.1	Detailed Description	46
5.5.2	Function Documentation	46
5.5.2.1	calibrate_best	46
5.5.2.2	calibrate_best_gradient	46
5.5.2.3	calibrate_estimate_gradient_coordinates	47
5.5.2.4	calibrate_estimate_gradient_random	47
5.5.2.5	calibrate_genetic_objective	48
5.5.2.6	calibrate_gradient_sequential	49
5.5.2.7	calibrate_gradient_thread	50
5.5.2.8	calibrate_input	51
5.5.2.9	calibrate_merge	53
5.5.2.10	calibrate_parse	53
5.5.2.11	calibrate_save_variables	55
5.5.2.12	calibrate_step_gradient	55
5.5.2.13	calibrate_thread	57
5.5.2.14	cores_number	58
5.5.2.15	input_open	58
5.5.2.16	input_save	66
5.5.2.17	input_save_gradient	69
5.5.2.18	main	70
5.5.2.19	show_error	72
5.5.2.20	show_message	73
5.5.2.21	window_get_algorithm	73
5.5.2.22	window_get_gradient	74
5.5.2.23	window_read	74
5.5.2.24	window_save	76
5.5.2.25	window_template_experiment	78
5.5.2.26	xml_node_get_float	78
5.5.2.27	xml_node_get_float_with_default	79

5.5.2.28	<a href="#">xml_node_get_int</a>	80
5.5.2.29	<a href="#">xml_node_get_uint</a>	80
5.5.2.30	<a href="#">xml_node_get_uint_with_default</a>	81
5.5.2.31	<a href="#">xml_node_set_float</a>	81
5.5.2.32	<a href="#">xml_node_set_int</a>	82
5.5.2.33	<a href="#">xml_node_set_uint</a>	82
5.5.3	Variable Documentation	82
5.5.3.1	<a href="#">format</a>	82
5.5.3.2	<a href="#">precision</a>	83
5.5.3.3	<a href="#">template</a>	83
5.6	<a href="#">mpcotool.c</a>	83
5.7	<a href="#">mpcotool.h</a> File Reference	136
5.7.1	Detailed Description	139
5.7.2	Enumeration Type Documentation	139
5.7.2.1	<a href="#">Algorithm</a>	139
5.7.2.2	<a href="#">GradientMethod</a>	139
5.7.3	Function Documentation	140
5.7.3.1	<a href="#">calibrate_best</a>	140
5.7.3.2	<a href="#">calibrate_best_gradient</a>	140
5.7.3.3	<a href="#">calibrate_genetic_objective</a>	141
5.7.3.4	<a href="#">calibrate_gradient_thread</a>	141
5.7.3.5	<a href="#">calibrate_input</a>	142
5.7.3.6	<a href="#">calibrate_merge</a>	144
5.7.3.7	<a href="#">calibrate_parse</a>	144
5.7.3.8	<a href="#">calibrate_save_variables</a>	146
5.7.3.9	<a href="#">calibrate_step_gradient</a>	146
5.7.3.10	<a href="#">calibrate_thread</a>	148
5.7.3.11	<a href="#">input_open</a>	149
5.7.3.12	<a href="#">show_error</a>	157
5.7.3.13	<a href="#">show_message</a>	158
5.7.3.14	<a href="#">xml_node_get_float</a>	159
5.7.3.15	<a href="#">xml_node_get_float_with_default</a>	159
5.7.3.16	<a href="#">xml_node_get_int</a>	160
5.7.3.17	<a href="#">xml_node_get_uint</a>	161
5.7.3.18	<a href="#">xml_node_get_uint_with_default</a>	161
5.7.3.19	<a href="#">xml_node_set_float</a>	162
5.7.3.20	<a href="#">xml_node_set_int</a>	162
5.7.3.21	<a href="#">xml_node_set_uint</a>	162
5.8	<a href="#">mpcotool.h</a>	163

<a href="#">Index</a>	167
-----------------------	-----

# Chapter 1

## MPCOTool

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

### VERSIONS

- 1.2.5: Stable and recommended version.
- 1.3.9: 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:

- 1.2.5/configure.ac: configure generator.
- 1.2.5/Makefile.in: Makefile generator.
- 1.2.5/config.h.in: config header generator.
- 1.2.5/mpcotool.c: main source code.
- 1.2.5/mpcotool.h: main header code.
- 1.2.5/interface.h: interface header code.
- 1.2.5/build: script to build all.
- 1.2.5/logo.png: logo figure.
- 1.2.5/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/1.2.5
$ 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/1.2.5):

```
$ 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 ../1.2.5
$ 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 . . . . .	13
<a href="#">Experiment</a>	Struct to define experiment data . . . . .	15
<a href="#">Input</a>	Struct to define the calibration input file . . . . .	16
<a href="#">Options</a>	Struct to define the options dialog . . . . .	17
<a href="#">ParallelData</a>	Struct to pass to the GThreads parallelized function . . . . .	18
<a href="#">Running</a>	Struct to define the running dialog . . . . .	19
<a href="#">Variable</a>	Struct to define variable data . . . . .	19
<a href="#">Window</a>	Struct to define the main window . . . . .	20





## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">config.h</a>	Configuration header file . . . . .	25
<a href="#">interface.h</a>	Header file of the interface . . . . .	29
<a href="#">mpcotool.c</a>	Source file of the mpcotool . . . . .	41
<a href="#">mpcotool.h</a>	Header file of the mpcotool . . . . .	136
<a href="#">prueba.c</a>	. . . . .	??



## 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](#)
- unsigned int \* [simulation\\_best](#)  
*Array of best simulation numbers.*
- double [tolerance](#)  
*Algorithm tolerance.*
- double [mutation\\_ratio](#)  
*Mutation probability.*
- double [reproduction\\_ratio](#)  
*Reproduction probability.*
- double [adaptation\\_ratio](#)  
*Adaptation probability.*
- double [relaxation](#)  
*Relaxation parameter.*
- double [calculation\\_time](#)  
*Calculation time.*
- 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 [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 76 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 94 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.*
- `double step`  
*Initial step size for the gradient based method.*
- `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:

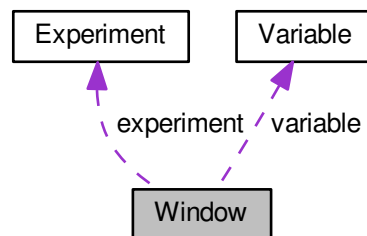
- [interface.h](#)

## 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.*
- GtkLabel \* [label\\_step](#)

*GtkLabel to set the step.*
- GtkSpinButton \* [spin\\_step](#)

*GtkSpinButton to set the step.*
- GtkScrolledWindow \* [scrolled\\_step](#)

*step GtkScrolledWindow.*
- GtkFrame \* [frame\\_experiment](#)

*Experiment GtkFrame.*
- GtkGrid \* [grid\\_experiment](#)

*Experiment GtkGrid.*
- GtkComboBoxText \* [combo\\_experiment](#)

*Experiment GtkComboBoxEntry.*
- GtkButton \* [button\\_add\\_experiment](#)

*GtkButton to add a experiment.*
- GtkButton \* [button\\_remove\\_experiment](#)

*GtkButton to remove a experiment.*
- GtkLabel \* [label\\_experiment](#)

*Experiment GtkLabel.*
- GtkFileChooserButton \* [button\\_experiment](#)

*GtkFileChooserButton to set the experimental data file.*
- GtkLabel \* [label\\_weight](#)

*Weight GtkLabel.*
- GtkSpinButton \* [spin\\_weight](#)

*Weight GtkSpinButton.*

- GtkCheckButton \* [check\\_template](#) [MAX\_NINPUTS]  
*Array of GtkCheckButtons to set the input templates.*
- GtkFileChooserButton \* [button\\_template](#) [MAX\_NINPUTS]  
*Array of GtkFileChooserButtons to set the input templates.*
- GdkPixbuf \* [logo](#)  
*Logo GdkPixbuf.*
- [Experiment](#) \* [experiment](#)  
*Array of experiments data.*
- [Variable](#) \* [variable](#)  
*Array of variables data.*
- char \* [application\\_directory](#)  
*Application directory.*
- gulong [id\\_experiment](#)  
*Identifier of the combo\_experiment signal.*
- gulong [id\\_experiment\\_name](#)  
*Identifier of the button\_experiment signal.*
- gulong [id\\_variable](#)  
*Identifier of the combo\_variable signal.*
- gulong [id\\_variable\\_label](#)  
*Identifier of the entry\_variable signal.*
- gulong [id\\_template](#) [MAX\_NINPUTS]  
*Array of identifiers of the check\_template signal.*
- gulong [id\\_input](#) [MAX\_NINPUTS]  
*Array of identifiers of the button\_template signal.*
- unsigned int [nexperiments](#)  
*Number of experiments.*
- unsigned int [nvariables](#)  
*Number of variables.*

#### 4.8.1 Detailed Description

Struct to define the main window.

Definition at line 104 of file [interface.h](#).

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

- [interface.h](#)

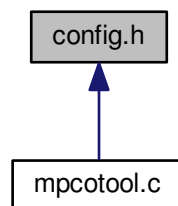
## Chapter 5

# 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
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
00044 #define NGRADIENTS 2
00045 #define NPRECISIONS 15
00046
00047
00048 // Default choices
00049
00050 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00051 #define DEFAULT_RANDOM_SEED 7007
00052 #define DEFAULT_RELAXATION 1.
00053
00054 // Interface labels
00055
00056 #define LOCALE_DIR "locales"
00057 #define PROGRAM_INTERFACE "mpcotool"
00058
00059 // XML labels
00060
00061 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00062 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00063 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00064 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00065 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00066 #define XML_COORDINATES (const xmlChar*)"coordinates"
00067 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00068 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00069 #define XML_GENETIC (const xmlChar*)"genetic"
00070 #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
00071 #define XML_MINIMUM (const xmlChar*)"minimum"
00072 #define XML_MAXIMUM (const xmlChar*)"maximum"
00073 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00074 #define XML_MUTATION (const xmlChar*)"mutation"
00075 #define XML_NAME (const xmlChar*)"name"
00076 #define XML_NBEST (const xmlChar*)"nbest"
00077 #define XML_NBITS (const xmlChar*)"nbits"
00078 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00079 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00080 #define XML_NITERATIONS (const xmlChar*)"niterations"
00081 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00082 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00083 #define XML_NSTEPS (const xmlChar*)"nsteps"
00084 #define XML_NSWEEPS (const xmlChar*)"nsweps"
00085 #define XML_PRECISION (const xmlChar*)"precision"
00086 #define XML_RANDOM (const xmlChar*)"random"
00087 #define XML_RELAXATION (const xmlChar*)"relaxation"
00088 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00089 #define XML_RESULT (const xmlChar*)"result"
00090 #define XML_SIMULATOR (const xmlChar*)"simulator"
00091 #define XML_SEED (const xmlChar*)"seed"

```

```

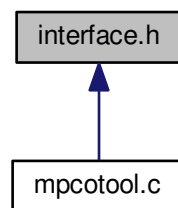
00111 #define XML_STEP (const xmlChar*)"step"
00112 #define XML_SWEEP (const xmlChar*)"sweep"
00113 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00114 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00116 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00118 #define XML_TEMPLATE4 (const xmlChar*)"template4"
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 [4875](#) of file [mpcotool.c](#).

```

04876 {
04877     #ifdef G_OS_WIN32
04878         SYSTEM_INFO sysinfo;
04879         GetSystemInfo (&sysinfo);
04880         return sysinfo.dwNumberOfProcessors;
04881     #else
04882         return (int) sysconf (_SC_NPROCESSORS_ONLN);
04883     #endif
04884 }
```

#### 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 2699 of file `mpcotool.c`.

```

02700 {
02701     unsigned int i, j;
02702     char *buffer;
02703     xmlDoc *doc;
02704     xmlNode *node, *child;
02705     GFile *file, *file2;
02706
02707     #if DEBUG
02708         fprintf (stderr, "input_save: start\n");
02709     #endif
02710
02711     // Getting the input file directory
02712     input->name = g_path_get_basename (filename);
02713     input->directory = g_path_get_dirname (filename);
02714     file = g_file_new_for_path (input->directory);
02715
02716     // Opening the input file
02717     doc = xmlNewDoc ((const xmlChar *) "1.0");
02718
02719     // Setting root XML node
02720     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02721     xmlDocSetRootElement (doc, node);
02722
02723     // Adding properties to the root XML node
02724     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02725         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02726     if (xmlStrcmp ((const xmlChar *) input->variables,
02727         variables_name))
02728         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02729         variables);
02730     file2 = g_file_new_for_path (input->simulator);
02731     buffer = g_file_get_relative_path (file, file2);
02732     g_object_unref (file2);
02733     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02734     g_free (buffer);
02735     if (input->evaluator)
02736     {
02737         file2 = g_file_new_for_path (input->evaluator);
02738         buffer = g_file_get_relative_path (file, file2);
02739         g_object_unref (file2);
02740         if (xmlStrlen ((xmlChar *) buffer))
02741             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02742         g_free (buffer);
02743     }
02744     if (input->seed != DEFAULT_RANDOM_SEED)
02745         xml_node_set_uint (node, XML_SEED, input->seed);
02746
02747     // Setting the algorithm
02748     buffer = (char *) g_malloc (64);
02749     switch (input->algorithm)
02750     {
02751     case ALGORITHM_MONTE_CARLO:
02752         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02753         snprintf (buffer, 64, "%u", input->nsimulations);
02754         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02755         snprintf (buffer, 64, "%u", input->niterations);
02756         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02757         snprintf (buffer, 64, "%.3lg", input->tolerance);
02758         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02759         snprintf (buffer, 64, "%u", input->nbest);
02760         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02761         input_save_gradient (node);
02762         break;
02763     case ALGORITHM_SWEEP:
02764         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02765         snprintf (buffer, 64, "%u", input->niterations);
02766         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02767         snprintf (buffer, 64, "%.3lg", input->tolerance);
02768         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02769         snprintf (buffer, 64, "%u", input->nbest);
02770         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02771         input_save_gradient (node);
02772         break;
02773     default:
02774         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02775         snprintf (buffer, 64, "%u", input->nsimulations);
02776         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02777         snprintf (buffer, 64, "%u", input->niterations);
02778         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02779         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);

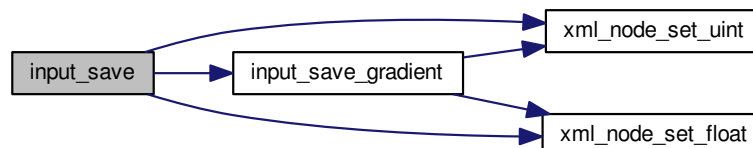
```

```

02778     xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02779     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02780     xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02781     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02782     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02783     break;
02784 }
02785 g_free (buffer);
02786
02787 // Setting the experimental data
02788 for (i = 0; i < input->nexperiments; ++i)
02789 {
02790     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02791     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02792     if (input->weight[i] != 1.)
02793         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02794     for (j = 0; j < input->ninputs; ++j)
02795         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02796 }
02797
02798 // Setting the variables data
02799 for (i = 0; i < input->nvariables; ++i)
02800 {
02801     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02802     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02803     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02804     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02805         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02806     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02807     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02808         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02809     if (input->precision[i] != DEFAULT_PRECISION)
02810         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02811     if (input->algorithm == ALGORITHM_SWEEP)
02812         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02813     else if (input->algorithm == ALGORITHM_GENETIC)
02814         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02815     if (input->nsteps)
02816         xml_node_set_float (child, XML_STEP, input->
step[i]);
02817 }
02818
02819 // Saving the XML file
02820 xmlSaveFormatFile (filename, doc, 1);
02821
02822 // Freeing memory
02823 xmlFreeDoc (doc);
02824
02825 #if DEBUG
02826 fprintf (stderr, "input_save: end\n");
02827 #endif
02828 }

```

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 2932 of file [mpcotool.c](#).

```

02933 {
02934     unsigned int i;
02935     #if DEBUG
02936         fprintf (stderr, "window_get_algorithm: start\n");
02937     #endif
02938     for (i = 0; i < NALGORITHMS; ++i)
02939         if (gtk_toggle_button_get_active
02940             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02941             break;
02942     #if DEBUG
02943         fprintf (stderr, "window_get_algorithm: %u\n", i);
02944         fprintf (stderr, "window_get_algorithm: end\n");
02945     #endif
02946     return i;
02947 }
```

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

Function to get the gradient base method number.

#### Returns

Gradient base method number.

Definition at line 2955 of file [mpcotool.c](#).

```

02956 {
02957     unsigned int i;
02958     #if DEBUG
02959         fprintf (stderr, "window_get_gradient: start\n");
02960     #endif
02961     for (i = 0; i < NGRADIENTS; ++i)
02962         if (gtk_toggle_button_get_active
02963             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02964             break;
02965     #if DEBUG
02966         fprintf (stderr, "window_get_gradient: %u\n", i);
02967         fprintf (stderr, "window_get_gradient: end\n");
02968     #endif
02969     return i;
02970 }
```

### 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 4052 of file [mpcotool.c](#).



```

04053 {
04054     unsigned int i;
04055     char *buffer;
04056     #if DEBUG
04057     fprintf (stderr, "window_read: start\n");
04058     #endif
04059     // Reading new input file
04060     input_free ();
04061     if (!input_open (filename))
04062         return 0;
04063     // Setting GTK+ widgets data
04064     gtk_entry_set_text (window->entry_result, input->result);
04065     gtk_entry_set_text (window->entry_variables, input->
variables);
04066     buffer = g_build_filename (input->directory, input->
simulator, NULL);
04067     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
04068     g_free (buffer);
04069     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
04070     if (input->evaluator)
04071     {
04072         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
04073         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
04074         g_free (buffer);
04075     }
04076     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
04077     switch (input->algorithm)
04078     {
04079     case ALGORITHM_MONTE_CARLO:
04080         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
04081     case ALGORITHM_SWEEP:
04082         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
04083         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
04084         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
04085         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
input->nsteps);
04086         if (input->nsteps)
04087         {
04088             gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
04089             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
04090             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
04091             switch (input->gradient_method)
04092             {
04093             case GRADIENT_METHOD_RANDOM:
04094                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
04095             }
04096         }
04097         break;
04098     default:
04099         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
04100         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
04101         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
04102         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
04103         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
04104     }
04105     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
04106     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
04107     gtk_combo_box_text_remove_all (window->combo_experiment);
04108     for (i = 0; i < input->nexperiments; ++i)
04109         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
04110     g_signal_handler_unblock

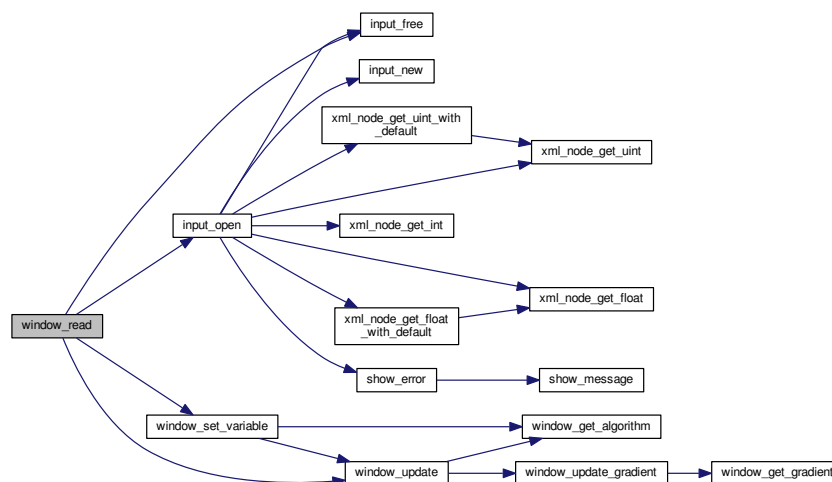
```

```

04131     (window->button_experiment, window->
id_experiment_name);
04132     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
04133     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04134     g_signal_handler_block (window->combo_variable, window->
id_variable);
04135     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
04136     gtk_combo_box_text_remove_all (window->combo_variable);
04137     for (i = 0; i < input->nvariables; ++i)
04138         gtk_combo_box_text_append_text (window->combo_variable,
input->label[i]);
04139     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
04140     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
04141     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04142     window_set_variable ();
04143     window_update ();
04144
04145 #if DEBUG
04146     fprintf (stderr, "window_read: end\n");
04147 #endif
04148     return 1;
04149 }

```

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 3010 of file [mpcotool.c](#).

```

03011 {
03012     GtkFileChooserDialog *dlg;
03013     GtkFileFilter *filter;
03014     char *buffer;
03015
03016 #if DEBUG
03017     fprintf (stderr, "window_save: start\n");
03018 #endif
03019

```

```

03020 // Opening the saving dialog
03021 dlg = (GtkFileChooserDialog *)
03022     gtk_file_chooser_dialog_new (gettext ("Save file"),
03023     window->window,
03024     GTK_FILE_CHOOSER_ACTION_SAVE,
03025     gettext ("Cancel"),
03026     GTK_RESPONSE_CANCEL,
03027     gettext ("OK"), GTK_RESPONSE_OK, NULL);
03028 gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03029 buffer = g_build_filename (input->directory, input->name, NULL);
03030 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03031 g_free (buffer);
03032
03033 // Adding XML filter
03034 filter = (GtkFileFilter *) gtk_file_filter_new ();
03035 gtk_file_filter_set_name (filter, "XML");
03036 gtk_file_filter_add_pattern (filter, "*.xml");
03037 gtk_file_filter_add_pattern (filter, "*.XML");
03038 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03039
03040 // If OK response then saving
03041 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03042 {
03043
03044     // Adding properties to the root XML node
03045     input->simulator = gtk_file_chooser_get_filename
03046         (GTK_FILE_CHOOSER (window->button_simulator));
03047     if (gtk_toggle_button_get_active
03048         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03049         input->evaluator = gtk_file_chooser_get_filename
03050             (GTK_FILE_CHOOSER (window->button_evaluator));
03051     else
03052         input->evaluator = NULL;
03053     input->result
03054         = (char *) xmlStrdup ((const xmlChar *)
03055             gtk_entry_get_text (window->entry_result));
03056     input->variables
03057         = (char *) xmlStrdup ((const xmlChar *)
03058             gtk_entry_get_text (window->entry_variables));
03059
03060     // Setting the algorithm
03061     switch (window_get_algorithm ())
03062     {
03063     case ALGORITHM_MONTE_CARLO:
03064         input->algorithm = ALGORITHM_MONTE_CARLO;
03065         input->nsimulations
03066             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03067         input->niterations
03068             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03069         input->tolerance = gtk_spin_button_get_value (window->
03070 spin_tolerance);
03071         input->nbest = gtk_spin_button_get_value_as_int (window->
03072 spin_bests);
03073         window_save_gradient ();
03074         break;
03075     case ALGORITHM_SWEEP:
03076         input->algorithm = ALGORITHM_SWEEP;
03077         input->niterations
03078             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03079         input->tolerance = gtk_spin_button_get_value (window->
03080 spin_tolerance);
03081         input->nbest = gtk_spin_button_get_value_as_int (window->
03082 spin_bests);
03083         window_save_gradient ();
03084         break;
03085     default:
03086         input->algorithm = ALGORITHM_GENETIC;
03087         input->nsimulations
03088             = gtk_spin_button_get_value_as_int (window->spin_population);
03089         input->niterations
03090             = gtk_spin_button_get_value_as_int (window->spin_generations);
03091         input->mutation_ratio
03092             = gtk_spin_button_get_value (window->spin_mutation);
03093         input->reproduction_ratio
03094             = gtk_spin_button_get_value (window->spin_reproduction);
03095         input->adaptation_ratio
03096             = gtk_spin_button_get_value (window->spin_adaptation);
03097         break;
03098     }
03099
03100     // Saving the XML file
03101     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03102     input_save (buffer);
03103
03104     // Closing and freeing memory
03105     g_free (buffer);
03106     gtk_widget_destroy (GTK_WIDGET (dlg));

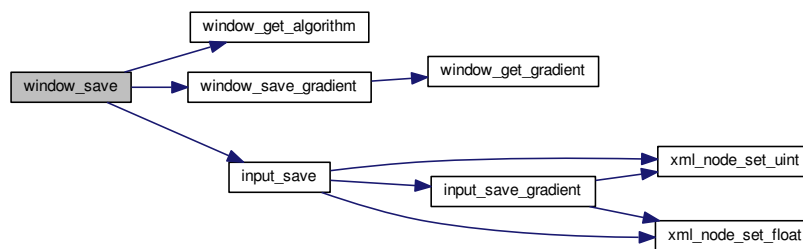
```

```

03103 #if DEBUG
03104     fprintf (stderr, "window_save: end\n");
03105 #endif
03106     return 1;
03107 }
03108
03109 // Closing and freeing memory
03110 gtk_widget_destroy (GTK_WIDGET (dlg));
03111 #if DEBUG
03112     fprintf (stderr, "window_save: end\n");
03113 #endif
03114     return 0;
03115 }

```

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 3656 of file [mpcotool.c](#).

```

03657 {
03658     unsigned int i, j;
03659     char *buffer;
03660     GFile *file1, *file2;
03661     #if DEBUG
03662         fprintf (stderr, "window_template_experiment: start\n");
03663     #endif
03664     i = (size_t) data;
03665     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03666     file1
03667         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03668     file2 = g_file_new_for_path (input->directory);
03669     buffer = g_file_get_relative_path (file2, file1);
03670     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03671     g_free (buffer);
03672     g_object_unref (file2);
03673     g_object_unref (file1);
03674     #if DEBUG
03675         fprintf (stderr, "window_template_experiment: end\n");
03676     #endif
03677 }

```

## 5.4 interface.h

```

00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burquete 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 INTERFACE__H
00037 #define INTERFACE__H 1
00038
00039 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00040
00046 typedef struct
00047 {
00048     char *template[MAX_NINPUTS];
00049     char *name;
00050     double weight;
00052 } Experiment;
00053
00058 typedef struct
00059 {
00060     char *label;
00061     double rangemin;
00062     double rangemax;
00063     double rangeminabs;
00064     double rangemaxabs;
00065     double step;
00067     unsigned int precision;
00068     unsigned int nsweeps;
00069     unsigned int nbits;
00070 } Variable;
00071
00076 typedef struct
00077 {
00078     GtkDialog *dialog;
00079     GtkGrid *grid;
00080     GtkLabel *label_seed;
00082     GtkSpinButton *spin_seed;
00084     GtkLabel *label_threads;
00085     GtkSpinButton *spin_threads;
00086     GtkLabel *label_gradient;
00087     GtkSpinButton *spin_gradient;
00088 } Options;
00089
00094 typedef struct
00095 {
00096     GtkDialog *dialog;
00097     GtkLabel *label;
00098 } Running;
00099
00104 typedef struct
00105 {
00106     GtkWindow *window;
00107     GtkGrid *grid;
00108     GtkToolbar *bar_buttons;
00109     GtkToolButton *button_open;
00110     GtkToolButton *button_save;
00111     GtkToolButton *button_run;
00112     GtkToolButton *button_options;
00113     GtkToolButton *button_help;
00114     GtkToolButton *button_about;
00115     GtkToolButton *button_exit;
00116     GtkGrid *grid_files;
00117     GtkLabel *label_simulator;
00118     GtkFileChooserButton *button_simulator;
00120     GtkCheckButton *check_evaluator;
00121     GtkFileChooserButton *button_evaluator;
00123     GtkLabel *label_result;
00124     GtkEntry *entry_result;
00125     GtkLabel *label_variables;

```

```

00126   GtkWidget *entry_variables;
00127   GtkWidget *frame_algorithm;
00128   GtkWidget *grid_algorithm;
00129   GtkWidget *button_algorithm[NALGORITHMS];
00131   GtkWidget *label_simulations;
00132   GtkWidget *spin_simulations;
00134   GtkWidget *label_iterations;
00135   GtkWidget *spin_iterations;
00137   GtkWidget *label_tolerance;
00138   GtkWidget *spin_tolerance;
00139   GtkWidget *label_bests;
00140   GtkWidget *spin_bests;
00141   GtkWidget *label_population;
00142   GtkWidget *spin_population;
00144   GtkWidget *label_generations;
00145   GtkWidget *spin_generations;
00147   GtkWidget *label_mutation;
00148   GtkWidget *spin_mutation;
00149   GtkWidget *label_reproduction;
00150   GtkWidget *spin_reproduction;
00152   GtkWidget *label_adaptation;
00153   GtkWidget *spin_adaptation;
00155   GtkWidget *check_gradient;
00157   GtkWidget *grid_gradient;
00159   GtkWidget *button_gradient[NGRADIENTS];
00161   GtkWidget *label_steps;
00162   GtkWidget *spin_steps;
00163   GtkWidget *label_estimates;
00164   GtkWidget *spin_estimates;
00166   GtkWidget *label_relaxation;
00168   GtkWidget *spin_relaxation;
00170   GtkWidget *frame_variable;
00171   GtkWidget *grid_variable;
00172   GtkWidget *comboBoxText *combo_variable;
00174   GtkWidget *button_add_variable;
00175   GtkWidget *button_remove_variable;
00176   GtkWidget *label_variable;
00177   GtkWidget *entry_variable;
00178   GtkWidget *label_min;
00179   GtkWidget *spin_min;
00180   GtkWidget *scrolled_min;
00181   GtkWidget *label_max;
00182   GtkWidget *spin_max;
00183   GtkWidget *scrolled_max;
00184   GtkWidget *check_minabs;
00185   GtkWidget *spin_minabs;
00186   GtkWidget *scrolled_minabs;
00187   GtkWidget *check_maxabs;
00188   GtkWidget *spin_maxabs;
00189   GtkWidget *scrolled_maxabs;
00190   GtkWidget *label_precision;
00191   GtkWidget *spin_precision;
00192   GtkWidget *label_sweeps;
00193   GtkWidget *spin_sweeps;
00194   GtkWidget *label_bits;
00195   GtkWidget *spin_bits;
00196   GtkWidget *label_step;
00197   GtkWidget *spin_step;
00198   GtkWidget *scrolled_step;
00199   GtkWidget *frame_experiment;
00200   GtkWidget *grid_experiment;
00201   GtkWidget *comboBoxText *combo_experiment;
00202   GtkWidget *button_add_experiment;
00203   GtkWidget *button_remove_experiment;
00204   GtkWidget *label_experiment;
00205   GtkWidget *FileChooserButton *button_experiment;
00207   GtkWidget *label_weight;
00208   GtkWidget *spin_weight;
00209   GtkWidget *check_template[MAX_NINPUTS];
00211   GtkWidget *FileChooserButton *button_template[MAX_NINPUTS];
00213   GdkPixbuf *logo;
00214   Experiment *experiment;
00215   Variable *variable;
00216   char *application_directory;
00217   gulong id_experiment;
00218   gulong id_experiment_name;
00219   gulong id_variable;
00220   gulong id_variable_label;
00221   gulong id_template[MAX_NINPUTS];
00223   gulong id_input[MAX_NINPUTS];
00225   unsigned int nexperiments;
00226   unsigned int nvariables;
00227 } Window;
00228
00229 // Public functions
00230 void input_save (char *filename);
00231 void options_new ();

```

```

00232 void running_new ();
00233 int window_get_algorithm ();
00234 int window_get_gradient ();
00235 void window_save_gradient ();
00236 int window_save ();
00237 void window_run ();
00238 void window_help ();
00239 void window_update_gradient ();
00240 void window_update ();
00241 void window_set_algorithm ();
00242 void window_set_experiment ();
00243 void window_remove_experiment ();
00244 void window_add_experiment ();
00245 void window_name_experiment ();
00246 void window_weight_experiment ();
00247 void window_inputs_experiment ();
00248 void window_template_experiment (void *data);
00249 void window_set_variable ();
00250 void window_remove_variable ();
00251 void window_add_variable ();
00252 void window_label_variable ();
00253 void window_precision_variable ();
00254 void window_rangemin_variable ();
00255 void window_rangemax_variable ();
00256 void window_rangeminabs_variable ();
00257 void window_rangemaxabs_variable ();
00258 void window_update_variable ();
00259 int window_read (char *filename);
00260 void window_open ();
00261 void window_new ();
00262 int cores_number ();
00263
00264 #endif

```

## 5.5 mpcotool.c File Reference

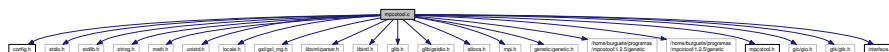
Source file of the mpcotool.

```

#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.*
- `unsigned int xml_node_get_uint_with_default (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)`  
*Function to get an unsigned integer number of a XML node property with a default value.*
- `double xml_node_get_float (xmlNode *node, const xmlChar *prop, int *error_code)`  
*Function to get a floating point number of a XML node property.*
- `double xml_node_get_float_with_default (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)`  
*Function to get a floating point number of a XML node property with a default value.*
- `void xml_node_set_int (xmlNode *node, const xmlChar *prop, int value)`  
*Function to set an integer number in a XML node property.*
- `void xml_node_set_uint (xmlNode *node, const xmlChar *prop, unsigned int value)`  
*Function to set an unsigned integer number in a XML node property.*
- `void xml_node_set_float (xmlNode *node, const xmlChar *prop, double value)`  
*Function to set a floating point number in a XML node property.*
- `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\\_step\\_variable](#) ()  
*Function to update the variable step in the main window.*
- void [window\\_update\\_variable](#) ()  
*Function to update the variable data in the main window.*
- int [window\\_read](#) (char \*filename)  
*Function to read the input data of a file.*
- void [window\\_open](#) ()  
*Function to open the input data.*
- void [window\\_new](#) ()  
*Function to open the main window.*
- 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 1443 of file [mpcotool.c](#).

```

01444 {
01445     unsigned int i, j;
01446     double e;
01447     #if DEBUG
01448         fprintf (stderr, "calibrate_best: start\n");
01449         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01450                 calibrate->nsaveds, calibrate->nbest);
01451     #endif
01452     if (calibrate->nsaveds < calibrate->nbest
01453         || value < calibrate->error_best[calibrate->nsaveds - 1])
01454     {
01455         if (calibrate->nsaveds < calibrate->nbest)
01456             ++calibrate->nsaveds;
01457         calibrate->error_best[calibrate->nsaveds - 1] = value;
01458         calibrate->simulation_best[calibrate->
01459             nsaveds - 1] = simulation;
01460         for (i = calibrate->nsaveds; --i;)
01461         {
01462             if (calibrate->error_best[i] < calibrate->
01463                 error_best[i - 1])
01464             {
01465                 j = calibrate->simulation_best[i];
01466                 e = calibrate->error_best[i];
01467                 calibrate->simulation_best[i] = calibrate->
01468                     simulation_best[i - 1];
01469                 calibrate->error_best[i] = calibrate->
01470                     error_best[i - 1];
01471                 calibrate->simulation_best[i - 1] = j;
01472                 calibrate->error_best[i - 1] = e;
01473             }
01474             else
01475                 break;
01476         }
01477     }
01478     #if DEBUG
01479         fprintf (stderr, "calibrate_best: end\n");
01480     #endif
01481 }
```

#### 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 1756 of file [mpcotool.c](#).

```

01757 {
01758     #if DEBUG
01759         fprintf (stderr, "calibrate_best_gradient: start\n");
01760         fprintf (stderr,
01761             "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01762             simulation, value, calibrate->error_best[0]);
01763     #endif
01764     if (value < calibrate->error_best[0])
01765     {
01766         calibrate->error_best[0] = value;
01767         calibrate->simulation_best[0] = simulation;
01768     #if DEBUG
01769         fprintf (stderr,
01770             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01771             simulation, value);
01772     #endif
01773     }
01774     #if DEBUG
01775         fprintf (stderr, "calibrate_best_gradient: end\n");
01776     #endif
01777 }

```

### 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 1893 of file [mpcotool.c](#).

```

01895 {
01896     double x;
01897     #if DEBUG
01898         fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01899     #endif
01900     x = calibrate->gradient[variable];
01901     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01902     {
01903         if (estimate & 1)
01904             x += calibrate->step[variable];
01905         else
01906             x -= calibrate->step[variable];
01907     }
01908     #if DEBUG
01909         fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01910             variable, x);
01911         fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01912     #endif
01913     return x;
01914 }

```

### 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 1866 of file `mpcotool.c`.

```

01868 {
01869     double x;
01870     #if DEBUG
01871     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01872     #endif
01873     x = calibrate->gradient[variable]
01874         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
01875         step[variable];
01876     #if DEBUG
01877     fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01878             variable, x);
01879     #endif
01880     return x;
01881 }

```

#### 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.

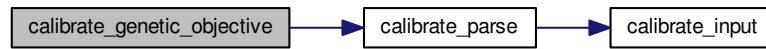
Definition at line 2059 of file `mpcotool.c`.

```

02060 {
02061     unsigned int j;
02062     double objective;
02063     char buffer[64];
02064     #if DEBUG
02065     fprintf (stderr, "calibrate_genetic_objective: start\n");
02066     #endif
02067     for (j = 0; j < calibrate->nvariables; ++j)
02068     {
02069         calibrate->value[entity->id * calibrate->nvariables + j]
02070             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02071     }
02072     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02073         objective += calibrate_parse (entity->id, j);
02074     g_mutex_lock (mutex);
02075     for (j = 0; j < calibrate->nvariables; ++j)
02076     {
02077         snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02078         fprintf (calibrate->file_variables, buffer,
02079                 genetic_get_variable (entity, calibrate->
02080                 genetic_variable + j));
02081     }
02082     fprintf (calibrate->file_variables, "%.14le\n", objective);
02083     g_mutex_unlock (mutex);
02084     #if DEBUG
02085     fprintf (stderr, "calibrate_genetic_objective: end\n");
02086     #endif
02087     return objective;
02088 }

```

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

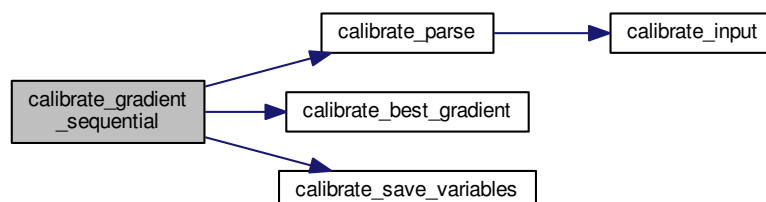
<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 1786 of file [mpcotool.c](#).

```

01787 {
01788     unsigned int i, j, k;
01789     double e;
01790     #if DEBUG
01791         fprintf (stderr, "calibrate_gradient_sequential: start\n");
01792         fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01793                 "nend_gradient=%u\n",
01794                 calibrate->nstart_gradient, calibrate->
01795                 nend_gradient);
01796     #endif
01797     for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01798     {
01799         k = simulation + i;
01800         e = 0.;
01801         for (j = 0; j < calibrate->nexperiments; ++j)
01802             e += calibrate_parse (k, j);
01803         calibrate_best_gradient (k, e);
01804         calibrate_save_variables (k, e);
01805     #if DEBUG
01806         fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01807     #endif
01808     }
01809     #if DEBUG
01810         fprintf (stderr, "calibrate_gradient_sequential: end\n");
01811     #endif
01812 }
  
```

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

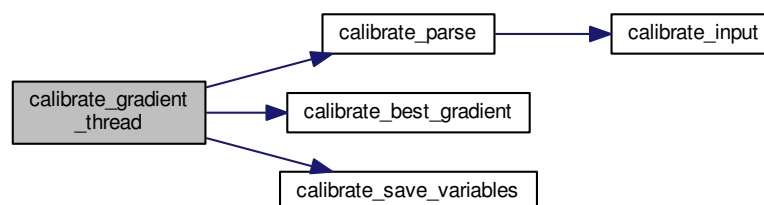
Definition at line 1821 of file [mpcotool.c](#).

```

01822 {
01823     unsigned int i, j, thread;
01824     double e;
01825     #if DEBUG
01826     fprintf (stderr, "calibrate_gradient_thread: start\n");
01827     #endif
01828     thread = data->thread;
01829     #if DEBUG
01830     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01831             thread,
01832             calibrate->thread_gradient[thread],
01833             calibrate->thread_gradient[thread + 1]);
01834     #endif
01835     for (i = calibrate->thread_gradient[thread];
01836          i < calibrate->thread_gradient[thread + 1]; ++i)
01837     {
01838         e = 0.;
01839         for (j = 0; j < calibrate->nexperiments; ++j)
01840             e += calibrate_parse (i, j);
01841         g_mutex_lock (mutex);
01842         calibrate_best_gradient (i, e);
01843         calibrate_save_variables (i, e);
01844         g_mutex_unlock (mutex);
01845     #if DEBUG
01846     fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01847     #endif
01848     }
01849     #if DEBUG
01850     fprintf (stderr, "calibrate_gradient_thread: end\n");
01851     #endif
01852     g_thread_exit (NULL);
01853     return NULL;
01854 }

```

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 1196 of file [mpcotoool.c](#).

```

01197 {
01198     unsigned int i;
01199     char buffer[32], value[32], *buffer2, *buffer3, *content;
01200     FILE *file;
01201     gsize length;
01202     GRegex *regex;
01203
01204     #if DEBUG
01205         fprintf (stderr, "calibrate_input: start\n");
01206     #endif
01207
01208     // Checking the file
01209     if (!template)
01210         goto calibrate_input_end;
01211
01212     // Opening template
01213     content = g_mapped_file_get_contents (template);
01214     length = g_mapped_file_get_length (template);
01215     #if DEBUG
01216         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01217                 content);
01218     #endif
01219     file = g_fopen (input, "w");
01220
01221     // Parsing template
01222     for (i = 0; i < calibrate->nvariables; ++i)
01223     {
01224         #if DEBUG
01225             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01226         #endif
01227         snprintf (buffer, 32, "@variable%u@", i + 1);
01228         regex = g_regex_new (buffer, 0, 0, NULL);
01229         if (i == 0)
01230         {
01231             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01232                                               calibrate->label[i], 0, NULL);
01233         }
01234         #if DEBUG
01235             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01236         #endif
01237         else
01238         {
01239             length = strlen (buffer3);
01240             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01241                                               calibrate->label[i], 0, NULL);
01242             g_free (buffer3);
01243         }
01244         g_regex_unref (regex);
01245         length = strlen (buffer2);
01246         snprintf (buffer, 32, "@value%u@", i + 1);
01247         regex = g_regex_new (buffer, 0, 0, NULL);
01248         snprintf (value, 32, format[calibrate->precision[i]],
01249                 calibrate->value[simulation * calibrate->
01250 nvariables + i]);
01251         #if DEBUG
01252             fprintf (stderr, "calibrate_input: value=%s\n", value);
01253         #endif
01254         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01255                                           0, NULL);
01256         g_free (buffer2);
01257         g_regex_unref (regex);
01258     }
01259
01260     // Saving input file
01261     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01262     g_free (buffer3);
01263     fclose (file);
01264
01265     calibrate_input_end:
01266     #if DEBUG
01267         fprintf (stderr, "calibrate_input: end\n");
01268     #endif
01269     return;
01270 }

```

### 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 1561 of file `mpcotool.c`.

```

01563 {
01564     unsigned int i, j, k, s[calibrate->nbest];
01565     double e[calibrate->nbest];
01566     #if DEBUG
01567     fprintf (stderr, "calibrate_merge: start\n");
01568     #endif
01569     i = j = k = 0;
01570     do
01571     {
01572         if (i == calibrate->nsaveds)
01573         {
01574             s[k] = simulation_best[j];
01575             e[k] = error_best[j];
01576             ++j;
01577             ++k;
01578             if (j == nsaveds)
01579                 break;
01580         }
01581         else if (j == nsaveds)
01582         {
01583             s[k] = calibrate->simulation_best[i];
01584             e[k] = calibrate->error_best[i];
01585             ++i;
01586             ++k;
01587             if (i == calibrate->nsaveds)
01588                 break;
01589         }
01590         else if (calibrate->error_best[i] > error_best[j])
01591         {
01592             s[k] = simulation_best[j];
01593             e[k] = error_best[j];
01594             ++j;
01595             ++k;
01596         }
01597         else
01598         {
01599             s[k] = calibrate->simulation_best[i];
01600             e[k] = calibrate->error_best[i];
01601             ++i;
01602             ++k;
01603         }
01604     }
01605     while (k < calibrate->nbest);
01606     calibrate->nsaveds = k;
01607     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01608     memcpy (calibrate->error_best, e, k * sizeof (double));
01609     #if DEBUG
01610     fprintf (stderr, "calibrate_merge: end\n");
01611     #endif
01612 }

```

### 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 1283 of file `mpcotool.c`.

```

01284 {
01285     unsigned int i;
01286     double e;
01287     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01288         *buffer3, *buffer4;
01289     FILE *file_result;
01290
01291     #if DEBUG
01292         fprintf (stderr, "calibrate_parse: start\n");
01293         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01294             experiment);
01295     #endif
01296
01297     // Opening input files
01298     for (i = 0; i < calibrate->ninputs; ++i)
01299     {
01300         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01301     #if DEBUG
01302         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01303     #endif
01304         calibrate_input (simulation, &input[i][0],
01305             calibrate->file[i][experiment]);
01306     }
01307     for (; i < MAX_NINPUTS; ++i)
01308         strcpy (&input[i][0], "");
01309     #if DEBUG
01310         fprintf (stderr, "calibrate_parse: parsing end\n");
01311     #endif
01312
01313     // Performing the simulation
01314     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01315     buffer2 = g_path_get_dirname (calibrate->simulator);
01316     buffer3 = g_path_get_basename (calibrate->simulator);
01317     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01318     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
01319         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01320         input[6], input[7], output);
01321     g_free (buffer4);
01322     g_free (buffer3);
01323     g_free (buffer2);
01324     #if DEBUG
01325         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01326     #endif
01327     system (buffer);
01328
01329     // Checking the objective value function
01330     if (calibrate->evaluator)
01331     {
01332         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01333         buffer2 = g_path_get_dirname (calibrate->evaluator);
01334         buffer3 = g_path_get_basename (calibrate->evaluator);
01335         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01336         snprintf (buffer, 512, "%s\ " %s %s %s",
01337             buffer4, output, calibrate->experiment[experiment], result);
01338         g_free (buffer4);
01339         g_free (buffer3);
01340         g_free (buffer2);
01341     #if DEBUG
01342         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01343     #endif
01344         system (buffer);
01345         file_result = g_fopen (result, "r");
01346         e = atof (fgets (buffer, 512, file_result));
01347         fclose (file_result);
01348     }
01349     else
01350     {
01351         strcpy (result, "");
01352         file_result = g_fopen (output, "r");
01353         e = atof (fgets (buffer, 512, file_result));
01354         fclose (file_result);
01355     }
01356
01357     // Removing files
01358     #if !DEBUG
01359     for (i = 0; i < calibrate->ninputs; ++i)
01360     {
01361         if (calibrate->file[i][0])
01362         {

```

```

01363         snprintf (buffer, 512, RM " %s", &input[i][0]);
01364         system (buffer);
01365     }
01366 }
01367     snprintf (buffer, 512, RM " %s %s", output, result);
01368     system (buffer);
01369 #endif
01370
01371 #if DEBUG
01372     fprintf (stderr, "calibrate_parse: end\n");
01373 #endif
01374
01375 // Returning the objective function
01376 return e * calibrate->weight[experiment];
01377 }

```

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 1415 of file [mpcotool.c](#).

```

01416 {
01417     unsigned int i;
01418     char buffer[64];
01419     #if DEBUG
01420         fprintf (stderr, "calibrate_save_variables: start\n");
01421     #endif
01422     for (i = 0; i < calibrate->nvariables; ++i)
01423     {
01424         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01425         fprintf (calibrate->file_variables, buffer,
01426                 calibrate->value[simulation * calibrate->
01427                               nvariables + i]);
01428     }
01429     fprintf (calibrate->file_variables, "%.14le\n", error);
01430     #if DEBUG
01431         fprintf (stderr, "calibrate_save_variables: end\n");
01432     #endif
01433 }

```

#### 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.
-------------------	--------------------

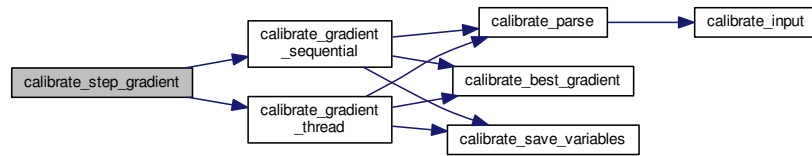
Definition at line 1923 of file `mpcotool.c`.

```

01924 {
01925     GThread *thread[nthreads_gradient];
01926     ParallelData data[nthreads_gradient];
01927     unsigned int i, j, k, b;
01928     #if DEBUG
01929         fprintf (stderr, "calibrate_step_gradient: start\n");
01930     #endif
01931     for (i = 0; i < calibrate->nestimates; ++i)
01932     {
01933         k = (simulation + i) * calibrate->nvariables;
01934         b = calibrate->simulation_best[0] * calibrate->
nvariables;
01935     #if DEBUG
01936         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01937                 simulation + i, calibrate->simulation_best[0]);
01938     #endif
01939         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01940         {
01941             #if DEBUG
01942                 fprintf (stderr,
01943                         "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01944                         i, j, calibrate->value[b]);
01945             #endif
01946             calibrate->value[k]
01947                 = calibrate->value[b] + calibrate_estimate_gradient (j
, i);
01948             calibrate->value[k] = fmin (fmax (calibrate->
value[k],
01949                                             calibrate->rangeminabs[j]),
01950                                         calibrate->rangemaxabs[j]);
01951             #if DEBUG
01952                 fprintf (stderr,
01953                         "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01954                         i, j, calibrate->value[k]);
01955             #endif
01956         }
01957     }
01958     if (nthreads_gradient == 1)
01959         calibrate_gradient_sequential (simulation);
01960     else
01961     {
01962         for (i = 0; i <= nthreads_gradient; ++i)
01963         {
01964             calibrate->thread_gradient[i]
01965                 = simulation + calibrate->nstart_gradient
01966                 + i * (calibrate->nend_gradient - calibrate->
nstart_gradient)
01967                 / nthreads_gradient;
01968             #if DEBUG
01969                 fprintf (stderr,
01970                         "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01971                         i, calibrate->thread_gradient[i]);
01972             #endif
01973         }
01974         for (i = 0; i < nthreads_gradient; ++i)
01975         {
01976             data[i].thread = i;
01977             thread[i] = g_thread_new
01978                 (NULL, (void (*) ) calibrate_gradient_thread, &data[i]);
01979         }
01980         for (i = 0; i < nthreads_gradient; ++i)
01981             g_thread_join (thread[i]);
01982     }
01983     #if DEBUG
01984         fprintf (stderr, "calibrate_step_gradient: end\n");
01985     #endif
01986 }

```

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

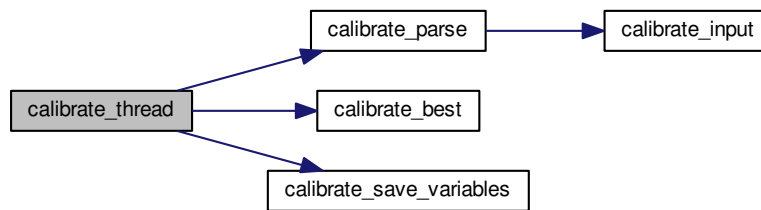
Definition at line 1517 of file [mpcotool.c](#).

```

01518 {
01519     unsigned int i, j, thread;
01520     double e;
01521     #if DEBUG
01522         fprintf (stderr, "calibrate_thread: start\n");
01523     #endif
01524     thread = data->thread;
01525     #if DEBUG
01526         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01527                 calibrate->thread[thread], calibrate->thread[thread + 1]);
01528     #endif
01529     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01530     {
01531         e = 0.;
01532         for (j = 0; j < calibrate->nexperiments; ++j)
01533             e += calibrate_parse (i, j);
01534         g_mutex_lock (mutex);
01535         calibrate_best (i, e);
01536         calibrate_save_variables (i, e);
01537         g_mutex_unlock (mutex);
01538     #if DEBUG
01539         fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01540     #endif
01541     }
01542     #if DEBUG
01543         fprintf (stderr, "calibrate_thread: end\n");
01544     #endif
01545     g_thread_exit (NULL);
01546     return NULL;
01547 }

```

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 [4875](#) of file [mpcotool.c](#).

```

04876 {
04877     #ifdef G_OS_WIN32
04878         SYSTEM_INFO sysinfo;
04879         GetSystemInfo (&sysinfo);
04880         return sysinfo.dwNumberOfProcessors;
04881     #else
04882         return (int) sysconf ( _SC_NPROCESSORS_ONLN );
04883     #endif
04884 }
  
```

#### 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 [548](#) of file [mpcotool.c](#).

```

00549 {
00550     char buffer2[64];
00551     char *buffert[MAX_NINPUTS] =
00552         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00553     xmlDoc *doc;
00554     xmlNode *node, *child;
00555     xmlChar *buffer;
00556     char *msg;
00557     int error_code;
00558     unsigned int i;
00559
00560     #if DEBUG
00561         fprintf (stderr, "input_open: start\n");
00562     #endif
  
```



```

00563
00564 // Resetting input data
00565 buffer = NULL;
00566 input_new ();
00567
00568 // Parsing the input file
00569 #if DEBUG
00570 fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00571 #endif
00572 doc = xmlParseFile (filename);
00573 if (!doc)
00574 {
00575     msg = gettext ("Unable to parse the input file");
00576     goto exit_on_error;
00577 }
00578
00579 // Getting the root node
00580 #if DEBUG
00581 fprintf (stderr, "input_open: getting the root node\n");
00582 #endif
00583 node = xmlDocGetRootElement (doc);
00584 if (xmlStrcmp (node->name, XML_CALIBRATE))
00585 {
00586     msg = gettext ("Bad root XML node");
00587     goto exit_on_error;
00588 }
00589
00590 // Getting results file names
00591 input->result = (char *) xmlGetProp (node, XML_RESULT);
00592 if (!input->result)
00593     input->result = (char *) xmlStrdup (result_name);
00594 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00595 if (!input->variables)
00596     input->variables = (char *) xmlStrdup (variables_name);
00597
00598 // Opening simulator program name
00599 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00600 if (!input->simulator)
00601 {
00602     msg = gettext ("Bad simulator program");
00603     goto exit_on_error;
00604 }
00605
00606 // Opening evaluator program name
00607 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00608
00609 // Obtaining pseudo-random numbers generator seed
00610 input->seed
00611     = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00612                                     &error_code);
00613 if (error_code)
00614 {
00615     msg = gettext ("Bad pseudo-random numbers generator seed");
00616     goto exit_on_error;
00617 }
00618
00619 // Opening algorithm
00620 buffer = xmlGetProp (node, XML_ALGORITHM);
00621 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00622 {
00623     input->algorithm = ALGORITHM_MONTE_CARLO;
00624
00625     // Obtaining simulations number
00626     input->nsimulations
00627         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00628     if (error_code)
00629     {
00630         msg = gettext ("Bad simulations number");
00631         goto exit_on_error;
00632     }
00633 }
00634 else if (!xmlStrcmp (buffer, XML_SWEEP))
00635     input->algorithm = ALGORITHM_SWEEP;
00636 else if (!xmlStrcmp (buffer, XML_GENETIC))
00637 {
00638     input->algorithm = ALGORITHM_GENETIC;
00639
00640     // Obtaining population
00641     if (xmlHasProp (node, XML_NPOPULATION))
00642     {
00643         input->nsimulations
00644             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00645         if (error_code || input->nsimulations < 3)
00646         {
00647             msg = gettext ("Invalid population number");
00648             goto exit_on_error;

```

```

00649         }
00650     }
00651     else
00652     {
00653         msg = gettext ("No population number");
00654         goto exit_on_error;
00655     }
00656
00657     // Obtaining generations
00658     if (xmlHasProp (node, XML_NGENERATIONS))
00659     {
00660         input->niterations
00661         = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00662         if (error_code || !input->niterations)
00663         {
00664             msg = gettext ("Invalid generations number");
00665             goto exit_on_error;
00666         }
00667     }
00668     else
00669     {
00670         msg = gettext ("No generations number");
00671         goto exit_on_error;
00672     }
00673
00674     // Obtaining mutation probability
00675     if (xmlHasProp (node, XML_MUTATION))
00676     {
00677         input->mutation_ratio
00678         = xml_node_get_float (node, XML_MUTATION, &error_code);
00679         if (error_code || input->mutation_ratio < 0.
00680             || input->mutation_ratio >= 1.)
00681         {
00682             msg = gettext ("Invalid mutation probability");
00683             goto exit_on_error;
00684         }
00685     }
00686     else
00687     {
00688         msg = gettext ("No mutation probability");
00689         goto exit_on_error;
00690     }
00691
00692     // Obtaining reproduction probability
00693     if (xmlHasProp (node, XML_REPRODUCTION))
00694     {
00695         input->reproduction_ratio
00696         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00697         if (error_code || input->reproduction_ratio < 0.
00698             || input->reproduction_ratio >= 1.0)
00699         {
00700             msg = gettext ("Invalid reproduction probability");
00701             goto exit_on_error;
00702         }
00703     }
00704     else
00705     {
00706         msg = gettext ("No reproduction probability");
00707         goto exit_on_error;
00708     }
00709
00710     // Obtaining adaptation probability
00711     if (xmlHasProp (node, XML_ADAPTATION))
00712     {
00713         input->adaptation_ratio
00714         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00715         if (error_code || input->adaptation_ratio < 0.
00716             || input->adaptation_ratio >= 1.)
00717         {
00718             msg = gettext ("Invalid adaptation probability");
00719             goto exit_on_error;
00720         }
00721     }
00722     else
00723     {
00724         msg = gettext ("No adaptation probability");
00725         goto exit_on_error;
00726     }
00727
00728     // Checking survivals
00729     i = input->mutation_ratio * input->nsimulations;
00730     i += input->reproduction_ratio * input->
00731     nsimulations;
00732     i += input->adaptation_ratio * input->
00733     nsimulations;
00734     if (i > input->nsimulations - 2)
00735     {

```

```

00734         msg = gettext
00735             ("No enough survival entities to reproduce the population");
00736         goto exit_on_error;
00737     }
00738 }
00739 else
00740 {
00741     msg = gettext ("Unknown algorithm");
00742     goto exit_on_error;
00743 }
00744 xmlFree (buffer);
00745 buffer = NULL;
00746
00747 if (input->algorithm == ALGORITHM_MONTE_CARLO
00748     || input->algorithm == ALGORITHM_SWEEP)
00749 {
00750     // Obtaining iterations number
00751     input->niterations
00752     = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00753     if (error_code == 1)
00754         input->niterations = 1;
00755     else if (error_code)
00756     {
00757         msg = gettext ("Bad iterations number");
00758         goto exit_on_error;
00759     }
00760
00761     // Obtaining best number
00762     input->nbest
00763     = xml_node_get_uint_with_default (node,
00764 XML_NBEST, 1, &error_code);
00765     if (error_code || !input->nbest)
00766     {
00767         msg = gettext ("Invalid best number");
00768         goto exit_on_error;
00769     }
00770
00771     // Obtaining tolerance
00772     input->tolerance
00773     = xml_node_get_float_with_default (node,
00774 XML_TOLERANCE, 0.,
00775                                     &error_code);
00776     if (error_code || input->tolerance < 0.)
00777     {
00778         msg = gettext ("Invalid tolerance");
00779         goto exit_on_error;
00780     }
00781
00782     // Getting gradient method parameters
00783     if (xmlHasProp (node, XML_NSTEPS))
00784     {
00785         input->nsteps = xml_node_get_uint (node,
00786 XML_NSTEPS, &error_code);
00787         if (error_code || !input->nsteps)
00788         {
00789             msg = gettext ("Invalid steps number");
00790             goto exit_on_error;
00791         }
00792         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00793         if (!xmlStrcmp (buffer, XML_COORDINATES))
00794             input->gradient_method =
00795 GRADIENT_METHOD_COORDINATES;
00796         else if (!xmlStrcmp (buffer, XML_RANDOM))
00797         {
00798             input->gradient_method =
00799 GRADIENT_METHOD_RANDOM;
00800             input->nestimates
00801             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00802             if (error_code || !input->nestimates)
00803             {
00804                 msg = gettext ("Invalid estimates number");
00805                 goto exit_on_error;
00806             }
00807         }
00808         else
00809         {
00810             msg = gettext ("Unknown method to estimate the gradient");
00811             goto exit_on_error;
00812         }
00813         xmlFree (buffer);
00814         buffer = NULL;
00815         input->relaxation
00816         = xml_node_get_float_with_default (node,
00817 XML_RELAXATION,
00818                                     DEFAULT_RELAXATION, &error_code);
00819         if (error_code || input->relaxation < 0. || input->

```

```

        relaxation > 2.)
00815     {
00816         msg = gettext ("Invalid relaxation parameter");
00817         goto exit_on_error;
00818     }
00819 }
00820 else
00821     input->nsteps = 0;
00822 }
00823
00824 // Reading the experimental data
00825 for (child = node->children; child; child = child->next)
00826 {
00827     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00828         break;
00829 #if DEBUG
00830     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00831 #endif
00832     if (xmlHasProp (child, XML_NAME))
00833         buffer = xmlGetProp (child, XML_NAME);
00834     else
00835     {
00836         snprintf (buffer2, 64, "%s %u: %s",
00837             gettext ("Experiment"),
00838             input->nexperiments + 1, gettext ("no data file name"));
00839         msg = buffer2;
00840         goto exit_on_error;
00841     }
00842 #if DEBUG
00843     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00844 #endif
00845     input->weight = g_realloc (input->weight,
00846         (1 + input->nexperiments) * sizeof (double));
00847     input->weight[input->nexperiments]
00848         = xml_node_get_float_with_default (child,
00849         XML_WEIGHT, 1., &error_code);
00850     if (error_code)
00851     {
00852         snprintf (buffer2, 64, "%s %s: %s",
00853             gettext ("Experiment"), buffer, gettext ("bad weight"));
00854         msg = buffer2;
00855         goto exit_on_error;
00856     }
00857 #if DEBUG
00858     fprintf (stderr, "input_open: weight=%lg\n",
00859         input->weight[input->nexperiments]);
00860 #endif
00861     if (!input->nexperiments)
00862         input->ninputs = 0;
00863 #if DEBUG
00864     fprintf (stderr, "input_open: template[0]\n");
00865 #endif
00866     if (xmlHasProp (child, XML_TEMPLATE1))
00867     {
00868         input->template[0]
00869             = (char **) g_realloc (input->template[0],
00870             (1 + input->nexperiments) * sizeof (char *));
00871         buffert[0] = (char *) xmlGetProp (child, template[0]);
00872 #if DEBUG
00873         fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00874             input->nexperiments, buffert[0]);
00875 #endif
00876         if (!input->nexperiments)
00877             ++input->ninputs;
00878 #if DEBUG
00879         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00880 #endif
00881     }
00882     else
00883     {
00884         snprintf (buffer2, 64, "%s %s: %s",
00885             gettext ("Experiment"), buffer, gettext ("no template"));
00886         msg = buffer2;
00887         goto exit_on_error;
00888     }
00889     for (i = 1; i < MAX_NINPUTS; ++i)
00890     {
00891 #if DEBUG
00892         fprintf (stderr, "input_open: template%u\n", i + 1);
00893 #endif
00894         if (xmlHasProp (child, template[i]))
00895         {
00896             if (input->nexperiments && input->ninputs <= i)
00897             {
00898                 snprintf (buffer2, 64, "%s %s: %s",
00899                     gettext ("Experiment"),
00900                     buffer, gettext ("bad templates number"));

```

```

00900         msg = buffer2;
00901         while (i-- > 0)
00902             xmlFree (buffert[i]);
00903         goto exit_on_error;
00904     }
00905     input->template[i] = (char **)
00906         g_realloc (input->template[i],
00907             (1 + input->nexperiments) * sizeof (char *));
00908     buffert[i] = (char *) xmlGetProp (child, template[i]);
00909 #if DEBUG
00910     fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00911         input->nexperiments, i + 1,
00912         input->template[i][input->nexperiments]);
00913 #endif
00914     if (!input->nexperiments)
00915         ++input->ninputs;
00916 #if DEBUG
00917     fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00918 #endif
00919 }
00920 else if (input->nexperiments && input->ninputs > i)
00921 {
00922     snprintf (buffer2, 64, "%s %s: %s",
00923         gettext ("Experiment"),
00924         buffer, gettext ("no template"), i + 1);
00925     msg = buffer2;
00926     while (i-- > 0)
00927         xmlFree (buffert[i]);
00928     goto exit_on_error;
00929 }
00930 else
00931     break;
00932 }
00933 input->experiment
00934     = g_realloc (input->experiment,
00935         (1 + input->nexperiments) * sizeof (char *));
00936 input->experiment[input->nexperiments] = (char *) buffer;
00937 for (i = 0; i < input->ninputs; ++i)
00938     input->template[i][input->nexperiments] = buffert[i];
00939 ++input->nexperiments;
00940 #if DEBUG
00941     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00942 #endif
00943 }
00944 if (!input->nexperiments)
00945 {
00946     msg = gettext ("No calibration experiments");
00947     goto exit_on_error;
00948 }
00949 buffer = NULL;
00950
00951 // Reading the variables data
00952 for (; child; child = child->next)
00953 {
00954     if (xmlStrcmp (child->name, XML_VARIABLE))
00955     {
00956         snprintf (buffer2, 64, "%s %u: %s",
00957             gettext ("Variable"),
00958             input->nvariables + 1, gettext ("bad XML node"));
00959         msg = buffer2;
00960         goto exit_on_error;
00961     }
00962     if (xmlHasProp (child, XML_NAME))
00963         buffer = xmlGetProp (child, XML_NAME);
00964     else
00965     {
00966         snprintf (buffer2, 64, "%s %u: %s",
00967             gettext ("Variable"),
00968             input->nvariables + 1, gettext ("no name"));
00969         msg = buffer2;
00970         goto exit_on_error;
00971     }
00972     if (xmlHasProp (child, XML_MINIMUM))
00973     {
00974         input->rangemin = g_realloc
00975             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00976         input->rangeminabs = g_realloc
00977             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00978         input->rangemin[input->nvariables]
00979             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00980         if (error_code)
00981         {
00982             snprintf (buffer2, 64, "%s %s: %s",
00983                 gettext ("Variable"), buffer, gettext ("bad minimum"));
00984             msg = buffer2;
00985             goto exit_on_error;
00986         }

```

```

00987         input->rangeminabs[input->nvariables]
00988         = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
00989                                         -G_MAXDOUBLE, &error_code);
00990         if (error_code)
00991         {
00992             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00993                     gettext ("bad absolute minimum"));
00994             msg = buffer2;
00995             goto exit_on_error;
00996         }
00997         if (input->rangemin[input->nvariables]
00998             < input->rangeminabs[input->nvariables])
00999         {
01000             snprintf (buffer2, 64, "%s %s: %s",
01001                     gettext ("Variable"),
01002                     buffer, gettext ("minimum range not allowed"));
01003             msg = buffer2;
01004             goto exit_on_error;
01005         }
01006     }
01007     else
01008     {
01009         snprintf (buffer2, 64, "%s %s: %s",
01010                 gettext ("Variable"), buffer, gettext ("no minimum range"));
01011         msg = buffer2;
01012         goto exit_on_error;
01013     }
01014     if (xmlHasProp (child, XML_MAXIMUM))
01015     {
01016         input->rangemax = g_realloc
01017             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01018         input->rangemaxabs = g_realloc
01019             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01020         input->rangemax[input->nvariables]
01021         = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01022         if (error_code)
01023         {
01024             snprintf (buffer2, 64, "%s %s: %s",
01025                     gettext ("Variable"), buffer, gettext ("bad maximum"));
01026             msg = buffer2;
01027             goto exit_on_error;
01028         }
01029         input->rangemaxabs[input->nvariables]
01030         = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
01031                                         G_MAXDOUBLE, &error_code);
01032         if (error_code)
01033         {
01034             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01035                     gettext ("bad absolute maximum"));
01036             msg = buffer2;
01037             goto exit_on_error;
01038         }
01039         if (input->rangemax[input->nvariables]
01040             > input->rangemaxabs[input->nvariables])
01041         {
01042             snprintf (buffer2, 64, "%s %s: %s",
01043                     gettext ("Variable"),
01044                     buffer, gettext ("maximum range not allowed"));
01045             msg = buffer2;
01046             goto exit_on_error;
01047         }
01048     }
01049     else
01050     {
01051         snprintf (buffer2, 64, "%s %s: %s",
01052                 gettext ("Variable"), buffer, gettext ("no maximum range"));
01053         msg = buffer2;
01054         goto exit_on_error;
01055     }
01056     if (input->rangemax[input->nvariables]
01057         < input->rangemin[input->nvariables])
01058     {
01059         snprintf (buffer2, 64, "%s %s: %s",
01060                 gettext ("Variable"), buffer, gettext ("bad range"));
01061         msg = buffer2;
01062         goto exit_on_error;
01063     }
01064     input->precision = g_realloc
01065         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01066     input->precision[input->nvariables]
01067     = xml_node_get_uint_with_default (child,
XML_PRECISION,
01068                                     DEFAULT_PRECISION, &error_code);
01069     if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)

```

```

01070     {
01071         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01072                 gettext ("bad precision"));
01073         msg = buffer2;
01074         goto exit_on_error;
01075     }
01076     if (input->algorithm == ALGORITHM_SWEEP)
01077     {
01078         if (xmlHasProp (child, XML_NSWEEPS))
01079         {
01080             input->nsweeps = (unsigned int *)
01081                 g_realloc (input->nsweeps,
01082                           (1 + input->nvariables) * sizeof (unsigned int));
01083             input->nsweeps[input->nvariables]
01084                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01085             if (error_code || !input->nsweeps[input->
01086         nvariables])
01087             {
01088                 snprintf (buffer2, 64, "%s %s: %s",
01089                         gettext ("Variable"),
01090                         buffer, gettext ("bad sweeps"));
01091                 msg = buffer2;
01092                 goto exit_on_error;
01093             }
01094         }
01095         else
01096         {
01097             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                     gettext ("no sweeps number"));
01099             msg = buffer2;
01100             goto exit_on_error;
01101         }
01102     }
01103     #if DEBUG
01104     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01105             input->nsweeps[input->nvariables],
01106             input->nsimulations);
01107     #endif
01108     if (input->algorithm == ALGORITHM_GENETIC)
01109     {
01110         // Obtaining bits representing each variable
01111         if (xmlHasProp (child, XML_NBITS))
01112         {
01113             input->nbits = (unsigned int *)
01114                 g_realloc (input->nbits,
01115                           (1 + input->nvariables) * sizeof (unsigned int));
01116             i = xml_node_get_uint (child, XML_NBITS, &error_code);
01117             if (error_code || !i)
01118             {
01119                 snprintf (buffer2, 64, "%s %s: %s",
01120                         gettext ("Variable"),
01121                         buffer, gettext ("invalid bits number"));
01122                 msg = buffer2;
01123                 goto exit_on_error;
01124             }
01125             input->nbits[input->nvariables] = i;
01126         }
01127         else
01128         {
01129             snprintf (buffer2, 64, "%s %s: %s",
01130                     gettext ("Variable"),
01131                     buffer, gettext ("no bits number"));
01132             msg = buffer2;
01133             goto exit_on_error;
01134         }
01135     }
01136     else if (input->nsteps)
01137     {
01138         input->step = (double *)
01139             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01140         input->step[input->nvariables]
01141             = xml_node_get_float (child, XML_STEP, &error_code);
01142         if (error_code || input->step[input->nvariables] < 0.)
01143         {
01144             snprintf (buffer2, 64, "%s %s: %s",
01145                     gettext ("Variable"),
01146                     buffer, gettext ("bad step size"));
01147             msg = buffer2;
01148             goto exit_on_error;
01149         }
01150     }
01151     input->label = g_realloc
01152         (input->label, (1 + input->nvariables) * sizeof (char *));
01153     input->label[input->nvariables] = (char *) buffer;
01154     ++input->nvariables;
01155 }
01156 if (!input->nvariables)

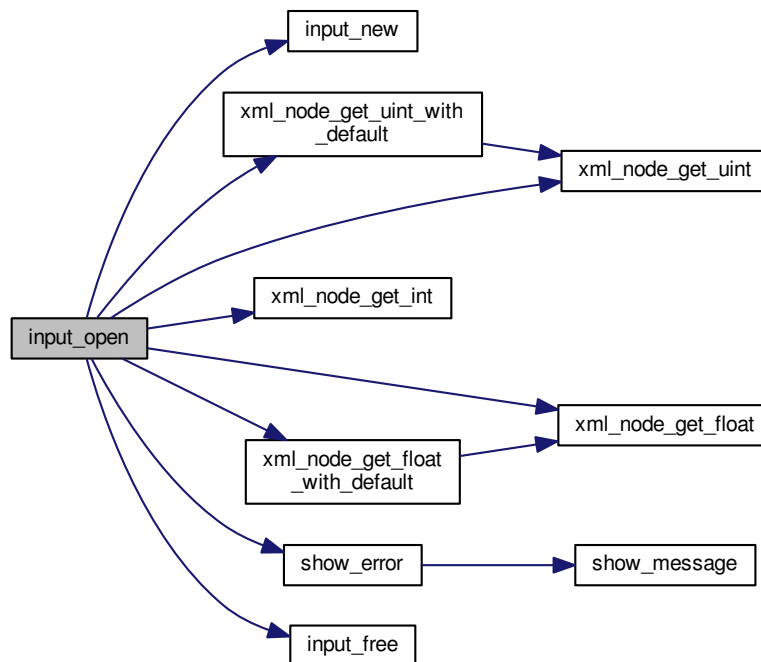
```

```

01155     {
01156         msg = gettext ("No calibration variables");
01157         goto exit_on_error;
01158     }
01159     buffer = NULL;
01160
01161     // Getting the working directory
01162     input->directory = g_path_get_dirname (filename);
01163     input->name = g_path_get_basename (filename);
01164
01165     // Closing the XML document
01166     xmlFreeDoc (doc);
01167
01168     #if DEBUG
01169     fprintf (stderr, "input_open: end\n");
01170     #endif
01171     return 1;
01172
01173 exit_on_error:
01174     xmlFree (buffer);
01175     xmlFreeDoc (doc);
01176     show_error (msg);
01177     input_free ();
01178     #if DEBUG
01179     fprintf (stderr, "input_open: end\n");
01180     #endif
01181     return 0;
01182 }

```

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 2699 of file mpcotool.c.

```

02700 {
02701     unsigned int i, j;
02702     char *buffer;
02703     xmlDoc *doc;
02704     xmlNode *node, *child;
02705     GFile *file, *file2;
02706
02707     #if DEBUG
02708         fprintf (stderr, "input_save: start\n");
02709     #endif
02710
02711     // Getting the input file directory
02712     input->name = g_path_get_basename (filename);
02713     input->directory = g_path_get_dirname (filename);
02714     file = g_file_new_for_path (input->directory);
02715
02716     // Opening the input file
02717     doc = xmlNewDoc ((const xmlChar *) "1.0");
02718
02719     // Setting root XML node
02720     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02721     xmlDocSetRootElement (doc, node);
02722
02723     // Adding properties to the root XML node
02724     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02725         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02726     if (xmlStrcmp ((const xmlChar *) input->variables,
02727         variables_name))
02727         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02728         variables);
02728     file2 = g_file_new_for_path (input->simulator);
02729     buffer = g_file_get_relative_path (file, file2);
02730     g_object_unref (file2);
02731     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02732     g_free (buffer);
02733     if (input->evaluator)
02734     {
02735         file2 = g_file_new_for_path (input->evaluator);
02736         buffer = g_file_get_relative_path (file, file2);
02737         g_object_unref (file2);
02738         if (xmlStrlen ((xmlChar *) buffer))
02739             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02740         g_free (buffer);
02741     }
02742     if (input->seed != DEFAULT_RANDOM_SEED)
02743         xml_node_set_uint (node, XML_SEED, input->seed);
02744
02745     // Setting the algorithm
02746     buffer = (char *) g_malloc (64);
02747     switch (input->algorithm)
02748     {
02749     case ALGORITHM_MONTE_CARLO:
02750         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02751         snprintf (buffer, 64, "%u", input->nsimulations);
02752         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02753         snprintf (buffer, 64, "%u", input->niterations);
02754         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02755         snprintf (buffer, 64, "%.3lg", input->tolerance);
02756         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02757         snprintf (buffer, 64, "%u", input->nbest);
02758         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02759         input_save_gradient (node);
02760         break;
02761     case ALGORITHM_SWEEP:
02762         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02763         snprintf (buffer, 64, "%u", input->niterations);
02764         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02765         snprintf (buffer, 64, "%.3lg", input->tolerance);
02766         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02767         snprintf (buffer, 64, "%u", input->nbest);
02768         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02769         input_save_gradient (node);
02770         break;
02771     default:
02772         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02773         snprintf (buffer, 64, "%u", input->nsimulations);
02774         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02775         snprintf (buffer, 64, "%u", input->niterations);
02776         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02777         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);

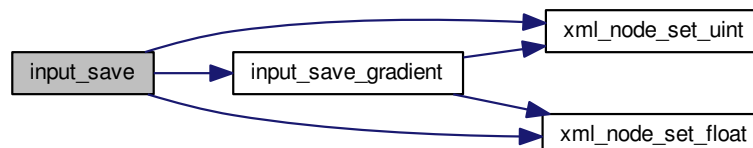
```

```

02778     xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02779     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02780     xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02781     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02782     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02783     break;
02784 }
02785 g_free (buffer);
02786
02787 // Setting the experimental data
02788 for (i = 0; i < input->nexperiments; ++i)
02789 {
02790     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02791     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02792     if (input->weight[i] != 1.)
02793         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02794     for (j = 0; j < input->ninputs; ++j)
02795         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02796 }
02797
02798 // Setting the variables data
02799 for (i = 0; i < input->nvariables; ++i)
02800 {
02801     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02802     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02803     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02804     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02805         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02806     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02807     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02808         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02809     if (input->precision[i] != DEFAULT_PRECISION)
02810         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02811     if (input->algorithm == ALGORITHM_SWEEP)
02812         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02813     else if (input->algorithm == ALGORITHM_GENETIC)
02814         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02815     if (input->nsteps)
02816         xml_node_set_float (child, XML_STEP, input->
step[i]);
02817 }
02818
02819 // Saving the XML file
02820 xmlSaveFormatFile (filename, doc, 1);
02821
02822 // Freeing memory
02823 xmlFreeDoc (doc);
02824
02825 #if DEBUG
02826 fprintf (stderr, "input_save: end\n");
02827 #endif
02828 }

```

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

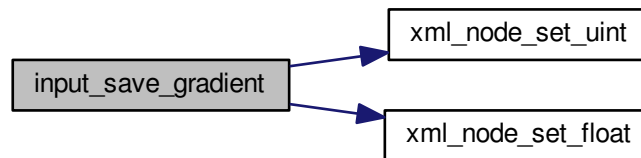
Definition at line 2667 of file [mpcotool.c](#).

```

02668 {
02669 #if DEBUG
02670     fprintf (stderr, "input_save_gradient: start\n");
02671 #endif
02672     if (input->nsteps)
02673     {
02674         xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
02675         if (input->relaxation != DEFAULT_RELAXATION)
02676             xml_node_set_float (node, XML_RELAXATION,
input->relaxation);
02677         switch (input->gradient_method)
02678         {
02679             case GRADIENT_METHOD_COORDINATES:
02680                 xmlSetProp (node, XML_GRADIENT_METHOD,
XML_COORDINATES);
02681                 break;
02682             default:
02683                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02684                 xml_node_set_uint (node, XML_NESTIMATES,
input->nestimates);
02685         }
02686     }
02687 #if DEBUG
02688     fprintf (stderr, "input_save_gradient: end\n");
02689 #endif
02690 }

```

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 4896 of file [mpcotool.c](#).

```

04897 {
04898 #if HAVE_GTK
04899     char *buffer;

```

```

04900 #endif
04901
04902 // Starting pseudo-random numbers generator
04903 calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04904 calibrate->seed = DEFAULT_RANDOM_SEED;
04905
04906 // Allowing spaces in the XML data file
04907 xmlKeepBlanksDefault (0);
04908
04909 // Starting MPI
04910 #if HAVE_MPI
04911 MPI_Init (&argn, &argc);
04912 MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04913 MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04914 printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04915 #else
04916 ntasks = 1;
04917 #endif
04918
04919 #if HAVE_GTK
04920
04921 // Getting threads number
04922 nthreads_gradient = nthreads = cores_number ();
04923
04924 // Setting local language and international floating point numbers notation
04925 setlocale (LC_ALL, "");
04926 setlocale (LC_NUMERIC, "C");
04927 window->application_directory = g_get_current_dir ();
04928 buffer = g_build_filename (window->application_directory,
    LOCALE_DIR, NULL);
04929 bindtextdomain (PROGRAM_INTERFACE, buffer);
04930 bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04931 textdomain (PROGRAM_INTERFACE);
04932
04933 // Initing GTK+
04934 gtk_disable_setlocale ();
04935 gtk_init (&argn, &argc);
04936
04937 // Opening the main window
04938 window_new ();
04939 gtk_main ();
04940
04941 // Freeing memory
04942 input_free ();
04943 g_free (buffer);
04944 gtk_widget_destroy (GTK_WIDGET (window->window));
04945 g_free (window->application_directory);
04946
04947 #else
04948
04949 // Checking syntax
04950 if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04951 {
04952     printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04953     return 1;
04954 }
04955
04956 // Getting threads number
04957 if (argn == 2)
04958     nthreads_gradient = nthreads = cores_number ();
04959 else
04960 {
04961     nthreads_gradient = nthreads = atoi (argc[2]);
04962     if (!nthreads)
04963     {
04964         printf ("Bad threads number\n");
04965         return 2;
04966     }
04967 }
04968 printf ("nthreads=%u\n", nthreads);
04969
04970 // Making calibration
04971 if (input_open (argc[argn - 1]))
04972     calibrate_open ();
04973
04974 // Freeing memory
04975 calibrate_free ();
04976 #endif
04977
04978 // Closing MPI
04979 #if HAVE_MPI
04980 MPI_Finalize ();
04981 #endif
04982
04983 // Freeing memory
04984 gsl_rng_free (calibrate->rng);

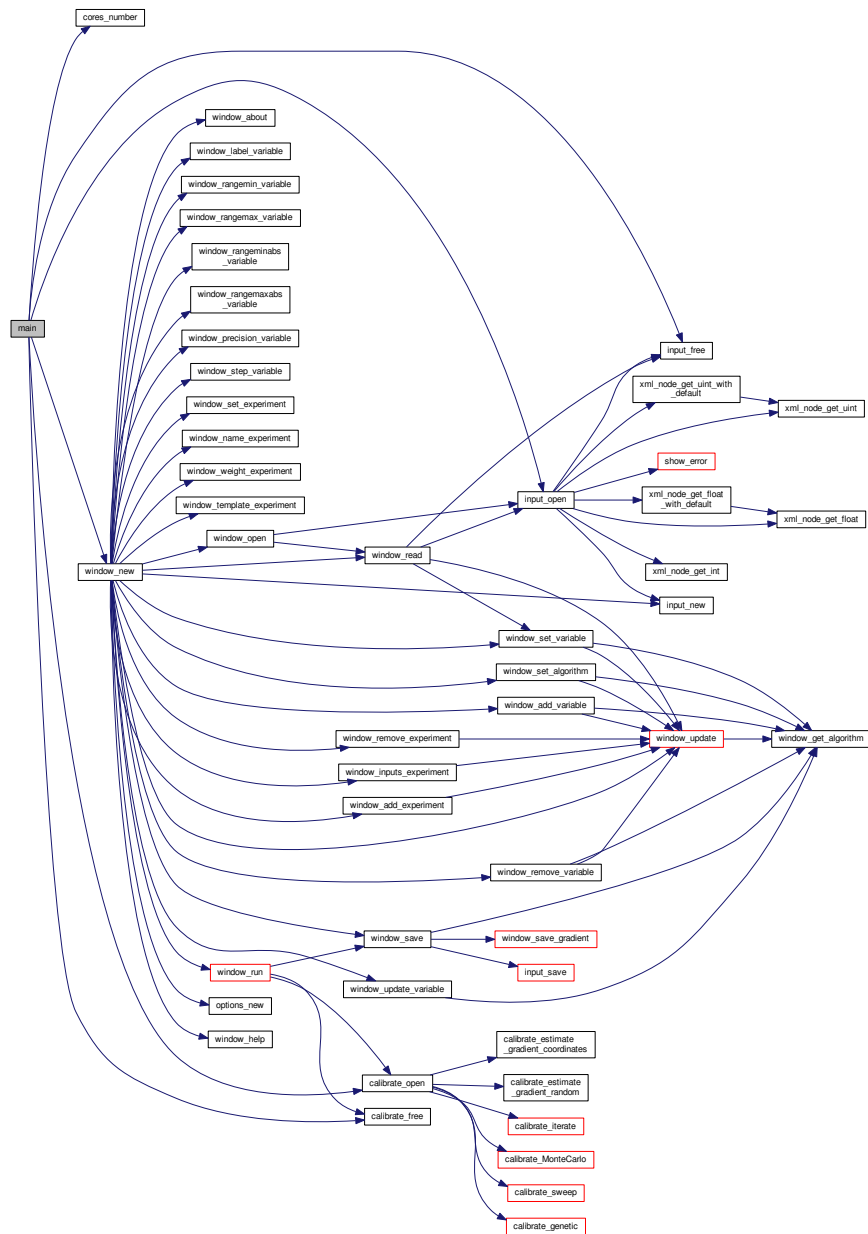
```

```

04986
04987 // Closing
04988 return 0;
04989 }

```

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 [2932](#) of file [mpcotool.c](#).

```

02933 {
02934     unsigned int i;
02935     #if DEBUG
02936     fprintf (stderr, "window_get_algorithm: start\n");
02937     #endif
02938     for (i = 0; i < NALGORITHMS; ++i)
02939         if (gtk_toggle_button_get_active
02940             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02941             break;
02942     #if DEBUG
02943     fprintf (stderr, "window_get_algorithm: %u\n", i);
02944     fprintf (stderr, "window_get_algorithm: end\n");
02945     #endif
02946     return i;
02947 }
```

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

Function to get the gradient base method number.

**Returns**

Gradient base method number.

Definition at line [2955](#) of file [mpcotool.c](#).

```

02956 {
02957     unsigned int i;
02958     #if DEBUG
02959     fprintf (stderr, "window_get_gradient: start\n");
02960     #endif
02961     for (i = 0; i < NGRADIENTS; ++i)
02962         if (gtk_toggle_button_get_active
02963             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02964             break;
02965     #if DEBUG
02966     fprintf (stderr, "window_get_gradient: %u\n", i);
02967     fprintf (stderr, "window_get_gradient: end\n");
02968     #endif
02969     return i;
02970 }
```

**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 [4052](#) of file [mpcotool.c](#).

```

04053 {
04054     unsigned int i;
04055     char *buffer;
04056     #if DEBUG
04057     fprintf (stderr, "window_read: start\n");
04058     #endif
```



```

04059
04060 // Reading new input file
04061 input_free ();
04062 if (!input_open (filename))
04063     return 0;
04064
04065 // Setting GTK+ widgets data
04066 gtk_entry_set_text (window->entry_result, input->result);
04067 gtk_entry_set_text (window->entry_variables, input->
variables);
04068 buffer = g_build_filename (input->directory, input->
simulator, NULL);
04069 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
04070 g_free (buffer);
04071 gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
04072
04073 if (input->evaluator)
04074 {
04075     buffer = g_build_filename (input->directory, input->
evaluator, NULL);
04076     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
04077     g_free (buffer);
04078 }
04079
04080 gtk_toggle_button_set_active
04081 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
04082 switch (input->algorithm)
04083 {
04084     case ALGORITHM_MONTE_CARLO:
04085         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
04086     case ALGORITHM_SWEEP:
04087         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
04088         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
04089         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
04090         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
input->nsteps);
04091
04092 if (input->nsteps)
04093 {
04094     gtk_toggle_button_set_active
04095 (GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
04096     gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
04097     gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
04098     switch (input->gradient_method)
04099     {
04100         case GRADIENT_METHOD_RANDOM:
04101             gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
04102         }
04103     }
04104     break;
04105 default:
04106     gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
04107     gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
04108     gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
04109     gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
04110     gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
04111 }
04112 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
04113 g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
04114 gtk_combo_box_text_remove_all (window->combo_experiment);
04115 for (i = 0; i < input->nexperiments; ++i)
04116     gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
04117 g_signal_handler_unblock
04118 (window->button_experiment, window->
id_experiment_name);
04119 g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
04120 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04121 g_signal_handler_block (window->combo_variable, window->

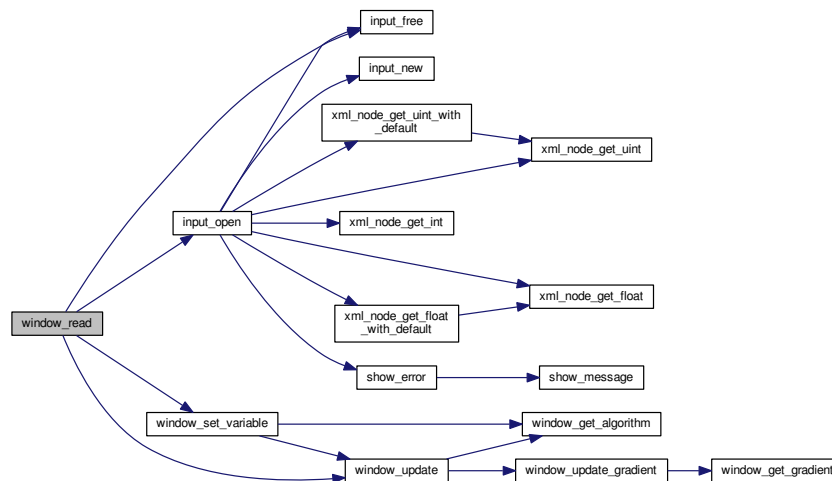
```

```

        id_variable);
04135 g_signal_handler_block (window->entry_variable, window->
        id_variable_label);
04136 gtk_combo_box_text_remove_all (window->combo_variable);
04137 for (i = 0; i < input->nvariables; ++i)
04138     gtk_combo_box_text_append_text (window->combo_variable,
input->label[i]);
04139 g_signal_handler_unblock (window->entry_variable, window->
        id_variable_label);
04140 g_signal_handler_unblock (window->combo_variable, window->
        id_variable);
04141 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04142 window_set_variable ();
04143 window_update ();
04144
04145 #if DEBUG
04146     fprintf (stderr, "window_read: end\n");
04147 #endif
04148     return 1;
04149 }

```

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 3010 of file [mpcotool.c](#).

```

03011 {
03012     GtkFileChooserDialog *dlg;
03013     GtkFileFilter *filter;
03014     char *buffer;
03015
03016     #if DEBUG
03017         fprintf (stderr, "window_save: start\n");
03018     #endif
03019
03020     // Opening the saving dialog
03021     dlg = (GtkFileChooserDialog *)
03022         gtk_file_chooser_dialog_new (gettext ("Save file"),
03023                                     window->window,
03024                                     GTK_FILE_CHOOSER_ACTION_SAVE,
03025                                     gettext ("_Cancel"),

```

```

03026                                     GTK_RESPONSE_CANCEL,
03027                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
03028 gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03029 buffer = g_build_filename (input->directory, input->name, NULL);
03030 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03031 g_free (buffer);
03032
03033 // Adding XML filter
03034 filter = (GtkFileFilter *) gtk_file_filter_new ();
03035 gtk_file_filter_set_name (filter, "XML");
03036 gtk_file_filter_add_pattern (filter, "*.xml");
03037 gtk_file_filter_add_pattern (filter, "*.XML");
03038 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03039
03040 // If OK response then saving
03041 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03042 {
03043
03044     // Adding properties to the root XML node
03045     input->simulator = gtk_file_chooser_get_filename
03046         (GTK_FILE_CHOOSER (window->button_simulator));
03047     if (gtk_toggle_button_get_active
03048         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03049         input->evaluator = gtk_file_chooser_get_filename
03050             (GTK_FILE_CHOOSER (window->button_evaluator));
03051     else
03052         input->evaluator = NULL;
03053     input->result
03054         = (char *) xmlStrdup ((const xmlChar *)
03055             gtk_entry_get_text (window->entry_result));
03056     input->variables
03057         = (char *) xmlStrdup ((const xmlChar *)
03058             gtk_entry_get_text (window->entry_variables));
03059
03060     // Setting the algorithm
03061     switch (window_get_algorithm ())
03062     {
03063     case ALGORITHM_MONTE_CARLO:
03064         input->algorithm = ALGORITHM_MONTE_CARLO;
03065         input->nsimulations
03066             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03067         input->niterations
03068             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03069         input->tolerance = gtk_spin_button_get_value (window->
03070 spin_tolerance);
03071         input->nbest = gtk_spin_button_get_value_as_int (window->
03072 spin_bests);
03073         window_save_gradient ();
03074         break;
03075     case ALGORITHM_SWEEP:
03076         input->algorithm = ALGORITHM_SWEEP;
03077         input->niterations
03078             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03079         input->tolerance = gtk_spin_button_get_value (window->
03080 spin_tolerance);
03081         input->nbest = gtk_spin_button_get_value_as_int (window->
03082 spin_bests);
03083         window_save_gradient ();
03084         break;
03085     default:
03086         input->algorithm = ALGORITHM_GENETIC;
03087         input->nsimulations
03088             = gtk_spin_button_get_value_as_int (window->spin_population);
03089         input->niterations
03090             = gtk_spin_button_get_value_as_int (window->spin_generations);
03091         input->mutation_ratio
03092             = gtk_spin_button_get_value (window->spin_mutation);
03093         input->reproduction_ratio
03094             = gtk_spin_button_get_value (window->spin_reproduction);
03095         input->adaptation_ratio
03096             = gtk_spin_button_get_value (window->spin_adaptation);
03097         break;
03098     }
03099
03100     // Saving the XML file
03101     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03102     input_save (buffer);
03103
03104     // Closing and freeing memory
03105     g_free (buffer);
03106     gtk_widget_destroy (GTK_WIDGET (dlg));
03107 #if DEBUG
03108     fprintf (stderr, "window_save: end\n");
03109 #endif
03110     return 1;
03111 }
03112

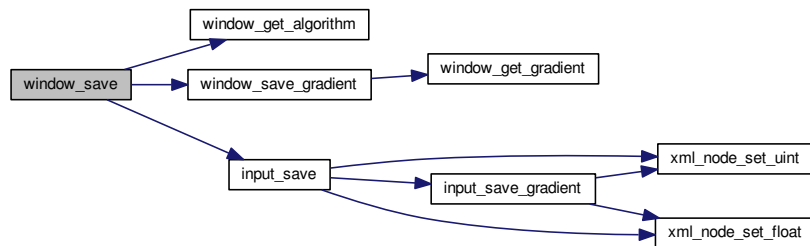
```

```

03109 // Closing and freeing memory
03110 gtk_widget_destroy (GTK_WIDGET (dlg));
03111 #if DEBUG
03112 fprintf (stderr, "window_save: end\n");
03113 #endif
03114 return 0;
03115 }

```

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 3656 of file [mpcotool.c](#).

```

03657 {
03658     unsigned int i, j;
03659     char *buffer;
03660     GFile *file1, *file2;
03661     #if DEBUG
03662     fprintf (stderr, "window_template_experiment: start\n");
03663     #endif
03664     i = (size_t) data;
03665     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03666     file1
03667     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03668     file2 = g_file_new_for_path (input->directory);
03669     buffer = g_file_get_relative_path (file2, file1);
03670     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03671     g_free (buffer);
03672     g_object_unref (file2);
03673     g_object_unref (file1);
03674     #if DEBUG
03675     fprintf (stderr, "window_template_experiment: end\n");
03676     #endif
03677 }

```

#### 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 366 of file [mpcotool.c](#).

```
00367 {
00368     double x = 0.;
00369     xmlChar *buffer;
00370     buffer = xmlGetProp (node, prop);
00371     if (!buffer)
00372         *error_code = 1;
00373     else
00374     {
00375         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00376             *error_code = 2;
00377         else
00378             *error_code = 0;
00379         xmlFree (buffer);
00380     }
00381     return x;
00382 }
```

**5.5.2.27** `double xml_node_get_float_with_default ( xmlNode * node, const xmlChar * prop, double default_value, int * error_code )`

Function to get a floating point number of a XML node property with a default value.

#### Parameters

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

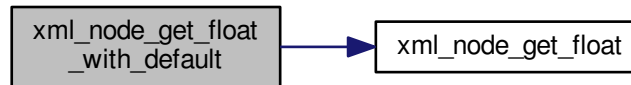
#### Returns

Floating point number value.

Definition at line 400 of file [mpcotool.c](#).

```
00402 {
00403     double x;
00404     if (xmlHasProp (node, prop))
00405         x = xml_node_get_float (node, prop, error_code);
00406     else
00407     {
00408         x = default_value;
00409         *error_code = 0;
00410     }
00411     return x;
00412 }
```

Here is the call graph for this function:



#### 5.5.2.28 `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 }
  
```

#### 5.5.2.29 `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 }

```

**5.5.2.30** `int xml_node_get_uint_with_default ( xmlNode * node, const xmlChar * prop, unsigned int default_value, int * error_code )`

Function to get an unsigned integer number of a XML node property with a default value.

#### Parameters

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

#### Returns

Unsigned integer number value.

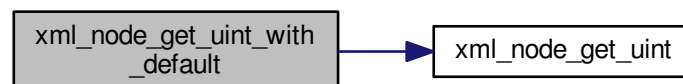
Definition at line 339 of file [mpcotool.c](#).

```

00341 {
00342     unsigned int i;
00343     if (xmlHasProp (node, prop))
00344         i = xml_node_get_uint (node, prop, error_code);
00345     else
00346     {
00347         i = default_value;
00348         *error_code = 0;
00349     }
00350     return i;
00351 }

```

Here is the call graph for this function:



**5.5.2.31** `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 463 of file [mpcotool.c](#).

```
00464 {
00465     xmlChar buffer[64];
00466     snprintf ((char *) buffer, 64, "%.14lg", value);
00467     xmlSetProp (node, prop, buffer);
00468 }
```

**5.5.2.32 void xml\_node\_set\_int ( xmlDoc \* 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 425 of file [mpcotool.c](#).

```
00426 {
00427     xmlChar buffer[64];
00428     snprintf ((char *) buffer, 64, "%d", value);
00429     xmlSetProp (node, prop, buffer);
00430 }
```

**5.5.2.33 void xml\_node\_set\_uint ( xmlDoc \* 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 444 of file [mpcotool.c](#).

```
00445 {
00446     xmlChar buffer[64];
00447     snprintf ((char *) buffer, 64, "%u", value);
00448     xmlSetProp (node, prop, buffer);
00449 }
```

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

```
= {
    "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
    "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
}
```

Array of C-strings with variable formats.

Definition at line 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 111 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
00036 #define _GNU_SOURCE
00037 #include "config.h"
00038 #include <stdio.h>
00039 #include <stdlib.h>
00040 #include <string.h>
00041 #include <math.h>
00042 #include <unistd.h>
00043 #include <locale.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #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
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00092 int ntasks;
00093 unsigned int nthreads;
00094 unsigned int nthreads_gradient;
00096 GMutex mutex[1];
00097 void (*calibrate_algorithm) ();
00099 double (*calibrate_estimate_gradient) (unsigned int variable,
00100                                       unsigned int estimate);
00102 Input input[1];
00104 Calibrate calibrate[1];
00105
00106 const xmlChar *result_name = (xmlChar *) "result";
00108 const xmlChar *variables_name = (xmlChar *) "variables";
00110
00111 const xmlChar *template[MAX_NINPUTS] = {
00112     XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
00113     XML_TEMPLATE4,
00114     XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
00115     XML_TEMPLATE8
00116 };
00117
00118 const char *format[NPRECISIONS] = {
00119     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00120     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00121 };
00122
00123 const double precision[NPRECISIONS] = {
00124     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,
00125     1e-13, 1e-14
00126 };
00127
00128 const char *logo[] = {
00129     "32 32 3 1",
00130     "      c None",
00131     ".      c #0000FF",
00132     "+      c #FF0000",
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 #FFFFFFFFFFFF",
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     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 }
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
00338 unsigned int
00339 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00340                                unsigned int default_value, int *error_code)
00341 {
00342     unsigned int i;
00343     if (xmlHasProp (node, prop))
00344         i = xml_node_get_uint (node, prop, error_code);
00345     else
00346     {
00347         i = default_value;
00348         *error_code = 0;
00349     }
00350     return i;
00351 }
00352
00365 double
00366 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00367 {
00368     double x = 0.;
00369     xmlChar *buffer;
00370     buffer = xmlGetProp (node, prop);
00371     if (!buffer)
00372         *error_code = 1;
00373     else
00374     {
00375         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00376             *error_code = 2;
00377         else
00378             *error_code = 0;
00379         xmlFree (buffer);
00380     }
00381     return x;
00382 }
00383
00399 double
00400 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00401                                 double default_value, int *error_code)
00402 {
00403     double x;
00404     if (xmlHasProp (node, prop))
00405         x = xml_node_get_float (node, prop, error_code);
00406     else
00407     {
00408         x = default_value;
00409         *error_code = 0;
00410     }

```

```

00411     return x;
00412 }
00413
00424 void
00425 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00426 {
00427     xmlChar buffer[64];
00428     snprintf ((char *) buffer, 64, "%d", value);
00429     xmlSetProp (node, prop, buffer);
00430 }
00431
00443 void
00444 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00445 {
00446     xmlChar buffer[64];
00447     snprintf ((char *) buffer, 64, "%u", value);
00448     xmlSetProp (node, prop, buffer);
00449 }
00450
00462 void
00463 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00464 {
00465     xmlChar buffer[64];
00466     snprintf ((char *) buffer, 64, "%.14lg", value);
00467     xmlSetProp (node, prop, buffer);
00468 }
00469
00474 void
00475 input_new ()
00476 {
00477     unsigned int i;
00478     #if DEBUG
00479     fprintf (stderr, "input_new: start\n");
00480     #endif
00481     input->nvariables = input->nexperiments = input->ninputs = input->
nsteps = 0;
00482     input->simulator = input->evaluator = input->directory = input->
name
00483     = input->result = input->variables = NULL;
00484     input->experiment = input->label = NULL;
00485     input->precision = input->nsweeps = input->nbits = NULL;
00486     input->rangemin = input->rangemax = input->rangeminabs = input->
rangemaxabs
00487     = input->weight = input->step = NULL;
00488     for (i = 0; i < MAX_NINPUTS; ++i)
00489         input->template[i] = NULL;
00490     #if DEBUG
00491     fprintf (stderr, "input_new: end\n");
00492     #endif
00493 }
00494
00499 void
00500 input_free ()
00501 {
00502     unsigned int i, j;
00503     #if DEBUG
00504     fprintf (stderr, "input_free: start\n");
00505     #endif
00506     g_free (input->name);
00507     g_free (input->directory);
00508     for (i = 0; i < input->nexperiments; ++i)
00509     {
00510         xmlFree (input->experiment[i]);
00511         for (j = 0; j < input->ninputs; ++j)
00512             xmlFree (input->template[j][i]);
00513         g_free (input->template[j]);
00514     }
00515     g_free (input->experiment);
00516     for (i = 0; i < input->ninputs; ++i)
00517         g_free (input->template[i]);
00518     for (i = 0; i < input->nvariables; ++i)
00519         xmlFree (input->label[i]);
00520     g_free (input->label);
00521     g_free (input->precision);
00522     g_free (input->rangemin);
00523     g_free (input->rangemax);
00524     g_free (input->rangeminabs);
00525     g_free (input->rangemaxabs);
00526     g_free (input->weight);
00527     g_free (input->step);
00528     g_free (input->nsweeps);
00529     g_free (input->nbits);
00530     xmlFree (input->evaluator);
00531     xmlFree (input->simulator);
00532     xmlFree (input->result);
00533     xmlFree (input->variables);
00534     input->nexperiments = input->ninputs = input->nvariables = input->

```

```

        nsteps = 0;
00535 #if DEBUG
00536     fprintf (stderr, "input_free: end\n");
00537 #endif
00538 }
00539
00547 int
00548 input_open (char *filename)
00549 {
00550     char buffer2[64];
00551     char *buffert[MAX_NINPUTS] =
00552         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00553     xmlDoc *doc;
00554     xmlNode *node, *child;
00555     xmlChar *buffer;
00556     char *msg;
00557     int error_code;
00558     unsigned int i;
00559
00560 #if DEBUG
00561     fprintf (stderr, "input_open: start\n");
00562 #endif
00563
00564     // Resetting input data
00565     buffer = NULL;
00566     input_new ();
00567
00568     // Parsing the input file
00569 #if DEBUG
00570     fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00571 #endif
00572     doc = xmlParseFile (filename);
00573     if (!doc)
00574     {
00575         msg = gettext ("Unable to parse the input file");
00576         goto exit_on_error;
00577     }
00578
00579     // Getting the root node
00580 #if DEBUG
00581     fprintf (stderr, "input_open: getting the root node\n");
00582 #endif
00583     node = xmlDocGetRootElement (doc);
00584     if (xmlStrcmp (node->name, XML_CALIBRATE))
00585     {
00586         msg = gettext ("Bad root XML node");
00587         goto exit_on_error;
00588     }
00589
00590     // Getting results file names
00591     input->result = (char *) xmlGetProp (node, XML_RESULT);
00592     if (!input->result)
00593         input->result = (char *) xmlStrdup (result_name);
00594     input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00595     if (!input->variables)
00596         input->variables = (char *) xmlStrdup (variables_name);
00597
00598     // Opening simulator program name
00599     input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00600     if (!input->simulator)
00601     {
00602         msg = gettext ("Bad simulator program");
00603         goto exit_on_error;
00604     }
00605
00606     // Opening evaluator program name
00607     input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00608
00609     // Obtaining pseudo-random numbers generator seed
00610     input->seed
00611         = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00612                                         &error_code);
00613     if (error_code)
00614     {
00615         msg = gettext ("Bad pseudo-random numbers generator seed");
00616         goto exit_on_error;
00617     }
00618
00619     // Opening algorithm
00620     buffer = xmlGetProp (node, XML_ALGORITHM);
00621     if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00622     {
00623         input->algorithm = ALGORITHM_MONTE_CARLO;
00624
00625         // Obtaining simulations number
00626         input->nsimulations

```

```

00627         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00628     if (error_code)
00629     {
00630         msg = gettext ("Bad simulations number");
00631         goto exit_on_error;
00632     }
00633 }
00634 else if (!xmlStrcmp (buffer, XML_SWEEP))
00635     input->algorithm = ALGORITHM_SWEEP;
00636 else if (!xmlStrcmp (buffer, XML_GENETIC))
00637 {
00638     input->algorithm = ALGORITHM_GENETIC;
00639
00640     // Obtaining population
00641     if (xmlHasProp (node, XML_NPOPULATION))
00642     {
00643         input->nsimulations
00644             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00645         if (error_code || input->nsimulations < 3)
00646         {
00647             msg = gettext ("Invalid population number");
00648             goto exit_on_error;
00649         }
00650     }
00651     else
00652     {
00653         msg = gettext ("No population number");
00654         goto exit_on_error;
00655     }
00656
00657     // Obtaining generations
00658     if (xmlHasProp (node, XML_NGENERATIONS))
00659     {
00660         input->niterations
00661             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00662         if (error_code || !input->niterations)
00663         {
00664             msg = gettext ("Invalid generations number");
00665             goto exit_on_error;
00666         }
00667     }
00668     else
00669     {
00670         msg = gettext ("No generations number");
00671         goto exit_on_error;
00672     }
00673
00674     // Obtaining mutation probability
00675     if (xmlHasProp (node, XML_MUTATION))
00676     {
00677         input->mutation_ratio
00678             = xml_node_get_float (node, XML_MUTATION, &error_code);
00679         if (error_code || input->mutation_ratio < 0.
00680             || input->mutation_ratio >= 1.)
00681         {
00682             msg = gettext ("Invalid mutation probability");
00683             goto exit_on_error;
00684         }
00685     }
00686     else
00687     {
00688         msg = gettext ("No mutation probability");
00689         goto exit_on_error;
00690     }
00691
00692     // Obtaining reproduction probability
00693     if (xmlHasProp (node, XML_REPRODUCTION))
00694     {
00695         input->reproduction_ratio
00696             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00697         if (error_code || input->reproduction_ratio < 0.
00698             || input->reproduction_ratio >= 1.0)
00699         {
00700             msg = gettext ("Invalid reproduction probability");
00701             goto exit_on_error;
00702         }
00703     }
00704     else
00705     {
00706         msg = gettext ("No reproduction probability");
00707         goto exit_on_error;
00708     }
00709
00710     // Obtaining adaptation probability
00711     if (xmlHasProp (node, XML_ADAPTATION))
00712     {
00713         input->adaptation_ratio

```

```

00714         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00715     if (error_code || input->adaptation_ratio < 0.
00716         || input->adaptation_ratio >= 1.)
00717     {
00718         msg = gettext ("Invalid adaptation probability");
00719         goto exit_on_error;
00720     }
00721 }
00722 else
00723 {
00724     msg = gettext ("No adaptation probability");
00725     goto exit_on_error;
00726 }
00727
00728 // Checking survivals
00729 i = input->mutation_ratio * input->nsimulations;
00730 i += input->reproduction_ratio * input->nsimulations;
00731 i += input->adaptation_ratio * input->nsimulations;
00732 if (i > input->nsimulations - 2)
00733 {
00734     msg = gettext
00735         ("No enough survival entities to reproduce the population");
00736     goto exit_on_error;
00737 }
00738 }
00739 else
00740 {
00741     msg = gettext ("Unknown algorithm");
00742     goto exit_on_error;
00743 }
00744 xmlFree (buffer);
00745 buffer = NULL;
00746
00747 if (input->algorithm == ALGORITHM_MONTE_CARLO
00748     || input->algorithm == ALGORITHM_SWEEP)
00749 {
00750
00751     // Obtaining iterations number
00752     input->niterations
00753         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00754     if (error_code == 1)
00755         input->niterations = 1;
00756     else if (error_code)
00757     {
00758         msg = gettext ("Bad iterations number");
00759         goto exit_on_error;
00760     }
00761
00762     // Obtaining best number
00763     input->nbest
00764         = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
00765     if (error_code || !input->nbest)
00766     {
00767         msg = gettext ("Invalid best number");
00768         goto exit_on_error;
00769     }
00770
00771     // Obtaining tolerance
00772     input->tolerance
00773         = xml_node_get_float_with_default (node,
XML_TOLERANCE, 0.,
00774                                             &error_code);
00775     if (error_code || input->tolerance < 0.)
00776     {
00777         msg = gettext ("Invalid tolerance");
00778         goto exit_on_error;
00779     }
00780
00781     // Getting gradient method parameters
00782     if (xmlHasProp (node, XML_NSTEPS))
00783     {
00784         input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
00785         if (error_code || !input->nsteps)
00786         {
00787             msg = gettext ("Invalid steps number");
00788             goto exit_on_error;
00789         }
00790         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00791         if (!xmlStrcmp (buffer, XML_COORDINATES))
00792             input->gradient_method = GRADIENT_METHOD_COORDINATES;
00793         else if (!xmlStrcmp (buffer, XML_RANDOM))
00794         {
00795             input->gradient_method = GRADIENT_METHOD_RANDOM;
00796             input->nestimates
00797                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);

```



```

00798         if (error_code || !input->nestimates)
00799         {
00800             msg = gettext ("Invalid estimates number");
00801             goto exit_on_error;
00802         }
00803     }
00804     else
00805     {
00806         msg = gettext ("Unknown method to estimate the gradient");
00807         goto exit_on_error;
00808     }
00809     xmlFree (buffer);
00810     buffer = NULL;
00811     input->relaxation
00812     = xml_node_get_float_with_default (node,
XML_RELAXATION,
00813                                     DEFAULT_RELAXATION, &error_code);
00814     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00815     {
00816         msg = gettext ("Invalid relaxation parameter");
00817         goto exit_on_error;
00818     }
00819 }
00820 else
00821     input->nsteps = 0;
00822 }
00823
00824 // Reading the experimental data
00825 for (child = node->children; child; child = child->next)
00826 {
00827     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00828         break;
00829 #if DEBUG
00830     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00831 #endif
00832     if (xmlHasProp (child, XML_NAME))
00833         buffer = xmlGetProp (child, XML_NAME);
00834     else
00835     {
00836         snprintf (buffer2, 64, "%s %u: %s",
00837                 gettext ("Experiment"),
00838                 input->nexperiments + 1, gettext ("no data file name"));
00839         msg = buffer2;
00840         goto exit_on_error;
00841     }
00842 #if DEBUG
00843     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00844 #endif
00845     input->weight = g_realloc (input->weight,
00846                             (1 + input->nexperiments) * sizeof (double));
00847     input->weight[input->nexperiments]
00848     = xml_node_get_float_with_default (child,
XML_WEIGHT, 1., &error_code);
00849     if (error_code)
00850     {
00851         snprintf (buffer2, 64, "%s %s: %s",
00852                 gettext ("Experiment"), buffer, gettext ("bad weight"));
00853         msg = buffer2;
00854         goto exit_on_error;
00855     }
00856 #if DEBUG
00857     fprintf (stderr, "input_open: weight=%lg\n",
00858             input->weight[input->nexperiments]);
00859 #endif
00860     if (!input->nexperiments)
00861         input->ninputs = 0;
00862 #if DEBUG
00863     fprintf (stderr, "input_open: template[0]\n");
00864 #endif
00865     if (xmlHasProp (child, XML_TEMPLATE1))
00866     {
00867         input->template[0]
00868         = (char **) g_realloc (input->template[0],
00869                             (1 + input->nexperiments) * sizeof (char *));
00870         buffert[0] = (char *) xmlGetProp (child, template[0]);
00871 #if DEBUG
00872         fprintf (stderr, "input_open: experiment=%u templatel=%s\n",
00873                 input->nexperiments, buffert[0]);
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

```

```

00882     {
00883         snprintf (buffer2, 64, "%s %s: %s",
00884                 gettext ("Experiment"), buffer, gettext ("no template"));
00885         msg = buffer2;
00886         goto exit_on_error;
00887     }
00888     for (i = 1; i < MAX_NINPUTS; ++i)
00889     {
00890 #if DEBUG
00891         fprintf (stderr, "input_open: template%u\n", i + 1);
00892 #endif
00893         if (xmlHasProp (child, template[i]))
00894         {
00895             if (input->nexperiments && input->ninputs <= i)
00896             {
00897                 snprintf (buffer2, 64, "%s %s: %s",
00898                         gettext ("Experiment"),
00899                         buffer, gettext ("bad templates number"));
00900                 msg = buffer2;
00901                 while (i-- > 0)
00902                     xmlFree (buffert[i]);
00903                 goto exit_on_error;
00904             }
00905             input->template[i] = (char **)
00906                 g_realloc (input->template[i],
00907                         (1 + input->nexperiments) * sizeof (char *));
00908             buffert[i] = (char *) xmlGetProp (child, template[i]);
00909 #if DEBUG
00910             fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00911                     input->nexperiments, i + 1,
00912                     input->template[i][input->nexperiments]);
00913 #endif
00914             if (!input->nexperiments)
00915                 ++input->ninputs;
00916 #if DEBUG
00917             fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00918 #endif
00919         }
00920         else if (input->nexperiments && input->ninputs > i)
00921         {
00922             snprintf (buffer2, 64, "%s %s: %s%u",
00923                     gettext ("Experiment"),
00924                     buffer, gettext ("no template"), i + 1);
00925             msg = buffer2;
00926             while (i-- > 0)
00927                 xmlFree (buffert[i]);
00928             goto exit_on_error;
00929         }
00930         else
00931             break;
00932     }
00933     input->experiment
00934     = g_realloc (input->experiment,
00935                 (1 + input->nexperiments) * sizeof (char *));
00936     input->experiment[input->nexperiments] = (char *) buffer;
00937     for (i = 0; i < input->ninputs; ++i)
00938         input->template[i][input->nexperiments] = buffert[i];
00939     ++input->nexperiments;
00940 #if DEBUG
00941     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00942 #endif
00943     }
00944     if (!input->nexperiments)
00945     {
00946         msg = gettext ("No calibration experiments");
00947         goto exit_on_error;
00948     }
00949     buffer = NULL;
00950
00951     // Reading the variables data
00952     for (; child; child = child->next)
00953     {
00954         if (xmlStrcmp (child->name, XML_VARIABLE))
00955         {
00956             snprintf (buffer2, 64, "%s %u: %s",
00957                     gettext ("Variable"),
00958                     input->nvariables + 1, gettext ("bad XML node"));
00959             msg = buffer2;
00960             goto exit_on_error;
00961         }
00962         if (xmlHasProp (child, XML_NAME))
00963             buffer = xmlGetProp (child, XML_NAME);
00964         else
00965         {
00966             snprintf (buffer2, 64, "%s %u: %s",
00967                     gettext ("Variable"),
00968                     input->nvariables + 1, gettext ("no name"));

```

```

00969         msg = buffer2;
00970         goto exit_on_error;
00971     }
00972     if (xmlHasProp (child, XML_MINIMUM))
00973     {
00974         input->rangemin = g_realloc
00975             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00976         input->rangeminabs = g_realloc
00977             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00978         input->rangemin[input->nvariables]
00979             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00980         if (error_code)
00981         {
00982             snprintf (buffer2, 64, "%s %s: %s",
00983                 gettext ("Variable"), buffer, gettext ("bad minimum"));
00984             msg = buffer2;
00985             goto exit_on_error;
00986         }
00987         input->rangeminabs[input->nvariables]
00988             = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
00989                 -G_MAXDOUBLE, &error_code);
00990         if (error_code)
00991         {
00992             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00993                 gettext ("bad absolute minimum"));
00994             msg = buffer2;
00995             goto exit_on_error;
00996         }
00997         if (input->rangemin[input->nvariables]
00998             < input->rangeminabs[input->nvariables])
00999         {
01000             snprintf (buffer2, 64, "%s %s: %s",
01001                 gettext ("Variable"),
01002                 buffer, gettext ("minimum range not allowed"));
01003             msg = buffer2;
01004             goto exit_on_error;
01005         }
01006     }
01007     else
01008     {
01009         snprintf (buffer2, 64, "%s %s: %s",
01010             gettext ("Variable"), buffer, gettext ("no minimum range"));
01011         msg = buffer2;
01012         goto exit_on_error;
01013     }
01014     if (xmlHasProp (child, XML_MAXIMUM))
01015     {
01016         input->rangemax = g_realloc
01017             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01018         input->rangemaxabs = g_realloc
01019             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01020         input->rangemax[input->nvariables]
01021             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01022         if (error_code)
01023         {
01024             snprintf (buffer2, 64, "%s %s: %s",
01025                 gettext ("Variable"), buffer, gettext ("bad maximum"));
01026             msg = buffer2;
01027             goto exit_on_error;
01028         }
01029         input->rangemaxabs[input->nvariables]
01030             = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
01031                 G_MAXDOUBLE, &error_code);
01032         if (error_code)
01033         {
01034             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01035                 gettext ("bad absolute maximum"));
01036             msg = buffer2;
01037             goto exit_on_error;
01038         }
01039         if (input->rangemax[input->nvariables]
01040             > input->rangemaxabs[input->nvariables])
01041         {
01042             snprintf (buffer2, 64, "%s %s: %s",
01043                 gettext ("Variable"),
01044                 buffer, gettext ("maximum range not allowed"));
01045             msg = buffer2;
01046             goto exit_on_error;
01047         }
01048     }
01049     else
01050     {
01051         snprintf (buffer2, 64, "%s %s: %s",
01052             gettext ("Variable"), buffer, gettext ("no maximum range"));
01053         msg = buffer2;

```

```

01054         goto exit_on_error;
01055     }
01056     if (input->rangemax[input->nvariables]
01057         < input->rangemin[input->nvariables])
01058     {
01059         snprintf (buffer2, 64, "%s %s: %s",
01060                 gettext ("Variable"), buffer, gettext ("bad range"));
01061         msg = buffer2;
01062         goto exit_on_error;
01063     }
01064     input->precision = g_realloc
01065         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01066     input->precision[input->nvariables]
01067         = xml_node_get_uint_with_default (child,
XML_PRECISION,
01068                                         DEFAULT_PRECISION, &error_code);
01069     if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
01070     {
01071         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01072                 gettext ("bad precision"));
01073         msg = buffer2;
01074         goto exit_on_error;
01075     }
01076     if (input->algorithm == ALGORITHM_SWEEP)
01077     {
01078         if (xmlHasProp (child, XML_NSWEEPS))
01079         {
01080             input->nsweeps = (unsigned int *)
01081                 g_realloc (input->nsweeps,
01082                         (1 + input->nvariables) * sizeof (unsigned int));
01083             input->nsweeps[input->nvariables]
01084                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01085             if (error_code || !input->nsweeps[input->nvariables])
01086             {
01087                 snprintf (buffer2, 64, "%s %s: %s",
01088                         gettext ("Variable"),
01089                         buffer, gettext ("bad sweeps"));
01090                 msg = buffer2;
01091                 goto exit_on_error;
01092             }
01093         }
01094         else
01095         {
01096             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01097                     gettext ("no sweeps number"));
01098             msg = buffer2;
01099             goto exit_on_error;
01100         }
01101     }
01102     #if DEBUG
01103     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01104             input->nsweeps[input->nvariables], input->
nsimulations);
01105     #endif
01106     if (input->algorithm == ALGORITHM_GENETIC)
01107     {
01108         // Obtaining bits representing each variable
01109         if (xmlHasProp (child, XML_NBITS))
01110         {
01111             input->nbits = (unsigned int *)
01112                 g_realloc (input->nbits,
01113                         (1 + input->nvariables) * sizeof (unsigned int));
01114             i = xml_node_get_uint (child, XML_NBITS, &error_code);
01115             if (error_code || !i)
01116             {
01117                 snprintf (buffer2, 64, "%s %s: %s",
01118                         gettext ("Variable"),
01119                         buffer, gettext ("invalid bits number"));
01120                 msg = buffer2;
01121                 goto exit_on_error;
01122             }
01123             input->nbits[input->nvariables] = i;
01124         }
01125         else
01126         {
01127             snprintf (buffer2, 64, "%s %s: %s",
01128                     gettext ("Variable"),
01129                     buffer, gettext ("no bits number"));
01130             msg = buffer2;
01131             goto exit_on_error;
01132         }
01133     }
01134     else if (input->nsteps)
01135     {
01136         input->step = (double *)
01137             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));

```

```

01138         input->step[input->nvariables]
01139         = xml_node_get_float (child, XML_STEP, &error_code);
01140         if (error_code || input->step[input->nvariables] < 0.)
01141         {
01142             snprintf (buffer2, 64, "%s %s: %s",
01143                     gettext ("Variable"),
01144                     buffer, gettext ("bad step size"));
01145             msg = buffer2;
01146             goto exit_on_error;
01147         }
01148     }
01149     input->label = g_realloc
01150     (input->label, (1 + input->nvariables) * sizeof (char *));
01151     input->label[input->nvariables] = (char *) buffer;
01152     ++input->nvariables;
01153 }
01154 if (!input->nvariables)
01155 {
01156     msg = gettext ("No calibration variables");
01157     goto exit_on_error;
01158 }
01159 buffer = NULL;
01160
01161 // Getting the working directory
01162 input->directory = g_path_get_dirname (filename);
01163 input->name = g_path_get_basename (filename);
01164
01165 // Closing the XML document
01166 xmlFreeDoc (doc);
01167
01168 #if DEBUG
01169 fprintf (stderr, "input_open: end\n");
01170 #endif
01171 return 1;
01172
01173 exit_on_error:
01174 xmlFree (buffer);
01175 xmlFreeDoc (doc);
01176 show_error (msg);
01177 input_free ();
01178 #if DEBUG
01179 fprintf (stderr, "input_open: end\n");
01180 #endif
01181 return 0;
01182 }
01183
01195 void
01196 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01197 {
01198     unsigned int i;
01199     char buffer[32], value[32], *buffer2, *buffer3, *content;
01200     FILE *file;
01201     gsize length;
01202     GRegex *regex;
01203
01204 #if DEBUG
01205 fprintf (stderr, "calibrate_input: start\n");
01206 #endif
01207
01208 // Checking the file
01209 if (!template)
01210     goto calibrate_input_end;
01211
01212 // Opening template
01213 content = g_mapped_file_get_contents (template);
01214 length = g_mapped_file_get_length (template);
01215 #if DEBUG
01216 fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01217         content);
01218 #endif
01219 file = g_fopen (input, "w");
01220
01221 // Parsing template
01222 for (i = 0; i < calibrate->nvariables; ++i)
01223 {
01224 #if DEBUG
01225 fprintf (stderr, "calibrate_input: variable=%u\n", i);
01226 #endif
01227     snprintf (buffer, 32, "@variable%u@", i + 1);
01228     regex = g_regex_new (buffer, 0, 0, NULL);
01229     if (i == 0)
01230     {
01231         buffer2 = g_regex_replace_literal (regex, content, length, 0,
01232             calibrate->label[i], 0, NULL);
01233 #if DEBUG
01234 fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01235 #endif

```

```

01236     }
01237     else
01238     {
01239         length = strlen (buffer3);
01240         buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01241                                           calibrate->label[i], 0, NULL);
01242         g_free (buffer3);
01243     }
01244     g_regex_unref (regex);
01245     length = strlen (buffer2);
01246     snprintf (buffer, 32, "@value%u@", i + 1);
01247     regex = g_regex_new (buffer, 0, 0, NULL);
01248     snprintf (value, 32, format[calibrate->precision[i]],
01249              calibrate->value[simulation * calibrate->nvariables + i]);
01250
01251     #if DEBUG
01252         fprintf (stderr, "calibrate_input: value=%s\n", value);
01253     #endif
01254     buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01255                                       0, NULL);
01256     g_free (buffer2);
01257     g_regex_unref (regex);
01258 }
01259
01260 // Saving input file
01261 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01262 g_free (buffer3);
01263 fclose (file);
01264
01265 calibrate_input_end:
01266 #if DEBUG
01267     fprintf (stderr, "calibrate_input: end\n");
01268 #endif
01269 return;
01270 }
01271
01272 double
01273 calibrate_parse (unsigned int simulation, unsigned int experiment)
01274 {
01275     unsigned int i;
01276     double e;
01277     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01278          *buffer3, *buffer4;
01279     FILE *file_result;
01280
01281     #if DEBUG
01282         fprintf (stderr, "calibrate_parse: start\n");
01283         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01284                 experiment);
01285     #endif
01286
01287     // Opening input files
01288     for (i = 0; i < calibrate->ninputs; ++i)
01289     {
01290         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01291         #if DEBUG
01292             fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01293         #endif
01294         calibrate_input (simulation, &input[i][0],
01295                          calibrate->file[i][experiment]);
01296     }
01297     for (; i < MAX_NINPUTS; ++i)
01298         strcpy (&input[i][0], "");
01299     #if DEBUG
01300         fprintf (stderr, "calibrate_parse: parsing end\n");
01301     #endif
01302
01303     // Performing the simulation
01304     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01305     buffer2 = g_path_get_dirname (calibrate->simulator);
01306     buffer3 = g_path_get_basename (calibrate->simulator);
01307     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01308     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01309              buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01310              input[6], input[7], output);
01311     g_free (buffer4);
01312     g_free (buffer3);
01313     g_free (buffer2);
01314     #if DEBUG
01315         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01316     #endif
01317     system (buffer);
01318
01319     // Checking the objective value function
01320     if (calibrate->evaluator)
01321     {
01322         snprintf (result, 32, "result-%u-%u", simulation, experiment);

```

```

01333     buffer2 = g_path_get_dirname (calibrate->evaluator);
01334     buffer3 = g_path_get_basename (calibrate->evaluator);
01335     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01336     snprintf (buffer, 512, "\"%s\" %s %s %s",
01337              buffer4, output, calibrate->experiment[experiment], result);
01338     g_free (buffer4);
01339     g_free (buffer3);
01340     g_free (buffer2);
01341     #if DEBUG
01342     fprintf (stderr, "calibrate_parse: %s\n", buffer);
01343     #endif
01344     system (buffer);
01345     file_result = g_fopen (result, "r");
01346     e = atof (fgets (buffer, 512, file_result));
01347     fclose (file_result);
01348 }
01349 else
01350 {
01351     strcpy (result, "");
01352     file_result = g_fopen (output, "r");
01353     e = atof (fgets (buffer, 512, file_result));
01354     fclose (file_result);
01355 }
01356
01357 // Removing files
01358 #if !DEBUG
01359 for (i = 0; i < calibrate->ninputs; ++i)
01360 {
01361     if (calibrate->file[i][0])
01362     {
01363         snprintf (buffer, 512, RM " %s", &input[i][0]);
01364         system (buffer);
01365     }
01366 }
01367     snprintf (buffer, 512, RM " %s %s", output, result);
01368     system (buffer);
01369 #endif
01370
01371 #if DEBUG
01372 fprintf (stderr, "calibrate_parse: end\n");
01373 #endif
01374
01375 // Returning the objective function
01376 return e * calibrate->weight[experiment];
01377 }
01378
01379 void
01380 calibrate_print ()
01381 {
01382     unsigned int i;
01383     char buffer[512];
01384     #if HAVE_MPI
01385     if (calibrate->mpi_rank)
01386         return;
01387     #endif
01388     printf ("%s\n", gettext ("Best result"));
01389     fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01390     printf ("error = %.15le\n", calibrate->error_old[0]);
01391     fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
01392             error_old[0]);
01393     for (i = 0; i < calibrate->nvariables; ++i)
01394     {
01395         snprintf (buffer, 512, "%s = %s\n",
01396                  calibrate->label[i], format[calibrate->precision[i]]);
01397         printf (buffer, calibrate->value_old[i]);
01398         fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01399     }
01400     fflush (calibrate->file_result);
01401 }
01402
01403 void
01404 calibrate_save_variables (unsigned int simulation, double error)
01405 {
01406     unsigned int i;
01407     char buffer[64];
01408     #if DEBUG
01409     fprintf (stderr, "calibrate_save_variables: start\n");
01410     #endif
01411     for (i = 0; i < calibrate->nvariables; ++i)
01412     {
01413         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01414         fprintf (calibrate->file_variables, buffer,
01415                 calibrate->value[simulation * calibrate->nvariables + i]);
01416     }
01417     fprintf (calibrate->file_variables, "%.14le\n", error);
01418     #if DEBUG
01419     fprintf (stderr, "calibrate_save_variables: end\n");
01420     #endif

```

```

01431 #endif
01432 }
01433
01442 void
01443 calibrate_best (unsigned int simulation, double value)
01444 {
01445     unsigned int i, j;
01446     double e;
01447     #if DEBUG
01448         fprintf (stderr, "calibrate_best: start\n");
01449         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01450                 calibrate->nsaveds, calibrate->nbest);
01451     #endif
01452     if (calibrate->nsaveds < calibrate->nbest
01453         || value < calibrate->error_best[calibrate->nsaveds - 1])
01454     {
01455         if (calibrate->nsaveds < calibrate->nbest)
01456             ++calibrate->nsaveds;
01457         calibrate->error_best[calibrate->nsaveds - 1] = value;
01458         calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01459         for (i = calibrate->nsaveds; --i;)
01460         {
01461             if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01462             {
01463                 j = calibrate->simulation_best[i];
01464                 e = calibrate->error_best[i];
01465                 calibrate->simulation_best[i] = calibrate->
simulation_best[i - 1];
01466                 calibrate->error_best[i] = calibrate->error_best[i - 1];
01467                 calibrate->simulation_best[i - 1] = j;
01468                 calibrate->error_best[i - 1] = e;
01469             }
01470             else
01471                 break;
01472         }
01473     }
01474     #if DEBUG
01475         fprintf (stderr, "calibrate_best: end\n");
01476     #endif
01477 }
01478
01483 void
01484 calibrate_sequential ()
01485 {
01486     unsigned int i, j;
01487     double e;
01488     #if DEBUG
01489         fprintf (stderr, "calibrate_sequential: start\n");
01490         fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01491                 calibrate->nstart, calibrate->nend);
01492     #endif
01493     for (i = calibrate->nstart; i < calibrate->nend; ++i)
01494     {
01495         e = 0.;
01496         for (j = 0; j < calibrate->nexperiments; ++j)
01497             e += calibrate_parse (i, j);
01498         calibrate_best (i, e);
01499         calibrate_save_variables (i, e);
01500     #if DEBUG
01501         fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01502     #endif
01503     }
01504     #if DEBUG
01505         fprintf (stderr, "calibrate_sequential: end\n");
01506     #endif
01507 }
01508
01516 void *
01517 calibrate_thread (ParallelData * data)
01518 {
01519     unsigned int i, j, thread;
01520     double e;
01521     #if DEBUG
01522         fprintf (stderr, "calibrate_thread: start\n");
01523     #endif
01524     thread = data->thread;
01525     #if DEBUG
01526         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01527                 calibrate->thread[thread], calibrate->thread[thread + 1]);
01528     #endif
01529     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01530     {
01531         e = 0.;
01532         for (j = 0; j < calibrate->nexperiments; ++j)
01533             e += calibrate_parse (i, j);
01534         g_mutex_lock (mutex);
01535         calibrate_best (i, e);

```



```

01536     calibrate_save_variables (i, e);
01537     g_mutex_unlock (mutex);
01538 #if DEBUG
01539     fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01540 #endif
01541 }
01542 #if DEBUG
01543     fprintf (stderr, "calibrate_thread: end\n");
01544 #endif
01545     g_thread_exit (NULL);
01546     return NULL;
01547 }
01548
01549 void
01550 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01551                 double *error_best)
01552 {
01553     unsigned int i, j, k, s[calibrate->nbest];
01554     double e[calibrate->nbest];
01555 #if DEBUG
01556     fprintf (stderr, "calibrate_merge: start\n");
01557 #endif
01558     i = j = k = 0;
01559     do
01560     {
01561         if (i == calibrate->nsaveds)
01562         {
01563             s[k] = simulation_best[j];
01564             e[k] = error_best[j];
01565             ++j;
01566             ++k;
01567             if (j == nsaveds)
01568                 break;
01569         }
01570         else if (j == nsaveds)
01571         {
01572             s[k] = calibrate->simulation_best[i];
01573             e[k] = calibrate->error_best[i];
01574             ++i;
01575             ++k;
01576             if (i == calibrate->nsaveds)
01577                 break;
01578         }
01579         else if (calibrate->error_best[i] > error_best[j])
01580         {
01581             s[k] = simulation_best[j];
01582             e[k] = error_best[j];
01583             ++j;
01584             ++k;
01585         }
01586         else
01587         {
01588             s[k] = calibrate->simulation_best[i];
01589             e[k] = calibrate->error_best[i];
01590             ++i;
01591             ++k;
01592         }
01593     }
01594     while (k < calibrate->nbest);
01595     calibrate->nsaveds = k;
01596     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01597     memcpy (calibrate->error_best, e, k * sizeof (double));
01598 #if DEBUG
01599     fprintf (stderr, "calibrate_merge: end\n");
01600 #endif
01601 }
01602
01603 #if HAVE_MPI
01604 void
01605 calibrate_synchronise ()
01606 {
01607     unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01608     double error_best[calibrate->nbest];
01609     MPI_Status mpi_stat;
01610 #if DEBUG
01611     fprintf (stderr, "calibrate_synchronise: start\n");
01612 #endif
01613     if (calibrate->mpi_rank == 0)
01614     {
01615         for (i = 1; i < ntasks; ++i)
01616         {
01617             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01618             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01619                     MPI_COMM_WORLD, &mpi_stat);
01620             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01621                     MPI_COMM_WORLD, &mpi_stat);
01622             calibrate_merge (nsaveds, simulation_best, error_best);
01623         }
01624     }
01625 }

```

```

01638     }
01639 }
01640 else
01641 {
01642     MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01643     MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01644             MPI_COMM_WORLD);
01645     MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01646             MPI_COMM_WORLD);
01647 }
01648 #if DEBUG
01649 fprintf (stderr, "calibrate_synchronise: end\n");
01650 #endif
01651 }
01652 #endif
01653
01654 void
01655 01659 calibrate_sweep ()
01660 {
01661     unsigned int i, j, k, l;
01662     double e;
01663     GThread *thread[nthreads];
01664     ParallelData data[nthreads];
01665     #if DEBUG
01666     fprintf (stderr, "calibrate_sweep: start\n");
01667     #endif
01668     for (i = 0; i < calibrate->nsimulations; ++i)
01669     {
01670         k = i;
01671         for (j = 0; j < calibrate->nvariables; ++j)
01672         {
01673             l = k % calibrate->nsweeps[j];
01674             k /= calibrate->nsweeps[j];
01675             e = calibrate->rangemin[j];
01676             if (calibrate->nsweeps[j] > 1)
01677                 e += 1 * (calibrate->rangemax[j] - calibrate->rangemin[j])
01678                     / (calibrate->nsweeps[j] - 1);
01679             calibrate->value[i * calibrate->nvariables + j] = e;
01680         }
01681     }
01682     calibrate->nsaveds = 0;
01683     if (nthreads <= 1)
01684         calibrate_sequential ();
01685     else
01686     {
01687         for (i = 0; i < nthreads; ++i)
01688         {
01689             data[i].thread = i;
01690             thread[i]
01691                 = g_thread_new (NULL, (void (*) ) calibrate_thread, &data[i]);
01692         }
01693         for (i = 0; i < nthreads; ++i)
01694             g_thread_join (thread[i]);
01695     }
01696     #if HAVE_MPI
01697     // Communicating tasks results
01698     calibrate_synchronise ();
01699     #endif
01700     #if DEBUG
01701     fprintf (stderr, "calibrate_sweep: end\n");
01702     #endif
01703 }
01704
01705 void
01710 calibrate_MonteCarlo ()
01711 {
01712     unsigned int i, j;
01713     GThread *thread[nthreads];
01714     ParallelData data[nthreads];
01715     #if DEBUG
01716     fprintf (stderr, "calibrate_MonteCarlo: start\n");
01717     #endif
01718     for (i = 0; i < calibrate->nsimulations; ++i)
01719     {
01720         for (j = 0; j < calibrate->nvariables; ++j)
01721             calibrate->value[i * calibrate->nvariables + j]
01722                 = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01723                     * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01724         calibrate->nsaveds = 0;
01725         if (nthreads <= 1)
01726             calibrate_sequential ();
01727         else
01728         {
01729             for (i = 0; i < nthreads; ++i)
01730             {
01731                 data[i].thread = i;
01732                 thread[i]
01733                     = g_thread_new (NULL, (void (*) ) calibrate_thread, &data[i]);

```

```

01733     }
01734     for (i = 0; i < nthreads; ++i)
01735         g_thread_join (thread[i]);
01736 }
01737 #if HAVE_MPI
01738 // Communicating tasks results
01739 calibrate_synchronise ();
01740 #endif
01741 #if DEBUG
01742 fprintf (stderr, "calibrate_MonteCarlo: end\n");
01743 #endif
01744 }
01745
01755 void
01756 calibrate_best_gradient (unsigned int simulation, double value)
01757 {
01758     #if DEBUG
01759     fprintf (stderr, "calibrate_best_gradient: start\n");
01760     fprintf (stderr,
01761             "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01762             simulation, value, calibrate->error_best[0]);
01763     #endif
01764     if (value < calibrate->error_best[0])
01765     {
01766         calibrate->error_best[0] = value;
01767         calibrate->simulation_best[0] = simulation;
01768     }
01769     #if DEBUG
01770     fprintf (stderr,
01771             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01772             simulation, value);
01773     #endif
01774     #if DEBUG
01775     fprintf (stderr, "calibrate_best_gradient: end\n");
01776     #endif
01777 }
01778
01785 void
01786 calibrate_gradient_sequential (unsigned int simulation)
01787 {
01788     unsigned int i, j, k;
01789     double e;
01790     #if DEBUG
01791     fprintf (stderr, "calibrate_gradient_sequential: start\n");
01792     fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01793             "nend_gradient=%u\n",
01794             calibrate->nstart_gradient, calibrate->nend_gradient);
01795     #endif
01796     for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01797     {
01798         k = simulation + i;
01799         e = 0.;
01800         for (j = 0; j < calibrate->nexperiments; ++j)
01801             e += calibrate_parse (k, j);
01802         calibrate_best_gradient (k, e);
01803         calibrate_save_variables (k, e);
01804     }
01805     #if DEBUG
01806     fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01807     #endif
01808     #if DEBUG
01809     fprintf (stderr, "calibrate_gradient_sequential: end\n");
01810     #endif
01811 }
01812
01820 void *
01821 calibrate_gradient_thread (ParallelData * data)
01822 {
01823     unsigned int i, j, thread;
01824     double e;
01825     #if DEBUG
01826     fprintf (stderr, "calibrate_gradient_thread: start\n");
01827     #endif
01828     thread = data->thread;
01829     #if DEBUG
01830     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01831             thread,
01832             calibrate->thread_gradient[thread],
01833             calibrate->thread_gradient[thread + 1]);
01834     #endif
01835     for (i = calibrate->thread_gradient[thread];
01836          i < calibrate->thread_gradient[thread + 1]; ++i)
01837     {
01838         e = 0.;
01839         for (j = 0; j < calibrate->nexperiments; ++j)
01840             e += calibrate_parse (i, j);
01841         g_mutex_lock (mutex);

```

```

01842     calibrate_best_gradient (i, e);
01843     calibrate_save_variables (i, e);
01844     g_mutex_unlock (mutex);
01845 #if DEBUG
01846     fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01847 #endif
01848 }
01849 #if DEBUG
01850     fprintf (stderr, "calibrate_gradient_thread: end\n");
01851 #endif
01852     g_thread_exit (NULL);
01853     return NULL;
01854 }
01855
01856 double
01866 calibrate_estimate_gradient_random (unsigned int variable,
01867                                     unsigned int estimate)
01868 {
01869     double x;
01870 #if DEBUG
01871     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01872 #endif
01873     x = calibrate->gradient [variable]
01874         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step [variable];
01875 #if DEBUG
01876     fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01877             variable, x);
01878     fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01879 #endif
01880     return x;
01881 }
01882
01883 double
01893 calibrate_estimate_gradient_coordinates (unsigned int variable,
01894                                         unsigned int estimate)
01895 {
01896     double x;
01897 #if DEBUG
01898     fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01899 #endif
01900     x = calibrate->gradient [variable];
01901     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01902     {
01903         if (estimate & 1)
01904             x += calibrate->step [variable];
01905         else
01906             x -= calibrate->step [variable];
01907     }
01908 #if DEBUG
01909     fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01910             variable, x);
01911     fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01912 #endif
01913     return x;
01914 }
01915
01922 void
01923 calibrate_step_gradient (unsigned int simulation)
01924 {
01925     GThread *thread[nthreads_gradient];
01926     ParallelData data[nthreads_gradient];
01927     unsigned int i, j, k, b;
01928 #if DEBUG
01929     fprintf (stderr, "calibrate_step_gradient: start\n");
01930 #endif
01931     for (i = 0; i < calibrate->nestimates; ++i)
01932     {
01933         k = (simulation + i) * calibrate->nvariables;
01934         b = calibrate->simulation_best[0] * calibrate->nvariables;
01935 #if DEBUG
01936         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01937                 simulation + i, calibrate->simulation_best[0]);
01938 #endif
01939         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01940         {
01941 #if DEBUG
01942             fprintf (stderr,
01943                     "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01944                     i, j, calibrate->value [b]);
01945 #endif
01946             calibrate->value [k]
01947                 = calibrate->value [b] + calibrate_estimate_gradient (j, i);
01948             calibrate->value [k] = fmin (fmax (calibrate->value [k],
01949                                             calibrate->rangeminabs [j]),
01950                                         calibrate->rangemaxabs [j]);
01951 #if DEBUG
01952             fprintf (stderr,

```

```

01953             "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01954             i, j, calibrate->value[k]);
01955 #endif
01956     }
01957 }
01958 if (nthreads_gradient == 1)
01959     calibrate_gradient_sequential (simulation);
01960 else
01961 {
01962     for (i = 0; i <= nthreads_gradient; ++i)
01963     {
01964         calibrate->thread_gradient[i]
01965             = simulation + calibrate->nstart_gradient
01966             + i * (calibrate->nend_gradient - calibrate->
nstart_gradient)
01967             / nthreads_gradient;
01968 #if DEBUG
01969         fprintf (stderr,
01970             "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01971             i, calibrate->thread_gradient[i]);
01972 #endif
01973     }
01974     for (i = 0; i < nthreads_gradient; ++i)
01975     {
01976         data[i].thread = i;
01977         thread[i] = g_thread_new
01978             (NULL, (void *) calibrate_gradient_thread, &data[i]);
01979     }
01980     for (i = 0; i < nthreads_gradient; ++i)
01981         g_thread_join (thread[i]);
01982 }
01983 #if DEBUG
01984 fprintf (stderr, "calibrate_step_gradient: end\n");
01985 #endif
01986 }
01987
01992 void
01993 calibrate_gradient ()
01994 {
01995     unsigned int i, j, k, b, s, adjust;
01996 #if DEBUG
01997     fprintf (stderr, "calibrate_gradient: start\n");
01998 #endif
01999     for (i = 0; i < calibrate->nvariables; ++i)
02000         calibrate->gradient[i] = 0.;
02001     b = calibrate->simulation_best[0] * calibrate->nvariables;
02002     s = calibrate->nsimulations;
02003     adjust = 1;
02004     for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
02005     {
02006 #if DEBUG
02007         fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
02008             i, calibrate->simulation_best[0]);
02009 #endif
02010         calibrate_step_gradient (s);
02011         k = calibrate->simulation_best[0] * calibrate->nvariables;
02012 #if DEBUG
02013         fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
02014             i, calibrate->simulation_best[0]);
02015 #endif
02016         if (k == b)
02017         {
02018             if (adjust)
02019                 for (j = 0; j < calibrate->nvariables; ++j)
02020                     calibrate->step[j] *= 0.5;
02021             for (j = 0; j < calibrate->nvariables; ++j)
02022                 calibrate->gradient[j] = 0.;
02023             adjust = 1;
02024         }
02025         else
02026         {
02027             for (j = 0; j < calibrate->nvariables; ++j)
02028             {
02029 #if DEBUG
02030                 fprintf (stderr,
02031                     "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02032                     j, calibrate->value[k + j], j, calibrate->value[b + j]);
02033 #endif
02034                 calibrate->gradient[j]
02035                     = (1. - calibrate->relaxation) * calibrate->gradient[j]
02036                     + calibrate->relaxation
02037                     * (calibrate->value[k + j] - calibrate->value[b + j]);
02038 #if DEBUG
02039                 fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02040                     j, calibrate->gradient[j]);
02041 #endif
02042             }

```

```

02043         adjust = 0;
02044     }
02045 }
02046 #if DEBUG
02047 fprintf (stderr, "calibrate_gradient: end\n");
02048 #endif
02049 }
02050
02051 double
02052 calibrate_genetic_objective (Entity * entity)
02053 {
02054     unsigned int j;
02055     double objective;
02056     char buffer[64];
02057 #if DEBUG
02058 fprintf (stderr, "calibrate_genetic_objective: start\n");
02059 #endif
02060     for (j = 0; j < calibrate->nvariables; ++j)
02061     {
02062         calibrate->value[entity->id * calibrate->nvariables + j]
02063             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02064     }
02065     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02066     {
02067         objective += calibrate_parse (entity->id, j);
02068         g_mutex_lock (mutex);
02069         for (j = 0; j < calibrate->nvariables; ++j)
02070         {
02071             snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02072             fprintf (calibrate->file_variables, buffer,
02073                     genetic_get_variable (entity, calibrate->genetic_variable + j));
02074         }
02075         fprintf (calibrate->file_variables, "%.14le\n", objective);
02076         g_mutex_unlock (mutex);
02077 #if DEBUG
02078 fprintf (stderr, "calibrate_genetic_objective: end\n");
02079 #endif
02080     }
02081     return objective;
02082 }
02083
02084 void
02085 calibrate_genetic ()
02086 {
02087     char *best_genome;
02088     double best_objective, *best_variable;
02089 #if DEBUG
02090 fprintf (stderr, "calibrate_genetic: start\n");
02091 fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02092         nthreads);
02093 fprintf (stderr,
02094         "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02095         calibrate->nvariables, calibrate->nsimulations,
02096         calibrate->niterations);
02097 fprintf (stderr,
02098         "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02099         calibrate->mutation_ratio, calibrate->
02100         reproduction_ratio,
02101         calibrate->adaptation_ratio);
02102 #endif
02103     genetic_algorithm_default (calibrate->nvariables,
02104                               calibrate->genetic_variable,
02105                               calibrate->nsimulations,
02106                               calibrate->niterations,
02107                               calibrate->mutation_ratio,
02108                               calibrate->reproduction_ratio,
02109                               calibrate->adaptation_ratio,
02110                               &calibrate_genetic_objective,
02111                               &best_genome, &best_variable, &best_objective);
02112 #if DEBUG
02113 fprintf (stderr, "calibrate_genetic: the best\n");
02114 #endif
02115     calibrate->error_old = (double *) g_malloc (sizeof (double));
02116     calibrate->value_old
02117         = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02118     calibrate->error_old[0] = best_objective;
02119     memcpy (calibrate->value_old, best_variable,
02120            calibrate->nvariables * sizeof (double));
02121     g_free (best_genome);
02122     g_free (best_variable);
02123     calibrate_print ();
02124 #if DEBUG
02125 fprintf (stderr, "calibrate_genetic: end\n");
02126 #endif
02127 }
02128
02129 void
02130 calibrate_save_old ()
02131 {

```

```

02144 unsigned int i, j;
02145 #if DEBUG
02146 fprintf (stderr, "calibrate_save_old: start\n");
02147 fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02148 #endif
02149 memcpy (calibrate->error_old, calibrate->error_best,
02150         calibrate->nbest * sizeof (double));
02151 for (i = 0; i < calibrate->nbest; ++i)
02152 {
02153     j = calibrate->simulation_best[i];
02154     #if DEBUG
02155     fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02156     #endif
02157     memcpy (calibrate->value_old + i * calibrate->nvariables,
02158             calibrate->value + j * calibrate->nvariables,
02159             calibrate->nvariables * sizeof (double));
02160 }
02161 #if DEBUG
02162 for (i = 0; i < calibrate->nvariables; ++i)
02163     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02164             i, calibrate->value_old[i]);
02165 fprintf (stderr, "calibrate_save_old: end\n");
02166 #endif
02167 }
02168
02174 void
02175 calibrate_merge_old ()
02176 {
02177     unsigned int i, j, k;
02178     double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
nbest],
02179          *enew, *eold;
02180     #if DEBUG
02181     fprintf (stderr, "calibrate_merge_old: start\n");
02182     #endif
02183     anew = calibrate->error_best;
02184     eold = calibrate->error_old;
02185     i = j = k = 0;
02186     do
02187     {
02188         if (*enew < *eold)
02189         {
02190             memcpy (v + k * calibrate->nvariables,
02191                     calibrate->value
02192                     + calibrate->simulation_best[i] * calibrate->
nvariables,
02193                     calibrate->nvariables * sizeof (double));
02194             e[k] = *enew;
02195             ++k;
02196             ++enew;
02197             ++i;
02198         }
02199         else
02200         {
02201             memcpy (v + k * calibrate->nvariables,
02202                     calibrate->value_old + j * calibrate->nvariables,
02203                     calibrate->nvariables * sizeof (double));
02204             e[k] = *eold;
02205             ++k;
02206             ++eold;
02207             ++j;
02208         }
02209     }
02210     while (k < calibrate->nbest);
02211     memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02212     memcpy (calibrate->error_old, e, k * sizeof (double));
02213     #if DEBUG
02214     fprintf (stderr, "calibrate_merge_old: end\n");
02215     #endif
02216 }
02217
02223 void
02224 calibrate_refine ()
02225 {
02226     unsigned int i, j;
02227     double d;
02228     #if HAVE_MPI
02229     MPI_Status mpi_stat;
02230     #endif
02231     #if DEBUG
02232     fprintf (stderr, "calibrate_refine: start\n");
02233     #endif
02234     #if HAVE_MPI
02235     if (!calibrate->mpi_rank)
02236     {
02237         #endif
02238         for (j = 0; j < calibrate->nvariables; ++j)

```

```

02239     {
02240         calibrate->rangemin[j] = calibrate->rangemax[j]
02241         = calibrate->value_old[j];
02242     }
02243     for (i = 0; ++i < calibrate->nbest;)
02244     {
02245         for (j = 0; j < calibrate->nvariables; ++j)
02246         {
02247             calibrate->rangemin[j]
02248             = fmin (calibrate->rangemin[j],
02249                     calibrate->value_old[i * calibrate->nvariables + j]);
02250             calibrate->rangemax[j]
02251             = fmax (calibrate->rangemax[j],
02252                     calibrate->value_old[i * calibrate->nvariables + j]);
02253         }
02254     }
02255     for (j = 0; j < calibrate->nvariables; ++j)
02256     {
02257         d = calibrate->tolerance
02258         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02259         switch (calibrate->algorithm)
02260         {
02261             case ALGORITHM_MONTE_CARLO:
02262                 d *= 0.5;
02263                 break;
02264             default:
02265                 if (calibrate->nsweeps[j] > 1)
02266                     d /= calibrate->nsweeps[j] - 1;
02267                 else
02268                     d = 0.;
02269         }
02270         calibrate->rangemin[j] -= d;
02271         calibrate->rangemin[j]
02272         = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02273         calibrate->rangemax[j] += d;
02274         calibrate->rangemax[j]
02275         = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02276         printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02277                 calibrate->rangemin[j], calibrate->rangemax[j]);
02278         fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02279                 calibrate->label[j], calibrate->rangemin[j],
02280                 calibrate->rangemax[j]);
02281     }
02282     #if HAVE_MPI
02283     for (i = 1; i < ntasks; ++i)
02284     {
02285         MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02286                  1, MPI_COMM_WORLD);
02287         MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02288                  1, MPI_COMM_WORLD);
02289     }
02290     }
02291     else
02292     {
02293         MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02294                  MPI_COMM_WORLD, &mpi_stat);
02295         MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02296                  MPI_COMM_WORLD, &mpi_stat);
02297     }
02298     #endif
02299     #if DEBUG
02300     fprintf (stderr, "calibrate_refine: end\n");
02301     #endif
02302 }
02303
02304 void
02305 02309 calibrate_step ()
02310 {
02311     #if DEBUG
02312     fprintf (stderr, "calibrate_step: start\n");
02313     #endif
02314     calibrate_algorithm ();
02315     if (calibrate->nsteps)
02316         calibrate_gradient ();
02317     #if DEBUG
02318     fprintf (stderr, "calibrate_step: end\n");
02319     #endif
02320 }
02321
02322 void
02327 calibrate_iterate ()
02328 {
02329     unsigned int i;
02330     #if DEBUG
02331     fprintf (stderr, "calibrate_iterate: start\n");
02332     #endif
02333     calibrate->error_old

```



```

02334     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02335     calibrate->value_old = (double *)
02336     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02337     calibrate_step ();
02338     calibrate_save_old ();
02339     calibrate_refine ();
02340     calibrate_print ();
02341     for (i = 1; i < calibrate->niterations; ++i)
02342     {
02343         calibrate_step ();
02344         calibrate_merge_old ();
02345         calibrate_refine ();
02346         calibrate_print ();
02347     }
02348     #if DEBUG
02349     fprintf (stderr, "calibrate_iterate: end\n");
02350     #endif
02351 }
02352
02353 void
02354 calibrate_free ()
02355 {
02356     unsigned int i, j;
02357     #if DEBUG
02358     fprintf (stderr, "calibrate_free: start\n");
02359     #endif
02360     for (j = 0; j < calibrate->ninputs; ++j)
02361     {
02362         for (i = 0; i < calibrate->nexperiments; ++i)
02363             g_mapped_file_unref (calibrate->file[j][i]);
02364         g_free (calibrate->file[j]);
02365     }
02366     g_free (calibrate->error_old);
02367     g_free (calibrate->value_old);
02368     g_free (calibrate->value);
02369     g_free (calibrate->genetic_variable);
02370     g_free (calibrate->rangemax);
02371     g_free (calibrate->rangemin);
02372     #if DEBUG
02373     fprintf (stderr, "calibrate_free: end\n");
02374     #endif
02375 }
02376
02377 void
02378 calibrate_open ()
02379 {
02380     GTimeZone *tz;
02381     GDateTime *t0, *t;
02382     unsigned int i, j, *nbits;
02383     #if DEBUG
02384     char *buffer;
02385     fprintf (stderr, "calibrate_open: start\n");
02386     #endif
02387     // Getting initial time
02388     #if DEBUG
02389     fprintf (stderr, "calibrate_open: getting initial time\n");
02390     #endif
02391     tz = g_time_zone_new_utc ();
02392     t0 = g_date_time_new_now (tz);
02393     // Obtaining and initing the pseudo-random numbers generator seed
02394     #if DEBUG
02395     fprintf (stderr, "calibrate_open: getting initial seed\n");
02396     #endif
02397     calibrate->seed = input->seed;
02398     gsl_rng_set (calibrate->rng, calibrate->seed);
02399     // Replacing the working directory
02400     #if DEBUG
02401     fprintf (stderr, "calibrate_open: replacing the working directory\n");
02402     #endif
02403     g_chdir (input->directory);
02404     // Getting results file names
02405     calibrate->result = input->result;
02406     calibrate->variables = input->variables;
02407     // Obtaining the simulator file
02408     calibrate->simulator = input->simulator;
02409     // Obtaining the evaluator file
02410     calibrate->evaluator = input->evaluator;
02411     // Reading the algorithm
02412     calibrate->algorithm = input->algorithm;

```

```

02429     switch (calibrate->algorithm)
02430     {
02431         case ALGORITHM_MONTE_CARLO:
02432             calibrate_algorithm = calibrate_MonteCarlo;
02433             break;
02434         case ALGORITHM_SWEEP:
02435             calibrate_algorithm = calibrate_sweep;
02436             break;
02437         default:
02438             calibrate_algorithm = calibrate_genetic;
02439             calibrate->mutation_ratio = input->mutation_ratio;
02440             calibrate->reproduction_ratio = input->
reproduction_ratio;
02441             calibrate->adaptation_ratio = input->adaptation_ratio;
02442     }
02443     calibrate->nvariables = input->nvariables;
02444     calibrate->nsimulations = input->nsimulations;
02445     calibrate->niterations = input->niterations;
02446     calibrate->nbest = input->nbest;
02447     calibrate->tolerance = input->tolerance;
02448     calibrate->nsteps = input->nsteps;
02449     calibrate->nestimates = 0;
02450     if (input->nsteps)
02451     {
02452         calibrate->gradient_method = input->gradient_method;
02453         calibrate->relaxation = input->relaxation;
02454         switch (input->gradient_method)
02455         {
02456             case GRADIENT_METHOD_COORDINATES:
02457                 calibrate->nestimates = 2 * calibrate->nvariables;
02458                 calibrate_estimate_gradient =
calibrate_estimate_gradient_coordinates;
02459                 break;
02460             default:
02461                 calibrate->nestimates = input->nestimates;
02462                 calibrate_estimate_gradient =
calibrate_estimate_gradient_random;
02463         }
02464     }
02465
02466     #if DEBUG
02467     fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02468     #endif
02469     calibrate->simulation_best
02470     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02471     calibrate->error_best
02472     = (double *) alloca (calibrate->nbest * sizeof (double));
02473
02474     // Reading the experimental data
02475     #if DEBUG
02476     buffer = g_get_current_dir ();
02477     fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02478     g_free (buffer);
02479     #endif
02480     calibrate->nexperiments = input->nexperiments;
02481     calibrate->ninputs = input->ninputs;
02482     calibrate->experiment = input->experiment;
02483     calibrate->weight = input->weight;
02484     for (i = 0; i < input->ninputs; ++i)
02485     {
02486         calibrate->template[i] = input->template[i];
02487         calibrate->file[i]
02488         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02489     }
02490     for (i = 0; i < input->nexperiments; ++i)
02491     {
02492         #if DEBUG
02493         fprintf (stderr, "calibrate_open: i=%u\n", i);
02494         fprintf (stderr, "calibrate_open: experiment=%s\n",
calibrate->experiment[i]);
02495         fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
02496         #endif
02497         for (j = 0; j < input->ninputs; ++j)
02498         {
02499             #if DEBUG
02500             fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02501             fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
i, j + 1, calibrate->template[j][i]);
02502             #endif
02503             calibrate->file[j][i]
02504             = g_mapped_file_new (input->template[j][i], 0, NULL);
02505         }
02506     }
02507
02508     // Reading the variables data
02509     #if DEBUG
02510     fprintf (stderr, "calibrate_open: reading variables\n");

```

```

02513 #endif
02514     calibrate->label = input->label;
02515     j = input->nvariables * sizeof (double);
02516     calibrate->rangemin = (double *) g_malloc (j);
02517     calibrate->rangemax = (double *) g_malloc (j);
02518     memcpy (calibrate->rangemin, input->rangemin, j);
02519     memcpy (calibrate->rangemax, input->rangemax, j);
02520     calibrate->rangeminabs = input->rangeminabs;
02521     calibrate->rangemaxabs = input->rangemaxabs;
02522     calibrate->precision = input->precision;
02523     calibrate->nsweeps = input->nsweeps;
02524     calibrate->step = input->step;
02525     nbits = input->nbits;
02526     if (input->algorithm == ALGORITHM_SWEEP)
02527     {
02528         calibrate->nsimulations = 1;
02529         for (i = 0; i < input->nvariables; ++i)
02530         {
02531             if (input->algorithm == ALGORITHM_SWEEP)
02532             {
02533                 calibrate->nsimulations *= input->nsweeps[i];
02534             }
02535             #if DEBUG
02536             fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
02537                     calibrate->nsweeps[i], calibrate->nsimulations);
02538             #endif
02539         }
02540     }
02541     if (calibrate->nsteps)
02542     {
02543         calibrate->gradient
02544             = (double *) alloca (calibrate->nvariables * sizeof (double));
02545         // Allocating values
02546         #if DEBUG
02547         fprintf (stderr, "calibrate_open: allocating variables\n");
02548         fprintf (stderr, "calibrate_open: nvariables=%u\n", calibrate->nvariables);
02549         #endif
02550         calibrate->genetic_variable = NULL;
02551         if (calibrate->algorithm == ALGORITHM_GENETIC)
02552         {
02553             calibrate->genetic_variable = (GeneticVariable *)
02554                 g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02555             for (i = 0; i < calibrate->nvariables; ++i)
02556             {
02557                 #if DEBUG
02558                 fprintf (stderr, "calibrate_open: i=%u min=%lg max=%lg nbits=%u\n",
02559                         i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02560                 #endif
02561                 calibrate->genetic_variable[i].minimum = calibrate->
02562                     rangemin[i];
02563                 calibrate->genetic_variable[i].maximum = calibrate->
02564                     rangemax[i];
02565                 calibrate->genetic_variable[i].nbits = nbits[i];
02566             }
02567             #if DEBUG
02568             fprintf (stderr, "calibrate_open: nvariables=%u nsimulations=%u\n",
02569                     calibrate->nvariables, calibrate->nsimulations);
02570             #endif
02571             calibrate->value = (double *)
02572                 g_malloc ((calibrate->nsimulations
02573                     + calibrate->nestimates * calibrate->nsteps)
02574                     * calibrate->nvariables * sizeof (double));
02575             // Calculating simulations to perform on each task
02576             #if HAVE_MPI
02577             #if DEBUG
02578             fprintf (stderr, "calibrate_open: rank=%u ntasks=%u\n",
02579                     calibrate->mpi_rank, ntasks);
02580             #endif
02581             calibrate->nstart = calibrate->mpi_rank * calibrate->
02582                 nsimulations / ntasks;
02583             calibrate->nend
02584                 = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
02585                 ntasks;
02586             if (calibrate->nsteps)
02587             {
02588                 calibrate->nstart_gradient
02589                     = calibrate->mpi_rank * calibrate->nestimates / ntasks;
02590                 calibrate->nend_gradient
02591                     = (1 + calibrate->mpi_rank) * calibrate->nestimates /
02592                     ntasks;
02593             }
02594             #else
02595             calibrate->nstart = 0;
02596             calibrate->nend = calibrate->nsimulations;
02597             if (calibrate->nsteps)

```

```

02595     {
02596         calibrate->nstart_gradient = 0;
02597         calibrate->nend_gradient = calibrate->nestimates;
02598     }
02599 #endif
02600 #if DEBUG
02601     fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02602             calibrate->nend);
02603 #endif
02604
02605     // Calculating simulations to perform for each thread
02606     calibrate->thread
02607     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02608     for (i = 0; i <= nthreads; ++i)
02609     {
02610         calibrate->thread[i] = calibrate->nstart
02611             + i * (calibrate->nend - calibrate->nstart) / nthreads;
02612 #if DEBUG
02613         fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02614                 calibrate->thread[i]);
02615 #endif
02616     }
02617     if (calibrate->nsteps)
02618         calibrate->thread_gradient = (unsigned int *)
02619             alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02620
02621     // Opening result files
02622     calibrate->file_result = g_fopen (calibrate->result, "w");
02623     calibrate->file_variables = g_fopen (calibrate->variables, "w");
02624
02625     // Performing the algorithm
02626     switch (calibrate->algorithm)
02627     {
02628         // Genetic algorithm
02629         case ALGORITHM_GENETIC:
02630             calibrate_genetic ();
02631             break;
02632
02633         // Iterative algorithm
02634         default:
02635             calibrate_iterate ();
02636     }
02637
02638     // Getting calculation time
02639     t = g_date_time_new_now (tz);
02640     calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02641     g_date_time_unref (t);
02642     g_date_time_unref (t0);
02643     g_time_zone_unref (tz);
02644     printf ("%s = %.6lg s\n",
02645             gettext ("Calculation time"), calibrate->calculation_time);
02646     fprintf (calibrate->file_result, "%s = %.6lg s\n",
02647             gettext ("Calculation time"), calibrate->calculation_time);
02648
02649     // Closing result files
02650     fclose (calibrate->file_variables);
02651     fclose (calibrate->file_result);
02652
02653 #if DEBUG
02654     fprintf (stderr, "calibrate_open: end\n");
02655 #endif
02656 }
02657
02658 #if HAVE_GTK
02659
02660 void
02661 input_save_gradient (xmlNode * node)
02662 {
02663     #if DEBUG
02664         fprintf (stderr, "input_save_gradient: start\n");
02665     #endif
02666     if (input->nsteps)
02667     {
02668         xml_node_set_uint (node, XML_NSTEPS, input->
02669 nsteps);
02670         if (input->relaxation != DEFAULT_RELAXATION)
02671             xml_node_set_float (node, XML_RELAXATION, input->
02672 relaxation);
02673         switch (input->gradient_method)
02674         {
02675             case GRADIENT_METHOD_COORDINATES:
02676                 xmlSetProp (node, XML_GRADIENT_METHOD,
02677 XML_COORDINATES);
02678                 break;
02679             default:
02680                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02681                 xml_node_set_uint (node, XML_NESTIMATES, input->

```

```

        nestimates);
02685     }
02686 }
02687 #if DEBUG
02688 fprintf (stderr, "input_save_gradient: end\n");
02689 #endif
02690 }
02691
02692 void
02693 input_save (char *filename)
02700 {
02701     unsigned int i, j;
02702     char *buffer;
02703     xmlDoc *doc;
02704     xmlNode *node, *child;
02705     GFile *file, *file2;
02706
02707 #if DEBUG
02708 fprintf (stderr, "input_save: start\n");
02709 #endif
02710
02711 // Getting the input file directory
02712 input->name = g_path_get_basename (filename);
02713 input->directory = g_path_get_dirname (filename);
02714 file = g_file_new_for_path (input->directory);
02715
02716 // Opening the input file
02717 doc = xmlNewDoc ((const xmlChar *) "1.0");
02718
02719 // Setting root XML node
02720 node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02721 xmlDocSetRootElement (doc, node);
02722
02723 // Adding properties to the root XML node
02724 if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02725     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02726 if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02727     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02728 file2 = g_file_new_for_path (input->simulator);
02729 buffer = g_file_get_relative_path (file, file2);
02730 g_object_unref (file2);
02731 xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02732 g_free (buffer);
02733 if (input->evaluator)
02734 {
02735     file2 = g_file_new_for_path (input->evaluator);
02736     buffer = g_file_get_relative_path (file, file2);
02737     g_object_unref (file2);
02738     if (xmlStrlen ((xmlChar *) buffer))
02739         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02740     g_free (buffer);
02741 }
02742 if (input->seed != DEFAULT_RANDOM_SEED)
02743     xml_node_set_uint (node, XML_SEED, input->seed);
02744
02745 // Setting the algorithm
02746 buffer = (char *) g_malloc (64);
02747 switch (input->algorithm)
02748 {
02749     case ALGORITHM_MONTE_CARLO:
02750         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02751         snprintf (buffer, 64, "%u", input->nsimulations);
02752         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02753         snprintf (buffer, 64, "%u", input->niterations);
02754         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02755         snprintf (buffer, 64, "%.3lg", input->tolerance);
02756         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02757         snprintf (buffer, 64, "%u", input->nbest);
02758         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02759         input_save_gradient (node);
02760         break;
02761     case ALGORITHM_SWEEP:
02762         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02763         snprintf (buffer, 64, "%u", input->niterations);
02764         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02765         snprintf (buffer, 64, "%.3lg", input->tolerance);
02766         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02767         snprintf (buffer, 64, "%u", input->nbest);
02768         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02769         input_save_gradient (node);
02770         break;
02771     default:
02772         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02773         snprintf (buffer, 64, "%u", input->nsimulations);
02774         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02775         snprintf (buffer, 64, "%u", input->niterations);
02776         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);

```

```

02777     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02778     xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02779     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02780     xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02781     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02782     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02783     break;
02784 }
02785 g_free (buffer);
02786
02787 // Setting the experimental data
02788 for (i = 0; i < input->nexperiments; ++i)
02789 {
02790     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02791     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02792     if (input->weight[i] != 1.)
02793         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02794     for (j = 0; j < input->ninputs; ++j)
02795         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02796 }
02797
02798 // Setting the variables data
02799 for (i = 0; i < input->nvariables; ++i)
02800 {
02801     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02802     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02803     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02804     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02805         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
rangeminabs[i]);
02806     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02807     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02808         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
rangemaxabs[i]);
02809     if (input->precision[i] != DEFAULT_PRECISION)
02810         xml_node_set_uint (child, XML_PRECISION, input->
precision[i]);
02811     if (input->algorithm == ALGORITHM_SWEEP)
02812         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02813     else if (input->algorithm == ALGORITHM_GENETIC)
02814         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02815     if (input->nsteps)
02816         xml_node_set_float (child, XML_STEP, input->
step[i]);
02817 }
02818
02819 // Saving the XML file
02820 xmlSaveFormatFile (filename, doc, 1);
02821
02822 // Freeing memory
02823 xmlFreeDoc (doc);
02824
02825 #if DEBUG
02826 fprintf (stderr, "input_save: end\n");
02827 #endif
02828 }
02829
02830 void
02831 options_new ()
02832 {
02833     #if DEBUG
02834     fprintf (stderr, "options_new: start\n");
02835     #endif
02836     options->label_seed = (GtkLabel *)
02837         gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02838     options->spin_seed = (GtkSpinButton *)
02839         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02840     gtk_widget_set_tooltip_text
02841         (GTK_WIDGET (options->spin_seed),
02842         gettext ("Seed to init the pseudo-random numbers generator"));
02843     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02844     options->label_threads = (GtkLabel *)
02845         gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02846     options->spin_threads
02847         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02848     gtk_widget_set_tooltip_text
02849         (GTK_WIDGET (options->spin_threads),
02850         gettext ("Number of threads to perform the calibration/optimization for "
02851         "the stochastic algorithm"));
02852     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
02853     options->label_gradient = (GtkLabel *)

```

```

02858     gtk_label_new (gettext ("Threads number for the gradient based method"));
02859     options->spin_gradient
02860     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02861     gtk_widget_set_tooltip_text
02862     (GTK_WIDGET (options->spin_gradient),
02863      gettext ("Number of threads to perform the calibration/optimization for "
02864               "the gradient based method"));
02865     gtk_spin_button_set_value (options->spin_gradient,
02866                               (gdouble) nthreads_gradient);
02867     options->grid = (GtkGrid *) gtk_grid_new ();
02868     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
02869     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
02870     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
02871                     0, 1, 1, 1);
02872     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
02873                     1, 1, 1, 1);
02874     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
02875                     0, 2, 1, 1);
02876     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
02877                     1, 2, 1, 1);
02878     gtk_widget_show_all (GTK_WIDGET (options->grid));
02879     options->dialog = (GtkDialog *)
02880     gtk_dialog_new_with_buttons (gettext ("Options"),
02881                                 window->window,
02882                                 GTK_DIALOG_MODAL,
02883                                 gettext ("OK"), GTK_RESPONSE_OK,
02884                                 gettext ("Cancel"), GTK_RESPONSE_CANCEL,
02885                                 NULL);
02886     gtk_container_add
02887     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02888      GTK_WIDGET (options->grid));
02889     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02890     {
02891         input->seed
02892         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02893         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
02894         nthreads_gradient
02895         = gtk_spin_button_get_value_as_int (options->spin_gradient);
02896     }
02897     gtk_widget_destroy (GTK_WIDGET (options->dialog));
02898     #if DEBUG
02899     fprintf (stderr, "options_new: end\n");
02900     #endif
02901 }
02902
02903 void
02904 running_new ()
02905 {
02906     #if DEBUG
02907     fprintf (stderr, "running_new: start\n");
02908     #endif
02909     running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02910     running->dialog = (GtkDialog *)
02911     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02912                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
02913     gtk_container_add
02914     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02915      GTK_WIDGET (running->label));
02916     gtk_widget_show_all (GTK_WIDGET (running->dialog));
02917     #if DEBUG
02918     fprintf (stderr, "running_new: end\n");
02919     #endif
02920 }
02921
02922 int
02923 window_get_algorithm ()
02924 {
02925     unsigned int i;
02926     #if DEBUG
02927     fprintf (stderr, "window_get_algorithm: start\n");
02928     #endif
02929     for (i = 0; i < NALGORITHMS; ++i)
02930         if (gtk_toggle_button_get_active
02931             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02932             break;
02933     #if DEBUG
02934     fprintf (stderr, "window_get_algorithm: %u\n", i);
02935     fprintf (stderr, "window_get_algorithm: end\n");
02936     #endif
02937     return i;
02938 }
02939
02940 int
02941 window_get_gradient ()
02942 {
02943     unsigned int i;
02944     #if DEBUG

```

```

02959     fprintf (stderr, "window_get_gradient: start\n");
02960 #endif
02961     for (i = 0; i < NGRADIENTS; ++i)
02962         if (gtk_toggle_button_get_active
02963             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02964             break;
02965 #if DEBUG
02966     fprintf (stderr, "window_get_gradient: %u\n", i);
02967     fprintf (stderr, "window_get_gradient: end\n");
02968 #endif
02969     return i;
02970 }
02971
02972 void
02973 window_save_gradient ()
02974 {
02975     #if DEBUG
02976     fprintf (stderr, "window_save_gradient: start\n");
02977     #endif
02978     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02979     {
02980         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02981         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
02982         switch (window_get_gradient ())
02983         {
02984             case GRADIENT_METHOD_COORDINATES:
02985                 input->gradient_method = GRADIENT_METHOD_COORDINATES;
02986                 break;
02987             default:
02988                 input->gradient_method = GRADIENT_METHOD_RANDOM;
02989                 input->nestimates
= gtk_spin_button_get_value_as_int (window->spin_estimates);
02990         }
02991     }
02992     else
02993         input->nsteps = 0;
02994     #if DEBUG
02995     fprintf (stderr, "window_save_gradient: end\n");
02996     #endif
02997 }
02998
02999 int
03000 window_save ()
03001 {
03002     GtkFileChooserDialog *dlg;
03003     GtkFileFilter *filter;
03004     char *buffer;
03005
03006     #if DEBUG
03007     fprintf (stderr, "window_save: start\n");
03008     #endif
03009
03010     // Opening the saving dialog
03011     dlg = (GtkFileChooserDialog *)
gtk_file_chooser_dialog_new (gettext ("Save file"),
03012                             window->window,
03013                             GTK_FILE_CHOOSER_ACTION_SAVE,
03014                             gettext ("_Cancel"),
03015                             GTK_RESPONSE_CANCEL,
03016                             gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03017     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03018     buffer = g_build_filename (input->directory, input->name, NULL);
03019     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03020     g_free (buffer);
03021
03022     // Adding XML filter
03023     filter = (GtkFileFilter *) gtk_file_filter_new ();
03024     gtk_file_filter_set_name (filter, "XML");
03025     gtk_file_filter_add_pattern (filter, "*.xml");
03026     gtk_file_filter_add_pattern (filter, "*.XML");
03027     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03028
03029     // If OK response then saving
03030     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03031     {
03032         // Adding properties to the root XML node
03033         input->simulator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_simulator));
03034         if (gtk_toggle_button_get_active
(GTK_TOGGLE_BUTTON (window->check_evaluator)))
03035             input->evaluator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_evaluator));
03036         else
03037             input->evaluator = NULL;
03038         input->result

```



```

03054         = (char *) xmlStrdup ((const xmlChar *)
03055                                gtk_entry_get_text (window->entry_result));
03056     input->variables
03057     = (char *) xmlStrdup ((const xmlChar *)
03058                            gtk_entry_get_text (window->entry_variables));
03059
03060     // Setting the algorithm
03061     switch (window_get_algorithm ())
03062     {
03063     case ALGORITHM_MONTE_CARLO:
03064         input->algorithm = ALGORITHM_MONTE_CARLO;
03065         input->nsimulations
03066         = gtk_spin_button_get_value_as_int (window->spin_simulations);
03067         input->niterations
03068         = gtk_spin_button_get_value_as_int (window->spin_iterations);
03069         input->tolerance = gtk_spin_button_get_value (window->
03070 spin_tolerance);
03071         input->nbest = gtk_spin_button_get_value_as_int (window->
03072 spin_bests);
03073         window_save_gradient ();
03074         break;
03075     case ALGORITHM_SWEEP:
03076         input->algorithm = ALGORITHM_SWEEP;
03077         input->niterations
03078         = gtk_spin_button_get_value_as_int (window->spin_iterations);
03079         input->tolerance = gtk_spin_button_get_value (window->
03080 spin_tolerance);
03081         input->nbest = gtk_spin_button_get_value_as_int (window->
03082 spin_bests);
03083         window_save_gradient ();
03084         break;
03085     default:
03086         input->algorithm = ALGORITHM_GENETIC;
03087         input->nsimulations
03088         = gtk_spin_button_get_value_as_int (window->spin_population);
03089         input->niterations
03090         = gtk_spin_button_get_value_as_int (window->spin_generations);
03091         input->mutation_ratio
03092         = gtk_spin_button_get_value (window->spin_mutation);
03093         input->reproduction_ratio
03094         = gtk_spin_button_get_value (window->spin_reproduction);
03095         input->adaptation_ratio
03096         = gtk_spin_button_get_value (window->spin_adaptation);
03097         break;
03098     }
03099
03100     // Saving the XML file
03101     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03102     input_save (buffer);
03103
03104     // Closing and freeing memory
03105     g_free (buffer);
03106     gtk_widget_destroy (GTK_WIDGET (dlg));
03107
03108     #if DEBUG
03109     fprintf (stderr, "window_save: end\n");
03110     #endif
03111     return 1;
03112 }
03113
03114 // Closing and freeing memory
03115 gtk_widget_destroy (GTK_WIDGET (dlg));
03116
03117 #if DEBUG
03118 fprintf (stderr, "window_save: end\n");
03119 #endif
03120 return 0;
03121 }
03122
03123 void
03124 window_run ()
03125 {
03126     unsigned int i;
03127     char *msg, *msg2, buffer[64], buffer2[64];
03128     #if DEBUG
03129     fprintf (stderr, "window_run: start\n");
03130     #endif
03131     if (!window_save ())
03132     {
03133         #if DEBUG
03134         fprintf (stderr, "window_run: end\n");
03135         #endif
03136         return;
03137     }
03138     running_new ();
03139     while (gtk_events_pending ())
03140         gtk_main_iteration ();
03141     calibrate_open ();
03142     gtk_widget_destroy (GTK_WIDGET (running->dialog));

```

```

03141     snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03142     msg2 = g_strdup (buffer);
03143     for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03144     {
03145         snprintf (buffer, 64, "%s = %s\n",
03146             calibrate->label[i], format[calibrate->precision[i]]);
03147         snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03148         msg = g_strconcat (msg2, buffer2, NULL);
03149         g_free (msg2);
03150     }
03151     snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03152         calibrate->calculation_time);
03153     msg = g_strconcat (msg2, buffer, NULL);
03154     g_free (msg2);
03155     show_message (gettext ("Best result"), msg, INFO_TYPE);
03156     g_free (msg);
03157     calibrate_free ();
03158     #if DEBUG
03159     fprintf (stderr, "window_run: end\n");
03160     #endif
03161 }
03162
03163 void
03164 window_help ()
03165 {
03166     char *buffer, *buffer2;
03167     #if DEBUG
03168     fprintf (stderr, "window_help: start\n");
03169     #endif
03170     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03171         gettext ("user-manual.pdf"), NULL);
03172     buffer = g_filename_to_uri (buffer2, NULL, NULL);
03173     g_free (buffer2);
03174     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03175     #if DEBUG
03176     fprintf (stderr, "window_help: uri=%s\n", buffer);
03177     #endif
03178     g_free (buffer);
03179     #if DEBUG
03180     fprintf (stderr, "window_help: end\n");
03181     #endif
03182 }
03183
03184 void
03185 window_about ()
03186 {
03187     static const gchar *authors[] = {
03188         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03189         "Borja Latorre Garcés <borja.latorre@csic.es>",
03190         NULL
03191     };
03192     #if DEBUG
03193     fprintf (stderr, "window_about: start\n");
03194     #endif
03195     gtk_show_about_dialog
03196     (window->window,
03197         "program_name", "MPCOTool",
03198         "comments",
03199         gettext ("A software to perform calibrations/optimizations of empirical "
03200             "parameters"),
03201         "authors", authors,
03202         "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03203         "version", "1.2.5",
03204         "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03205         "logo", window->logo,
03206         "website", "https://github.com/jburguete/mpcotool",
03207         "license-type", GTK_LICENSE_BSD, NULL);
03208     #if DEBUG
03209     fprintf (stderr, "window_about: end\n");
03210     #endif
03211 }
03212
03213 void
03214 window_update_gradient ()
03215 {
03216     #if DEBUG
03217     fprintf (stderr, "window_update_gradient: start\n");
03218     #endif
03219     gtk_widget_show (GTK_WIDGET (window->check_gradient));
03220     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03221     {
03222         gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03223         gtk_widget_show (GTK_WIDGET (window->label_step));
03224         gtk_widget_show (GTK_WIDGET (window->spin_step));
03225     }
03226     switch (window_get_gradient ())
03227     {
03228     }
03229 }

```

```

03241     case GRADIENT_METHOD_COORDINATES:
03242         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03243         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03244         break;
03245     default:
03246         gtk_widget_show (GTK_WIDGET (window->label_estimates));
03247         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03248     }
03249 #if DEBUG
03250     fprintf (stderr, "window_update_gradient: end\n");
03251 #endif
03252 }
03253
03254 void
03255 window_update ()
03256 {
03257     unsigned int i;
03258     #if DEBUG
03259     fprintf (stderr, "window_update: start\n");
03260     #endif
03261     gtk_widget_set_sensitive
03262         (GTK_WIDGET (window->button_evaluator),
03263          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03264                                         (window->check_evaluator)));
03265     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03266     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03267     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03268     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03269     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03270     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03271     gtk_widget_hide (GTK_WIDGET (window->label_bests));
03272     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03273     gtk_widget_hide (GTK_WIDGET (window->label_population));
03274     gtk_widget_hide (GTK_WIDGET (window->spin_population));
03275     gtk_widget_hide (GTK_WIDGET (window->label_generations));
03276     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03277     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03278     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03279     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03280     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03281     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03282     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03283     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03284     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03285     gtk_widget_hide (GTK_WIDGET (window->label_bits));
03286     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03287     gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03288     gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03289     gtk_widget_hide (GTK_WIDGET (window->label_step));
03290     gtk_widget_hide (GTK_WIDGET (window->spin_step));
03291     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03292     switch (window_get_algorithm ())
03293     {
03294     case ALGORITHM_MONTE_CARLO:
03295         gtk_widget_show (GTK_WIDGET (window->label_simulations));
03296         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03297         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03298         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03299         if (i > 1)
03300         {
03301             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03302             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03303             gtk_widget_show (GTK_WIDGET (window->label_bests));
03304             gtk_widget_show (GTK_WIDGET (window->spin_bests));
03305         }
03306         window_update_gradient ();
03307         break;
03308     case ALGORITHM_SWEEP:
03309         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03310         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03311         if (i > 1)
03312         {
03313             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03314             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03315             gtk_widget_show (GTK_WIDGET (window->label_bests));
03316             gtk_widget_show (GTK_WIDGET (window->spin_bests));
03317         }
03318         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03319         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03320         gtk_widget_show (GTK_WIDGET (window->check_gradient));
03321         window_update_gradient ();
03322         break;
03323     default:
03324         gtk_widget_show (GTK_WIDGET (window->label_population));
03325         gtk_widget_show (GTK_WIDGET (window->spin_population));
03326         gtk_widget_show (GTK_WIDGET (window->label_generations));
03327         gtk_widget_show (GTK_WIDGET (window->spin_generations));
03328     }
03329 }

```

```

03332     gtk_widget_show (GTK_WIDGET (window->label_mutation));
03333     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03334     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03335     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03336     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03337     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03338     gtk_widget_show (GTK_WIDGET (window->label_bits));
03339     gtk_widget_show (GTK_WIDGET (window->spin_bits));
03340 }
03341 gtk_widget_set_sensitive
03342 (GTK_WIDGET (window->button_remove_experiment), input->
nexperiments > 1);
03343 gtk_widget_set_sensitive
03344 (GTK_WIDGET (window->button_remove_variable), input->
nvariables > 1);
03345 for (i = 0; i < input->ninputs; ++i)
03346 {
03347     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03348     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03349     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03350     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03351     g_signal_handler_block
03352     (window->check_template[i], window->id_template[i]);
03353     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03354     gtk_toggle_button_set_active
03355     (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03356     g_signal_handler_unblock
03357     (window->button_template[i], window->id_input[i]);
03358     g_signal_handler_unblock
03359     (window->check_template[i], window->id_template[i]);
03360 }
03361 if (i > 0)
03362 {
03363     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03364     gtk_widget_set_sensitive
03365     (GTK_WIDGET (window->button_template[i - 1]),
03366      gtk_toggle_button_get_active
03367      (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
03368 }
03369 if (i < MAX_NINPUTS)
03370 {
03371     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03372     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03373     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03374     gtk_widget_set_sensitive
03375     (GTK_WIDGET (window->button_template[i]),
03376      gtk_toggle_button_get_active
03377      (GTK_TOGGLE_BUTTON (window->check_template[i])));
03378     g_signal_handler_block
03379     (window->check_template[i], window->id_template[i]);
03380     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03381     gtk_toggle_button_set_active
03382     (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03383     g_signal_handler_unblock
03384     (window->button_template[i], window->id_input[i]);
03385     g_signal_handler_unblock
03386     (window->check_template[i], window->id_template[i]);
03387 }
03388 while (++i < MAX_NINPUTS)
03389 {
03390     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03391     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03392 }
03393 gtk_widget_set_sensitive
03394 (GTK_WIDGET (window->spin_minabs),
03395  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03396 gtk_widget_set_sensitive
03397 (GTK_WIDGET (window->spin_maxabs),
03398  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03399 #if DEBUG
03400 fprintf (stderr, "window_update: end\n");
03401 #endif
03402 }
03403
03404 void
03405 window_set_algorithm ()
03406 {
03407     int i;
03408     #if DEBUG
03409     fprintf (stderr, "window_set_algorithm: start\n");
03410     #endif
03411     i = window_get_algorithm ();
03412     switch (i)
03413     {
03414     case ALGORITHM_SWEEP:

```

```

03419     input->nsweeps = (unsigned int *) g_realloc
03420         (input->nsweeps, input->nvariables * sizeof (unsigned int));
03421     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03422     if (i < 0)
03423         i = 0;
03424     gtk_spin_button_set_value (window->spin_sweeps,
03425                               (gdouble) input->nsweeps[i]);
03426     break;
03427     case ALGORITHM_GENETIC:
03428         input->nbits = (unsigned int *) g_realloc
03429             (input->nbits, input->nvariables * sizeof (unsigned int));
03430         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03431         if (i < 0)
03432             i = 0;
03433         gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
nbits[i]);
03434     }
03435     window_update ();
03436     #if DEBUG
03437     fprintf (stderr, "window_set_algorithm: end\n");
03438     #endif
03439 }
03440
03441 void
03442 window_set_experiment ()
03443 {
03444     unsigned int i, j;
03445     char *buffer1, *buffer2;
03446     #if DEBUG
03447     fprintf (stderr, "window_set_experiment: start\n");
03448     #endif
03449     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03450     gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03451     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03452     buffer2 = g_build_filename (input->directory, buffer1, NULL);
03453     g_free (buffer1);
03454     g_signal_handler_block
03455         (window->button_experiment, window->id_experiment_name);
03456     gtk_file_chooser_set_filename
03457         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03458     g_signal_handler_unblock
03459         (window->button_experiment, window->id_experiment_name);
03460     g_free (buffer2);
03461     for (j = 0; j < input->ninputs; ++j)
03462     {
03463         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03464         buffer2
03465             = g_build_filename (input->directory, input->template[j][i], NULL);
03466         gtk_file_chooser_set_filename
03467             (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03468         g_free (buffer2);
03469         g_signal_handler_unblock
03470             (window->button_template[j], window->id_input[j]);
03471     }
03472     #if DEBUG
03473     fprintf (stderr, "window_set_experiment: end\n");
03474     #endif
03475 }
03476
03477 void
03478 window_remove_experiment ()
03479 {
03480     unsigned int i, j;
03481     #if DEBUG
03482     fprintf (stderr, "window_remove_experiment: start\n");
03483     #endif
03484     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03485     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03486     gtk_combo_box_text_remove (window->combo_experiment, i);
03487     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03488     xmlFree (input->experiment[i]);
03489     --input->nexperiments;
03490     for (j = i; j < input->nexperiments; ++j)
03491     {
03492         input->experiment[j] = input->experiment[j + 1];
03493         input->weight[j] = input->weight[j + 1];
03494     }
03495     j = input->nexperiments - 1;
03496     if (i > j)
03497         i = j;
03498     for (j = 0; j < input->ninputs; ++j)
03499         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03500     g_signal_handler_block

```

```

03509     (window->button_experiment, window->id_experiment_name);
03510     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03511     g_signal_handler_unblock
03512     (window->button_experiment, window->id_experiment_name);
03513     for (j = 0; j < input->ninputs; ++j)
03514         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03515     window_update ();
03516     #if DEBUG
03517     fprintf (stderr, "window_remove_experiment: end\n");
03518     #endif
03519     }
03520
03521 void
03522 window_add_experiment ()
03523 {
03524     unsigned int i, j;
03525     #if DEBUG
03526     fprintf (stderr, "window_add_experiment: start\n");
03527     #endif
03528     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03529     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03530     gtk_combo_box_text_insert_text
03531     (window->combo_experiment, i, input->experiment[i]);
03532     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03533     input->experiment = (char **) g_realloc
03534     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03535     input->weight = (double *) g_realloc
03536     (input->weight, (input->nexperiments + 1) * sizeof (double));
03537     for (j = input->nexperiments - 1; j > i; --j)
03538     {
03539         input->experiment[j + 1] = input->experiment[j];
03540         input->weight[j + 1] = input->weight[j];
03541     }
03542     input->experiment[j + 1] = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03543     input->weight[j + 1] = input->weight[j];
03544     ++input->nexperiments;
03545     for (j = 0; j < input->ninputs; ++j)
03546         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03547     g_signal_handler_block
03548     (window->button_experiment, window->id_experiment_name);
03549     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03550     g_signal_handler_unblock
03551     (window->button_experiment, window->id_experiment_name);
03552     for (j = 0; j < input->ninputs; ++j)
03553         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03554     window_update ();
03555     #if DEBUG
03556     fprintf (stderr, "window_add_experiment: end\n");
03557     #endif
03558     }
03559
03560 void
03561 window_name_experiment ()
03562 {
03563     unsigned int i;
03564     char *buffer;
03565     GFile *file1, *file2;
03566     #if DEBUG
03567     fprintf (stderr, "window_name_experiment: start\n");
03568     #endif
03569     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03570     file1
03571     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03572     file2 = g_file_new_for_path (input->directory);
03573     buffer = g_file_get_relative_path (file2, file1);
03574     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03575     gtk_combo_box_text_remove (window->combo_experiment, i);
03576     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03577     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03578     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03579     g_free (buffer);
03580     g_object_unref (file2);
03581     g_object_unref (file1);
03582     #if DEBUG
03583     fprintf (stderr, "window_name_experiment: end\n");
03584     #endif
03585     }
03586
03587 void

```

```

03601 window_weight_experiment ()
03602 {
03603     unsigned int i;
03604     #if DEBUG
03605         fprintf (stderr, "window_weight_experiment: start\n");
03606     #endif
03607     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03608     input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03609     #if DEBUG
03610         fprintf (stderr, "window_weight_experiment: end\n");
03611     #endif
03612 }
03613
03614 void
03615 window_inputs_experiment ()
03616 {
03617     unsigned int j;
03618     #if DEBUG
03619         fprintf (stderr, "window_inputs_experiment: start\n");
03620     #endif
03621     j = input->ninputs - 1;
03622     if (j
03623         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03624                                             (window->check_template[j]))
03625         --input->ninputs;
03626     if (input->ninputs < MAX_NINPUTS
03627         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03628                                             (window->check_template[j]))
03629     {
03630         ++input->ninputs;
03631         for (j = 0; j < input->ninputs; ++j)
03632         {
03633             input->template[j] = (char **)
03634                 g_realloc (input->template[j], input->nvariables * sizeof (char *));
03635         }
03636     }
03637     window_update ();
03638     #if DEBUG
03639         fprintf (stderr, "window_inputs_experiment: end\n");
03640     #endif
03641 }
03642
03643 void
03644 window_template_experiment (void *data)
03645 {
03646     unsigned int i, j;
03647     char *buffer;
03648     GFile *file1, *file2;
03649     #if DEBUG
03650         fprintf (stderr, "window_template_experiment: start\n");
03651     #endif
03652     i = (size_t) data;
03653     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03654     file1
03655         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03656     file2 = g_file_new_for_path (input->directory);
03657     buffer = g_file_get_relative_path (file2, file1);
03658     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03659     g_free (buffer);
03660     g_object_unref (file2);
03661     g_object_unref (file1);
03662     #if DEBUG
03663         fprintf (stderr, "window_template_experiment: end\n");
03664     #endif
03665 }
03666
03667 void
03668 window_set_variable ()
03669 {
03670     unsigned int i;
03671     #if DEBUG
03672         fprintf (stderr, "window_set_variable: start\n");
03673     #endif
03674     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03675     g_signal_handler_block (window->entry_variable, window->
03676                             id_variable_label);
03677     gtk_entry_set_text (window->entry_variable, input->label[i]);
03678     g_signal_handler_unblock (window->entry_variable, window->
03679                               id_variable_label);
03680     gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03681     gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03682     if (input->rangeminabs[i] != -G_MAXDOUBLE)
03683     {
03684         gtk_spin_button_set_value (window->spin_minabs, input->
03685                                     rangeminabs[i]);
03686         gtk_toggle_button_set_active
03687             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03688     }
03689 }

```

```

03701     }
03702     else
03703     {
03704         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03705         gtk_toggle_button_set_active
03706             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03707     }
03708     if (input->rangemaxabs[i] != G_MAXDOUBLE)
03709     {
03710         gtk_spin_button_set_value (window->spin_maxabs, input->
rangemaxabs[i]);
03711         gtk_toggle_button_set_active
03712             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03713     }
03714     else
03715     {
03716         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03717         gtk_toggle_button_set_active
03718             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03719     }
03720     gtk_spin_button_set_value (window->spin_precision, input->
precision[i]);
03721     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
03722     if (input->nsteps)
03723         gtk_spin_button_set_value (window->spin_step, input->step[i]);
03724     #if DEBUG
03725     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
input->precision[i]);
03726     #endif
03727     switch (window_get_algorithm ())
03728     {
03729     case ALGORITHM_SWEEP:
03730         gtk_spin_button_set_value (window->spin_sweeps,
(gdouble) input->nsweeps[i]);
03731         #if DEBUG
03732         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
input->nsweeps[i]);
03733         #endif
03734         break;
03735     case ALGORITHM_GENETIC:
03736         gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
nbits[i]);
03737         #if DEBUG
03738         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
input->nbits[i]);
03739         #endif
03740         break;
03741     }
03742     window_update ();
03743     #if DEBUG
03744     fprintf (stderr, "window_set_variable: end\n");
03745     #endif
03746 }
03747 void
03748 window_remove_variable ()
03749 {
03750     unsigned int i, j;
03751     #if DEBUG
03752     fprintf (stderr, "window_remove_variable: start\n");
03753     #endif
03754     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03755     g_signal_handler_block (window->combo_variable, window->
id_variable);
03756     gtk_combo_box_text_remove (window->combo_variable, i);
03757     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03758     xmlFree (input->label[i]);
03759     --input->nvariables;
03760     for (j = i; j < input->nvariables; ++j)
03761     {
03762         input->label[j] = input->label[j + 1];
03763         input->rangemin[j] = input->rangemin[j + 1];
03764         input->rangemax[j] = input->rangemax[j + 1];
03765         input->rangeminabs[j] = input->rangeminabs[j + 1];
03766         input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03767         input->precision[j] = input->precision[j + 1];
03768         input->step[j] = input->step[j + 1];
03769         switch (window_get_algorithm ())
03770         {
03771         case ALGORITHM_SWEEP:
03772             input->nsweeps[j] = input->nsweeps[j + 1];
03773             break;
03774         case ALGORITHM_GENETIC:
03775             input->nbits[j] = input->nbits[j + 1];
03776         }
03777     }

```



```

03786     }
03787     j = input->nvariables - 1;
03788     if (i > j)
03789         i = j;
03790     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03791     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03792     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03793     window_update ();
03794     #if DEBUG
03795     fprintf (stderr, "window_remove_variable: end\n");
03796     #endif
03797 }
03798
03803 void
03804 window_add_variable ()
03805 {
03806     unsigned int i, j;
03807     #if DEBUG
03808     fprintf (stderr, "window_add_variable: start\n");
03809     #endif
03810     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03811     g_signal_handler_block (window->combo_variable, window->
id_variable);
03812     gtk_combo_box_text_insert_text (window->combo_variable, i, input->
label[i]);
03813     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03814     input->label = (char **) g_realloc
(input->label, (input->nvariables + 1) * sizeof (char *));
03816     input->rangemin = (double *) g_realloc
(input->rangemin, (input->nvariables + 1) * sizeof (double));
03817     input->rangemax = (double *) g_realloc
(input->rangemax, (input->nvariables + 1) * sizeof (double));
03818     input->rangeminabs = (double *) g_realloc
(input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03819     input->rangemaxabs = (double *) g_realloc
(input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03820     input->precision = (unsigned int *) g_realloc
(input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03821     input->step = (double *) g_realloc
(input->step, (input->nvariables + 1) * sizeof (double));
03822     for (j = input->nvariables - 1; j > i; --j)
03823     {
03824         input->label[j + 1] = input->label[j];
03825         input->rangemin[j + 1] = input->rangemin[j];
03826         input->rangemax[j + 1] = input->rangemax[j];
03827         input->rangeminabs[j + 1] = input->rangeminabs[j];
03828         input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03829         input->precision[j + 1] = input->precision[j];
03830         input->step[j + 1] = input->step[j];
03831     }
03832     input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03833     input->rangemin[j + 1] = input->rangemin[j];
03834     input->rangemax[j + 1] = input->rangemax[j];
03835     input->rangeminabs[j + 1] = input->rangeminabs[j];
03836     input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03837     input->precision[j + 1] = input->precision[j];
03838     input->step[j + 1] = input->step[j];
03839     switch (window_get_algorithm ())
03840     {
03841     case ALGORITHM_SWEEP:
03842         input->nsweeps = (unsigned int *) g_realloc
(input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03843         for (j = input->nvariables - 1; j > i; --j)
03844             input->nsweeps[j + 1] = input->nsweeps[j];
03845         input->nsweeps[j + 1] = input->nsweeps[j];
03846         break;
03847     case ALGORITHM_GENETIC:
03848         input->nbits = (unsigned int *) g_realloc
(input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03849         for (j = input->nvariables - 1; j > i; --j)
03850             input->nbits[j + 1] = input->nbits[j];
03851         input->nbits[j + 1] = input->nbits[j];
03852     }
03853     ++input->nvariables;
03854     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03855     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03856     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03857     window_update ();
03858     #if DEBUG
03859     fprintf (stderr, "window_add_variable: end\n");
03860     #endif
03861 }

```

```

03870
03875 void
03876 window_label_variable ()
03877 {
03878     unsigned int i;
03879     const char *buffer;
03880     #if DEBUG
03881         fprintf (stderr, "window_label_variable: start\n");
03882     #endif
03883     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03884     buffer = gtk_entry_get_text (window->entry_variable);
03885     g_signal_handler_block (window->combo_variable, window->
id_variable);
03886     gtk_combo_box_text_remove (window->combo_variable, i);
03887     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03888     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03889     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03890     #if DEBUG
03891         fprintf (stderr, "window_label_variable: end\n");
03892     #endif
03893 }
03894
03899 void
03900 window_precision_variable ()
03901 {
03902     unsigned int i;
03903     #if DEBUG
03904         fprintf (stderr, "window_precision_variable: start\n");
03905     #endif
03906     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03907     input->precision[i]
03908     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03909     gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03910     gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03911     gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03912     gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03913     #if DEBUG
03914         fprintf (stderr, "window_precision_variable: end\n");
03915     #endif
03916 }
03917
03922 void
03923 window_rangemin_variable ()
03924 {
03925     unsigned int i;
03926     #if DEBUG
03927         fprintf (stderr, "window_rangemin_variable: start\n");
03928     #endif
03929     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03930     input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03931     #if DEBUG
03932         fprintf (stderr, "window_rangemin_variable: end\n");
03933     #endif
03934 }
03935
03940 void
03941 window_rangemax_variable ()
03942 {
03943     unsigned int i;
03944     #if DEBUG
03945         fprintf (stderr, "window_rangemax_variable: start\n");
03946     #endif
03947     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03948     input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03949     #if DEBUG
03950         fprintf (stderr, "window_rangemax_variable: end\n");
03951     #endif
03952 }
03953
03958 void
03959 window_rangeminabs_variable ()
03960 {
03961     unsigned int i;
03962     #if DEBUG
03963         fprintf (stderr, "window_rangeminabs_variable: start\n");
03964     #endif
03965     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03966     input->rangeminabs[i] = gtk_spin_button_get_value (window->
spin_minabs);
03967     #if DEBUG
03968         fprintf (stderr, "window_rangeminabs_variable: end\n");
03969     #endif
03970 }
03971
03976 void
03977 window_rangemaxabs_variable ()

```

```

03978 {
03979     unsigned int i;
03980     #if DEBUG
03981         fprintf (stderr, "window_rangemaxabs_variable: start\n");
03982     #endif
03983     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03984     input->rangemaxabs[i] = gtk_spin_button_get_value (window->
        spin_maxabs);
03985     #if DEBUG
03986         fprintf (stderr, "window_rangemaxabs_variable: end\n");
03987     #endif
03988 }
03989
03994 void
03995 window_step_variable ()
03996 {
03997     unsigned int i;
03998     #if DEBUG
03999         fprintf (stderr, "window_step_variable: start\n");
04000     #endif
04001     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04002     input->step[i] = gtk_spin_button_get_value (window->spin_step);
04003     #if DEBUG
04004         fprintf (stderr, "window_step_variable: end\n");
04005     #endif
04006 }
04007
04012 void
04013 window_update_variable ()
04014 {
04015     int i;
04016     #if DEBUG
04017         fprintf (stderr, "window_update_variable: start\n");
04018     #endif
04019     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04020     if (i < 0)
04021         i = 0;
04022     switch (window_get_algorithm ())
04023     {
04024         case ALGORITHM_SWEEP:
04025             input->nsweeps[i]
04026                 = gtk_spin_button_get_value_as_int (window->spin_sweeps);
04027             #if DEBUG
04028                 fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
04029                     input->nsweeps[i]);
04030             #endif
04031             break;
04032         case ALGORITHM_GENETIC:
04033             input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
04034             #if DEBUG
04035                 fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
04036                     input->nbits[i]);
04037             #endif
04038         }
04039     #if DEBUG
04040         fprintf (stderr, "window_update_variable: end\n");
04041     #endif
04042 }
04043
04051 int
04052 window_read (char *filename)
04053 {
04054     unsigned int i;
04055     char *buffer;
04056     #if DEBUG
04057         fprintf (stderr, "window_read: start\n");
04058     #endif
04059
04060     // Reading new input file
04061     input_free ();
04062     if (!input_open (filename))
04063         return 0;
04064
04065     // Setting GTK+ widgets data
04066     gtk_entry_set_text (window->entry_result, input->result);
04067     gtk_entry_set_text (window->entry_variables, input->variables);
04068     buffer = g_build_filename (input->directory, input->simulator, NULL);
04069     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
        (window->button_simulator), buffer);
04070
04071     g_free (buffer);
04072     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
        (size_t) input->evaluator);
04073
04074     if (input->evaluator)
04075     {
04076         buffer = g_build_filename (input->directory, input->evaluator, NULL);
04077         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
            (window->button_evaluator), buffer);
04078     }

```

```

04079     g_free (buffer);
04080 }
04081 gtk_toggle_button_set_active
04082 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
04083 switch (input->algorithm)
04084 {
04085     case ALGORITHM_MONTE_CARLO:
04086         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
04087     case ALGORITHM_SWEEP:
04088         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
04089         gtk_spin_button_set_value (window->spin_best, (gdouble) input->
nbest);
04090         gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
04091         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
input->nsteps);
04092         if (input->nsteps)
04093         {
04094             gtk_toggle_button_set_active
04095             (GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
04096             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
04097             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
04098             switch (input->gradient_method)
04099             {
04100                 case GRADIENT_METHOD_RANDOM:
04101                     gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
04102             }
04103         }
04104         break;
04105     default:
04106         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
04107         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
04108         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
04109         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
04110         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
04111     }
04112     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
04113     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
04114     gtk_combo_box_text_remove_all (window->combo_experiment);
04115     for (i = 0; i < input->nexperiments; ++i)
04116         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
04117     g_signal_handler_unblock
04118     (window->button_experiment, window->id_experiment_name);
04119     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
04120     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04121     g_signal_handler_block (window->combo_variable, window->
id_variable);
04122     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
04123     gtk_combo_box_text_remove_all (window->combo_variable);
04124     for (i = 0; i < input->nvariables; ++i)
04125         gtk_combo_box_text_append_text (window->combo_variable, input->
label[i]);
04126     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
04127     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
04128     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04129     window_set_variable ();
04130     window_update ();
04131 }
04132 #if DEBUG
04133 fprintf (stderr, "window_read: end\n");
04134 #endif
04135 return 1;
04136 }
04137 void
04138 window_open ()
04139 {
04140     GtkFileChooserDialog *dlg;

```

```

04159  GtkFileFilter *filter;
04160  char *buffer, *directory, *name;
04161
04162  #if DEBUG
04163      fprintf (stderr, "window_open: start\n");
04164  #endif
04165
04166      // Saving a backup of the current input file
04167      directory = g_strdup (input->directory);
04168      name = g_strdup (input->name);
04169
04170      // Opening dialog
04171      dlg = (GtkFileChooserDialog *)
04172          gtk_file_chooser_dialog_new (gettext ("Open input file"),
04173                                     window->window,
04174                                     GTK_FILE_CHOOSER_ACTION_OPEN,
04175                                     gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04176                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04177
04178      // Adding XML filter
04179      filter = (GtkFileFilter *) gtk_file_filter_new ();
04180      gtk_file_filter_set_name (filter, "XML");
04181      gtk_file_filter_add_pattern (filter, "*.xml");
04182      gtk_file_filter_add_pattern (filter, "*.XML");
04183      gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
04184
04185      // If OK saving
04186      while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04187      {
04188          // Traying to open the input file
04189          buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04190          if (!window_read (buffer))
04191          {
04192              #if DEBUG
04193                  fprintf (stderr, "window_open: error reading input file\n");
04194              #endif
04195              g_free (buffer);
04196
04197              // Reading backup file on error
04198              buffer = g_build_filename (directory, name, NULL);
04199              if (!input_open (buffer))
04200              {
04201                  // Closing on backup file reading error
04202                  #if DEBUG
04203                      fprintf (stderr, "window_read: error reading backup file\n");
04204                  #endif
04205                  g_free (buffer);
04206                  break;
04207              }
04208              g_free (buffer);
04209          }
04210          else
04211          {
04212              g_free (buffer);
04213              break;
04214          }
04215      }
04216
04217      // Freeing and closing
04218      g_free (name);
04219      g_free (directory);
04220      gtk_widget_destroy (GTK_WIDGET (dlg));
04221
04222      #if DEBUG
04223          fprintf (stderr, "window_open: end\n");
04224      #endif
04225  }
04226
04227  void
04228  window_new ()
04229  {
04230      unsigned int i;
04231      char *buffer, *buffer2, buffer3[64];
04232      char *label_algorithm[NALGORITHMS] = {
04233          "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04234      };
04235      char *tip_algorithm[NALGORITHMS] = {
04236          gettext ("Monte-Carlo brute force algorithm"),
04237          gettext ("Sweep brute force algorithm"),
04238          gettext ("Genetic algorithm")
04239      };
04240      char *label_gradient[NGRADIENTS] = {
04241          gettext ("_Coordinates descent"), gettext ("_Random")
04242      };
04243      char *tip_gradient[NGRADIENTS] = {
04244          gettext ("Coordinates descent gradient estimate method"),

```

```

04250     gettext ("Random gradient estimate method")
04251 };
04252
04253 #if DEBUG
04254     fprintf (stderr, "window_new: start\n");
04255 #endif
04256
04257 // Creating the window
04258 window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04259
04260 // Finish when closing the window
04261 g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04262
04263 // Setting the window title
04264 gtk_window_set_title (window->window, "MPCOTool");
04265
04266 // Creating the open button
04267 window->button_open = (GtkToolButton *) gtk_tool_button_new
04268     (gtk_image_new_from_icon_name ("document-open",
04269         GTK_ICON_SIZE_LARGE_TOOLBAR),
04270     gettext ("Open"));
04271 g_signal_connect (window->button_open, "clicked", window_open, NULL);
04272
04273 // Creating the save button
04274 window->button_save = (GtkToolButton *) gtk_tool_button_new
04275     (gtk_image_new_from_icon_name ("document-save",
04276         GTK_ICON_SIZE_LARGE_TOOLBAR),
04277     gettext ("Save"));
04278 g_signal_connect (window->button_save, "clicked", (void (*)(
04279     window_save,
04280     NULL));
04281
04282 // Creating the run button
04283 window->button_run = (GtkToolButton *) gtk_tool_button_new
04284     (gtk_image_new_from_icon_name ("system-run",
04285         GTK_ICON_SIZE_LARGE_TOOLBAR),
04286     gettext ("Run"));
04287 g_signal_connect (window->button_run, "clicked", window_run, NULL);
04288
04289 // Creating the options button
04290 window->button_options = (GtkToolButton *) gtk_tool_button_new
04291     (gtk_image_new_from_icon_name ("preferences-system",
04292         GTK_ICON_SIZE_LARGE_TOOLBAR),
04293     gettext ("Options"));
04294 g_signal_connect (window->button_options, "clicked", options_new, NULL);
04295
04296 // Creating the help button
04297 window->button_help = (GtkToolButton *) gtk_tool_button_new
04298     (gtk_image_new_from_icon_name ("help-browser",
04299         GTK_ICON_SIZE_LARGE_TOOLBAR),
04300     gettext ("Help"));
04301 g_signal_connect (window->button_help, "clicked", window_help, NULL);
04302
04303 // Creating the about button
04304 window->button_about = (GtkToolButton *) gtk_tool_button_new
04305     (gtk_image_new_from_icon_name ("help-about",
04306         GTK_ICON_SIZE_LARGE_TOOLBAR),
04307     gettext ("About"));
04308 g_signal_connect (window->button_about, "clicked", window_about, NULL);
04309
04310 // Creating the exit button
04311 window->button_exit = (GtkToolButton *) gtk_tool_button_new
04312     (gtk_image_new_from_icon_name ("application-exit",
04313         GTK_ICON_SIZE_LARGE_TOOLBAR),
04314     gettext ("Exit"));
04315 g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
04316
04317 // Creating the buttons bar
04318 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04319 gtk_toolbar_insert
04320     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
04321 gtk_toolbar_insert
04322     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
04323 gtk_toolbar_insert
04324     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
04325 gtk_toolbar_insert
04326     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
04327 gtk_toolbar_insert
04328     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
04329 gtk_toolbar_insert
04330     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
04331 gtk_toolbar_insert
04332     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
04333 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04334
04335 // Creating the simulator program label and entry
04336 window->label_simulator

```

```

04336     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04337 window->button_simulator = (GtkFileChooserButton *)
04338     gtk_file_chooser_button_new (gettext ("Simulator program"),
04339     GTK_FILE_CHOOSER_ACTION_OPEN);
04340 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04341     gettext ("Simulator program executable file"));
04342
04343 // Creating the evaluator program label and entry
04344 window->check_evaluator = (GtkCheckButton *)
04345     gtk_check_button_new_with_mnemonic (gettext ("Evaluator program"));
04346 g_signal_connect (window->check_evaluator, "toggled",
04347     window_update, NULL);
04348 window->button_evaluator = (GtkFileChooserButton *)
04349     gtk_file_chooser_button_new (gettext ("Evaluator program"),
04350     GTK_FILE_CHOOSER_ACTION_OPEN);
04351 gtk_widget_set_tooltip_text
04352     (GTK_WIDGET (window->button_evaluator),
04353     gettext ("Optional evaluator program executable file"));
04354
04355 // Creating the results files labels and entries
04356 window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
04357 window->entry_result = (GtkEntry *) gtk_entry_new ();
04358 gtk_widget_set_tooltip_text
04359     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
04360 window->label_variables
04361     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
04362 window->entry_variables = (GtkEntry *) gtk_entry_new ();
04363 gtk_widget_set_tooltip_text
04364     (GTK_WIDGET (window->entry_variables),
04365     gettext ("All simulated results file"));
04366
04367 // Creating the files grid and attaching widgets
04368 window->grid_files = (GtkGrid *) gtk_grid_new ();
04369 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04370     label_simulator),
04371     0, 0, 1, 1);
04372 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04373     button_simulator),
04374     1, 0, 1, 1);
04375 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04376     check_evaluator),
04377     2, 0, 1, 1);
04378 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04379     button_evaluator),
04380     3, 0, 1, 1);
04381 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04382     label_result),
04383     0, 1, 1, 1);
04384 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04385     entry_result),
04386     1, 1, 1, 1);
04387 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04388     label_variables),
04389     2, 1, 1, 1);
04390 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
04391     entry_variables),
04392     3, 1, 1, 1);
04393
04394 // Creating the algorithm properties
04395 window->label_simulations = (GtkLabel *) gtk_label_new
04396     (gettext ("Simulations number"));
04397 window->spin_simulations
04398     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04399 gtk_widget_set_tooltip_text
04400     (GTK_WIDGET (window->spin_simulations),
04401     gettext ("Number of simulations to perform for each iteration"));
04402 window->label_iterations = (GtkLabel *)
04403     gtk_label_new (gettext ("Iterations number"));
04404 window->spin_iterations
04405     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04406 gtk_widget_set_tooltip_text
04407     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04408 g_signal_connect
04409     (window->spin_iterations, "value-changed", window_update, NULL);
04410 window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04411 window->spin_tolerance
04412     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04413 gtk_widget_set_tooltip_text
04414     (GTK_WIDGET (window->spin_tolerance),
04415     gettext ("Tolerance to set the variable interval on the next iteration"));
04416 window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04417 window->spin_bests
04418     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04419 gtk_widget_set_tooltip_text
04420     (GTK_WIDGET (window->spin_bests),
04421     gettext ("Number of best simulations used to set the variable interval "
04422     "on the next iteration"));

```

```

04414 window->label_population
04415 = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04416 window->spin_population
04417 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04418 gtk_widget_set_tooltip_text
04419 (GTK_WIDGET (window->spin_population),
04420  gettext ("Number of population for the genetic algorithm"));
04421 window->label_generations
04422 = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04423 window->spin_generations
04424 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04425 gtk_widget_set_tooltip_text
04426 (GTK_WIDGET (window->spin_generations),
04427  gettext ("Number of generations for the genetic algorithm"));
04428 window->label_mutation
04429 = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04430 window->spin_mutation
04431 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04432 gtk_widget_set_tooltip_text
04433 (GTK_WIDGET (window->spin_mutation),
04434  gettext ("Ratio of mutation for the genetic algorithm"));
04435 window->label_reproduction
04436 = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04437 window->spin_reproduction
04438 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04439 gtk_widget_set_tooltip_text
04440 (GTK_WIDGET (window->spin_reproduction),
04441  gettext ("Ratio of reproduction for the genetic algorithm"));
04442 window->label_adaptation
04443 = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04444 window->spin_adaptation
04445 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04446 gtk_widget_set_tooltip_text
04447 (GTK_WIDGET (window->spin_adaptation),
04448  gettext ("Ratio of adaptation for the genetic algorithm"));
04449
04450 // Creating the gradient based method properties
04451 window->check_gradient = (GtkCheckButton *)
04452   gtk_check_button_new_with_mnemonic (gettext ("Gradient based method"));
04453 g_signal_connect (window->check_gradient, "clicked",
04454   window_update, NULL);
04455 window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04456 window->button_gradient[0] = (GtkRadioButton *)
04457   gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04458 gtk_grid_attach (window->grid_gradient,
04459   GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04459 g_signal_connect (window->button_gradient[0], "clicked",
04460   window_update, NULL);
04461 for (i = 0; ++i < NGRADIENTS;)
04462 {
04463   window->button_gradient[i] = (GtkRadioButton *)
04464     gtk_radio_button_new_with_mnemonic
04465       (gtk_radio_button_get_group (window->button_gradient[0]),
04466        label_gradient[i]);
04467   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04468     tip_gradient[i]);
04469   gtk_grid_attach (window->grid_gradient,
04470     GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04471   g_signal_connect (window->button_gradient[i], "clicked",
04472     window_update, NULL);
04473 }
04474 window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04475 window->spin_steps = (GtkSpinButton *)
04476   gtk_spin_button_new_with_range (1., 1.e12, 1.);
04477 window->label_estimates
04478 = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04479 window->spin_estimates = (GtkSpinButton *)
04480   gtk_spin_button_new_with_range (1., 1.e3, 1.);
04481 window->label_relaxation
04482 = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04483 window->spin_relaxation = (GtkSpinButton *)
04484   gtk_spin_button_new_with_range (0., 2., 0.001);
04485 gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04486   label_steps),
04487   0, NGRADIENTS, 1, 1);
04488 gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04489   spin_steps),
04490   1, NGRADIENTS, 1, 1);
04491 gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04492   label_estimates),
04493   0, NGRADIENTS + 1, 1, 1);
04494 gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04495   spin_estimates),
04496   1, NGRADIENTS + 1, 1, 1);
04497 gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04498   label_relaxation),
04499   0, NGRADIENTS + 2, 1, 1);

```



```
04494   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_relaxation),
04495                       1, NGRADIENTS + 2, 1, 1);
04496
04497   // Creating the array of algorithms
04498   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
04499   window->button_algorithm[0] = (GtkRadioButton *)
04500       gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04501   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
04502       tip_algorithm[0]);
04503   gtk_grid_attach (window->grid_algorithm,
04504       GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
04505   g_signal_connect (window->button_algorithm[0], "clicked",
04506       window_set_algorithm, NULL);
04507   for (i = 0; ++i < NALGORITHMS;)
04508   {
04509       window->button_algorithm[i] = (GtkRadioButton *)
04510       gtk_radio_button_new_with_mnemonic
04511       (gtk_radio_button_get_group (window->button_algorithm[0]),
04512       label_algorithm[i]);
04513       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
04514       tip_algorithm[i]);
04515       gtk_grid_attach (window->grid_algorithm,
04516       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
04517       g_signal_connect (window->button_algorithm[i], "clicked",
04518       window_set_algorithm, NULL);
04519   }
04520   gtk_grid_attach (window->grid_algorithm,
04521       GTK_WIDGET (window->label_simulations), 0,
04522       NALGORITHMS, 1, 1);
04523   gtk_grid_attach (window->grid_algorithm,
04524       GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
04525   gtk_grid_attach (window->grid_algorithm,
04526       GTK_WIDGET (window->label_iterations), 0,
04527       NALGORITHMS + 1, 1, 1);
04528   gtk_grid_attach (window->grid_algorithm,
04529       GTK_WIDGET (window->spin_iterations), 1,
04530       NALGORITHMS + 1, 1, 1);
04531   gtk_grid_attach (window->grid_algorithm,
04532       GTK_WIDGET (window->label_tolerance), 0,
04533       NALGORITHMS + 2, 1, 1);
04534   gtk_grid_attach (window->grid_algorithm,
04535       GTK_WIDGET (window->spin_tolerance), 1,
04536       NALGORITHMS + 2, 1, 1);
04537   gtk_grid_attach (window->grid_algorithm,
04538       GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
04539   gtk_grid_attach (window->grid_algorithm,
04540       GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
04541   gtk_grid_attach (window->grid_algorithm,
04542       GTK_WIDGET (window->label_population), 0,
04543       NALGORITHMS + 4, 1, 1);
04544   gtk_grid_attach (window->grid_algorithm,
04545       GTK_WIDGET (window->spin_population), 1,
04546       NALGORITHMS + 4, 1, 1);
04547   gtk_grid_attach (window->grid_algorithm,
04548       GTK_WIDGET (window->label_generations), 0,
04549       NALGORITHMS + 5, 1, 1);
04550   gtk_grid_attach (window->grid_algorithm,
04551       GTK_WIDGET (window->spin_generations), 1,
04552       NALGORITHMS + 5, 1, 1);
04553   gtk_grid_attach (window->grid_algorithm,
04554       GTK_WIDGET (window->label_mutation), 0,
04555       NALGORITHMS + 6, 1, 1);
04556   gtk_grid_attach (window->grid_algorithm,
04557       GTK_WIDGET (window->spin_mutation), 1,
04558       NALGORITHMS + 6, 1, 1);
04559   gtk_grid_attach (window->grid_algorithm,
04560       GTK_WIDGET (window->label_reproduction), 0,
04561       NALGORITHMS + 7, 1, 1);
04562   gtk_grid_attach (window->grid_algorithm,
04563       GTK_WIDGET (window->spin_reproduction), 1,
04564       NALGORITHMS + 7, 1, 1);
04565   gtk_grid_attach (window->grid_algorithm,
04566       GTK_WIDGET (window->label_adaptation), 0,
04567       NALGORITHMS + 8, 1, 1);
04568   gtk_grid_attach (window->grid_algorithm,
04569       GTK_WIDGET (window->spin_adaptation), 1,
04570       NALGORITHMS + 8, 1, 1);
04571   gtk_grid_attach (window->grid_algorithm,
04572       GTK_WIDGET (window->check_gradient), 0,
04573       NALGORITHMS + 9, 2, 1);
04574   gtk_grid_attach (window->grid_algorithm,
04575       GTK_WIDGET (window->grid_gradient), 0,
04576       NALGORITHMS + 10, 2, 1);
04577   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
04578   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04579       GTK_WIDGET (window->grid_algorithm));
```

```

04580
04581 // Creating the variable widgets
04582 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04583 gtk_widget_set_tooltip_text
04584 (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04585 window->id_variable = g_signal_connect
04586 (window->combo_variable, "changed", window_set_variable, NULL);
04587 window->button_add_variable
04588 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04589 GTK_ICON_SIZE_BUTTON);
04590 g_signal_connect
04591 (window->button_add_variable, "clicked",
window_add_variable, NULL);
04592 gtk_widget_set_tooltip_text
04593 (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
04594 window->button_remove_variable
04595 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04596 GTK_ICON_SIZE_BUTTON);
04597 g_signal_connect
04598 (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
04599 gtk_widget_set_tooltip_text
04600 (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04601 window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04602 window->entry_variable = (GtkEntry *) gtk_entry_new ();
04603 gtk_widget_set_tooltip_text
04604 (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04605 window->id_variable_label = g_signal_connect
04606 (window->entry_variable, "changed", window_label_variable, NULL);
04607 window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04608 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04609 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04610 gtk_widget_set_tooltip_text
04611 (GTK_WIDGET (window->spin_min),
04612 gettext ("Minimum initial value of the variable"));
04613 window->scrolled_min
04614 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04615 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04616 GTK_WIDGET (window->spin_min));
04617 g_signal_connect (window->spin_min, "value-changed",
04618 window_rangemin_variable, NULL);
04619 window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04620 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04621 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04622 gtk_widget_set_tooltip_text
04623 (GTK_WIDGET (window->spin_max),
04624 gettext ("Maximum initial value of the variable"));
04625 window->scrolled_max
04626 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04627 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04628 GTK_WIDGET (window->spin_max));
04629 g_signal_connect (window->spin_max, "value-changed",
04630 window_rangemax_variable, NULL);
04631 window->check_minabs = (GtkCheckButton *)
04632 gtk_check_button_new_with_mnemonic (gettext ("Absolute minimum"));
04633 g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04634 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04635 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04636 gtk_widget_set_tooltip_text
04637 (GTK_WIDGET (window->spin_minabs),
04638 gettext ("Minimum allowed value of the variable"));
04639 window->scrolled_minabs
04640 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04641 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04642 GTK_WIDGET (window->spin_minabs));
04643 g_signal_connect (window->spin_minabs, "value-changed",
04644 window_rangeminabs_variable, NULL);
04645 window->check_maxabs = (GtkCheckButton *)
04646 gtk_check_button_new_with_mnemonic (gettext ("Absolute maximum"));
04647 g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04648 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04649 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04650 gtk_widget_set_tooltip_text
04651 (GTK_WIDGET (window->spin_maxabs),
04652 gettext ("Maximum allowed value of the variable"));
04653 window->scrolled_maxabs
04654 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04655 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04656 GTK_WIDGET (window->spin_maxabs));
04657 g_signal_connect (window->spin_maxabs, "value-changed",
04658 window_rangemaxabs_variable, NULL);
04659 window->label_precision
04660 = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04661 window->spin_precision = (GtkSpinButton *)
04662 gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04663 gtk_widget_set_tooltip_text
04664 (GTK_WIDGET (window->spin_precision),

```

```

04665     gettext ("Number of precision floating point digits\n"
04666              "0 is for integer numbers"));
04667     g_signal_connect (window->spin_precision, "value-changed",
04668                      window_precision_variable, NULL);
04669     window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
04670     window->spin_sweeps
04671     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04672     gtk_widget_set_tooltip_text
04673     (GTK_WIDGET (window->spin_sweeps),
04674      gettext ("Number of steps sweeping the variable"));
04675     g_signal_connect
04676     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
04677     window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
04678     window->spin_bits
04679     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
04680     gtk_widget_set_tooltip_text
04681     (GTK_WIDGET (window->spin_bits),
04682      gettext ("Number of bits to encode the variable"));
04683     g_signal_connect
04684     (window->spin_bits, "value-changed", window_update_variable, NULL);
04685     window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
04686     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
04687     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04688     gtk_widget_set_tooltip_text
04689     (GTK_WIDGET (window->spin_step),
04690      gettext ("Initial step size for the gradient based method"));
04691     window->scrolled_step
04692     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04693     gtk_container_add (GTK_CONTAINER (window->scrolled_step),
04694                       GTK_WIDGET (window->spin_step));
04695     g_signal_connect
04696     (window->spin_step, "value-changed", window_step_variable, NULL);
04697     window->grid_variable = (GtkGrid *) gtk_grid_new ();
04698     gtk_grid_attach (window->grid_variable,
04699                     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
04700     gtk_grid_attach (window->grid_variable,
04701                     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
04702     gtk_grid_attach (window->grid_variable,
04703                     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
04704     gtk_grid_attach (window->grid_variable,
04705                     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
04706     gtk_grid_attach (window->grid_variable,
04707                     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
04708     gtk_grid_attach (window->grid_variable,
04709                     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
04710     gtk_grid_attach (window->grid_variable,
04711                     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04712     gtk_grid_attach (window->grid_variable,
04713                     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04714     gtk_grid_attach (window->grid_variable,
04715                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04716     gtk_grid_attach (window->grid_variable,
04717                     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
04718     gtk_grid_attach (window->grid_variable,
04719                     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
04720     gtk_grid_attach (window->grid_variable,
04721                     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04722     gtk_grid_attach (window->grid_variable,
04723                     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
04724     gtk_grid_attach (window->grid_variable,
04725                     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
04726     gtk_grid_attach (window->grid_variable,
04727                     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
04728     gtk_grid_attach (window->grid_variable,
04729                     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04730     gtk_grid_attach (window->grid_variable,
04731                     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04732     gtk_grid_attach (window->grid_variable,
04733                     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04734     gtk_grid_attach (window->grid_variable,
04735                     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04736     gtk_grid_attach (window->grid_variable,
04737                     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
04738     gtk_grid_attach (window->grid_variable,
04739                     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
04740     window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
04741     gtk_container_add (GTK_CONTAINER (window->frame_variable),
04742                       GTK_WIDGET (window->grid_variable));
04743
04744     // Creating the experiment widgets
04745     window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
04746     gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
04747                                 gettext ("Experiment selector"));
04748     window->id_experiment = g_signal_connect
04749     (window->combo_experiment, "changed", window_set_experiment, NULL)
04750 ;
04751     window->button_add_experiment

```

```

04751     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04752                                                    GTK_ICON_SIZE_BUTTON);
04753     g_signal_connect
04754     (window->button_add_experiment, "clicked",
04755      window_add_experiment, NULL);
04756     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04757                                  gettext ("Add experiment"));
04758     window->button_remove_experiment
04759     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04760                                                    GTK_ICON_SIZE_BUTTON);
04761     g_signal_connect (window->button_remove_experiment, "clicked",
04762                       window_remove_experiment, NULL);
04763     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04764                                  gettext ("Remove experiment"));
04765     window->label_experiment
04766     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04767     window->button_experiment = (GtkFileChooserButton *)
04768     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04769                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04770     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04771                                  gettext ("Experimental data file"));
04772     window->id_experiment_name
04773     = g_signal_connect (window->button_experiment, "selection-changed",
04774                       window_name_experiment, NULL);
04775     window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
04776     window->spin_weight
04777     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04778     gtk_widget_set_tooltip_text
04779     (GTK_WIDGET (window->spin_weight),
04780      gettext ("Weight factor to build the objective function"));
04781     g_signal_connect
04782     (window->spin_weight, "value-changed", window_weight_experiment,
04783      NULL);
04784     window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04785     gtk_grid_attach (window->grid_experiment,
04786                     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
04787     gtk_grid_attach (window->grid_experiment,
04788                     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
04789     gtk_grid_attach (window->grid_experiment,
04790                     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
04791     gtk_grid_attach (window->grid_experiment,
04792                     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
04793     gtk_grid_attach (window->grid_experiment,
04794                     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04795     gtk_grid_attach (window->grid_experiment,
04796                     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04797     for (i = 0; i < MAX_NINPUS; ++i)
04798     {
04799         snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04800         window->check_template[i] = (GtkCheckButton *)
04801         gtk_check_button_new_with_label (buffer3);
04802         window->id_template[i]
04803         = g_signal_connect (window->check_template[i], "toggled",
04804                             window_inputs_experiment, NULL);
04805         gtk_grid_attach (window->grid_experiment,
04806                         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
04807         window->button_template[i] = (GtkFileChooserButton *)
04808         gtk_file_chooser_button_new (gettext ("Input template"),
04809                                     GTK_FILE_CHOOSER_ACTION_OPEN);
04810         gtk_widget_set_tooltip_text
04811         (GTK_WIDGET (window->button_template[i]),
04812          gettext ("Experimental input template file"));
04813         window->id_input[i]
04814         = g_signal_connect_swapped (window->button_template[i],
04815                                     "selection-changed",
04816                                     (void (*)(void *)) window_template_experiment,
04817                                     (void *) (size_t) i);
04818         gtk_grid_attach (window->grid_experiment,
04819                         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
04820     }
04821     window->frame_experiment
04822     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04823     gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04824                       GTK_WIDGET (window->grid_experiment));
04825
04826     // Creating the grid and attaching the widgets to the grid
04827     window->grid = (GtkGrid *) gtk_grid_new ();
04828     gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04829     gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04830     gtk_grid_attach (window->grid,
04831                     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04832     gtk_grid_attach (window->grid,
04833                     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04834     gtk_grid_attach (window->grid,
04835                     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);

```

```

04836  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));
04837
04838  // Setting the window logo
04839  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04840  gtk_window_set_icon (window->window, window->logo);
04841
04842  // Showing the window
04843  gtk_widget_show_all (GTK_WIDGET (window->window));
04844
04845  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04846  #if GTK_MINOR_VERSION >= 16
04847  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04848  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04849  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04850  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04851  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
04852  #endif
04853
04854  // Reading initial example
04855  input_new ();
04856  buffer2 = g_get_current_dir ();
04857  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04858  g_free (buffer2);
04859  window_read (buffer);
04860  g_free (buffer);
04861
04862  #if DEBUG
04863  fprintf (stderr, "window_new: start\n");
04864  #endif
04865  }
04866
04867  #endif
04868
04874  int
04875  cores_number ()
04876  {
04877  #ifdef G_OS_WIN32
04878  SYSTEM_INFO sysinfo;
04879  GetSystemInfo (&sysinfo);
04880  return sysinfo.dwNumberOfProcessors;
04881  #else
04882  return (int) sysconf (_SC_NPROCESSORS_ONLN);
04883  #endif
04884  }
04885
04895  int
04896  main (int argn, char **argc)
04897  {
04898  #if HAVE_GTK
04899  char *buffer;
04900  #endif
04901
04902  // Starting pseudo-random numbers generator
04903  calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04904  calibrate->seed = DEFAULT_RANDOM_SEED;
04905
04906  // Allowing spaces in the XML data file
04907  xmlKeepBlanksDefault (0);
04908
04909  // Starting MPI
04910  #if HAVE_MPI
04911  MPI_Init (&argn, &argc);
04912  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04913  MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04914  printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04915  #else
04916  ntasks = 1;
04917  #endif
04918
04919  #if HAVE_GTK
04920
04921  // Getting threads number
04922  nthreads_gradient = nthreads = cores_number ();
04923
04924  // Setting local language and international floating point numbers notation
04925  setlocale (LC_ALL, "");
04926  setlocale (LC_NUMERIC, "C");
04927  window->application_directory = g_get_current_dir ();
04928  buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
04929  bindtextdomain (PROGRAM_INTERFACE, buffer);
04930  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04931  textdomain (PROGRAM_INTERFACE);
04932
04933  // Initing GTK+
04934  gtk_disable_setlocale ();

```

```

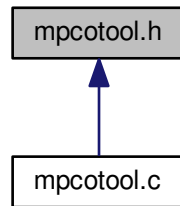
04935  gtk_init (&argn, &argc);
04936
04937  // Opening the main window
04938  window_new ();
04939  gtk_main ();
04940
04941  // Freeing memory
04942  input_free ();
04943  g_free (buffer);
04944  gtk_widget_destroy (GTK_WIDGET (window->window));
04945  g_free (window->application_directory);
04946
04947  #else
04948
04949  // Checking syntax
04950  if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04951  {
04952      printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04953      return 1;
04954  }
04955
04956  // Getting threads number
04957  if (argn == 2)
04958      nthreads_gradient = nthreads = cores_number ();
04959  else
04960  {
04961      nthreads_gradient = nthreads = atoi (argc[2]);
04962      if (!nthreads)
04963      {
04964          printf ("Bad threads number\n");
04965          return 2;
04966      }
04967  }
04968  printf ("nthreads=%u\n", nthreads);
04969
04970  // Making calibration
04971  if (input_open (argc[argn - 1]))
04972      calibrate_open ();
04973
04974  // Freeing memory
04975  calibrate_free ();
04976
04977  #endif
04978
04979  // Closing MPI
04980  #if HAVE_MPI
04981  MPI_Finalize ();
04982  #endif
04983
04984  // Freeing memory
04985  gsl_rng_free (calibrate->rng);
04986
04987  // Closing
04988  return 0;
04989 }

```

## 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 to define the algorithms.*
- enum [GradientMethod](#) { [GRADIENT\\_METHOD\\_COORDINATES](#) = 0, [GRADIENT\\_METHOD\\_RANDOM](#) = 1 }  
*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.*
- unsigned int [xml\\_node\\_get\\_uint\\_with\\_default](#) (xmlNode \*node, const xmlChar \*prop, unsigned int default\_value, int \*error\_code)  
*Function to get an unsigned integer number of a XML node property with a default value.*
- double [xml\\_node\\_get\\_float](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get a floating point number of a XML node property.*
- double [xml\\_node\\_get\\_float\\_with\\_default](#) (xmlNode \*node, const xmlChar \*prop, double default\_value, int \*error\_code)

- Function to get a floating point number of a XML node property with a default value.*

  - void `xml_node_set_int` (xmlNode \*node, const xmlChar \*prop, int value)
- Function to set an integer number in a XML node property.*

  - void `xml_node_set_uint` (xmlNode \*node, const xmlChar \*prop, unsigned int value)
- Function to set an unsigned integer number in a XML node property.*

  - void `xml_node_set_float` (xmlNode \*node, const xmlChar \*prop, double value)
- Function to set a floating point number in a XML node property.*

  - 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 1443 of file `mpcotool.c`.

```

01444 {
01445     unsigned int i, j;
01446     double e;
01447     #if DEBUG
01448         fprintf (stderr, "calibrate_best: start\n");
01449         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01450                 calibrate->nsaveds, calibrate->nbest);
01451     #endif
01452     if (calibrate->nsaveds < calibrate->nbest
01453         || value < calibrate->error_best[calibrate->nsaveds - 1])
01454     {
01455         if (calibrate->nsaveds < calibrate->nbest)
01456             ++calibrate->nsaveds;
01457         calibrate->error_best[calibrate->nsaveds - 1] = value;
01458         calibrate->simulation_best[calibrate->
01459             nsaveds - 1] = simulation;
01460         for (i = calibrate->nsaveds; --i;)
01461         {
01462             if (calibrate->error_best[i] < calibrate->
01463                 error_best[i - 1])
01464             {
01465                 j = calibrate->simulation_best[i];
01466                 e = calibrate->error_best[i];
01467                 calibrate->simulation_best[i] = calibrate->
01468                     simulation_best[i - 1];
01469                 calibrate->error_best[i] = calibrate->
01470                     error_best[i - 1];
01471                 calibrate->simulation_best[i - 1] = j;
01472                 calibrate->error_best[i - 1] = e;
01473             }
01474             else
01475                 break;
01476         }
01477     }
01478     #if DEBUG
01479         fprintf (stderr, "calibrate_best: end\n");
01480     #endif
01481 }

```

#### 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 1756 of file `mpcotool.c`.

```

01757 {
01758     #if DEBUG
01759         fprintf (stderr, "calibrate_best_gradient: start\n");
01760         fprintf (stderr,
01761                 "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01762                 simulation, value, calibrate->error_best[0]);
01763     #endif
01764     if (value < calibrate->error_best[0])
01765     {
01766         calibrate->error_best[0] = value;
01767         calibrate->simulation_best[0] = simulation;
01768     }
01769     #if DEBUG
01770         fprintf (stderr,

```

```

01770             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01771             simulation, value);
01772 #endif
01773     }
01774 #if DEBUG
01775     fprintf (stderr, "calibrate_best_gradient: end\n");
01776 #endif
01777 }

```

### 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.

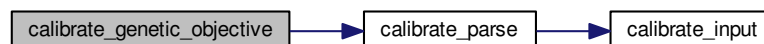
Definition at line 2059 of file [mpcotool.c](#).

```

02060 {
02061     unsigned int j;
02062     double objective;
02063     char buffer[64];
02064 #if DEBUG
02065     fprintf (stderr, "calibrate_genetic_objective: start\n");
02066 #endif
02067     for (j = 0; j < calibrate->nvariables; ++j)
02068     {
02069         calibrate->value[entity->id * calibrate->nvariables + j]
02070             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02071     }
02072     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02073         objective += calibrate_parse (entity->id, j);
02074     g_mutex_lock (mutex);
02075     for (j = 0; j < calibrate->nvariables; ++j)
02076     {
02077         snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02078         fprintf (calibrate->file_variables, buffer,
02079             genetic_get_variable (entity, calibrate->
02080                 genetic_variable + j));
02081     }
02082     fprintf (calibrate->file_variables, "%.14le\n", objective);
02083     g_mutex_unlock (mutex);
02084 #if DEBUG
02085     fprintf (stderr, "calibrate_genetic_objective: end\n");
02086 #endif
02087     return objective;
02088 }

```

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

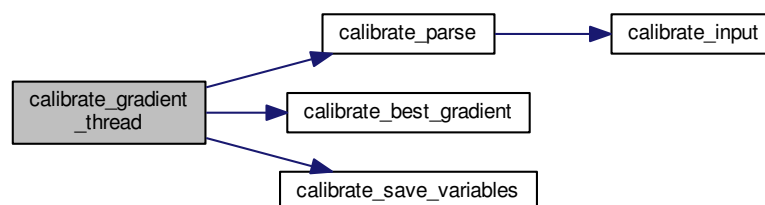
Definition at line 1821 of file [mpcotool.c](#).

```

01822 {
01823     unsigned int i, j, thread;
01824     double e;
01825     #if DEBUG
01826     fprintf (stderr, "calibrate_gradient_thread: start\n");
01827     #endif
01828     thread = data->thread;
01829     #if DEBUG
01830     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01831             thread,
01832             calibrate->thread_gradient[thread],
01833             calibrate->thread_gradient[thread + 1]);
01834     #endif
01835     for (i = calibrate->thread_gradient[thread];
01836          i < calibrate->thread_gradient[thread + 1]; ++i)
01837     {
01838         e = 0.;
01839         for (j = 0; j < calibrate->nexperiments; ++j)
01840             e += calibrate_parse (i, j);
01841         g_mutex_lock (mutex);
01842         calibrate_best_gradient (i, e);
01843         calibrate_save_variables (i, e);
01844         g_mutex_unlock (mutex);
01845     #if DEBUG
01846     fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01847     #endif
01848     }
01849     #if DEBUG
01850     fprintf (stderr, "calibrate_gradient_thread: end\n");
01851     #endif
01852     g_thread_exit (NULL);
01853     return NULL;
01854 }

```

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 1196 of file [mpcotool.c](#).

```

01197 {
01198     unsigned int i;
01199     char buffer[32], value[32], *buffer2, *buffer3, *content;
01200     FILE *file;
01201     gsize length;
01202     GRegex *regex;
01203
01204     #if DEBUG
01205         fprintf (stderr, "calibrate_input: start\n");
01206     #endif
01207
01208     // Checking the file
01209     if (!template)
01210         goto calibrate_input_end;
01211
01212     // Opening template
01213     content = g_mapped_file_get_contents (template);
01214     length = g_mapped_file_get_length (template);
01215     #if DEBUG
01216         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01217                 content);
01218     #endif
01219     file = g_fopen (input, "w");
01220
01221     // Parsing template
01222     for (i = 0; i < calibrate->nvariables; ++i)
01223     {
01224         #if DEBUG
01225             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01226         #endif
01227         snprintf (buffer, 32, "@variable%u@", i + 1);
01228         regex = g_regex_new (buffer, 0, 0, NULL);
01229         if (i == 0)
01230         {
01231             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01232                                               calibrate->label[i], 0, NULL);
01233         }
01234         #if DEBUG
01235             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01236         #endif
01237         else
01238         {
01239             length = strlen (buffer3);
01240             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01241                                               calibrate->label[i], 0, NULL);
01242             g_free (buffer3);
01243         }
01244         g_regex_unref (regex);
01245         length = strlen (buffer2);
01246         snprintf (buffer, 32, "@value%u@", i + 1);
01247         regex = g_regex_new (buffer, 0, 0, NULL);
01248         snprintf (value, 32, format[calibrate->precision[i]],
01249                 calibrate->value[simulation * calibrate->
01250 nvariables + i]);
01251         #if DEBUG
01252             fprintf (stderr, "calibrate_input: value=%s\n", value);
01253         #endif
01254         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01255                                           0, NULL);
01256         g_free (buffer2);
01257         g_regex_unref (regex);
01258     }
01259
01260     // Saving input file
01261     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01262     g_free (buffer3);
01263     fclose (file);
01264
01265     calibrate_input_end:
01266     #if DEBUG
01267         fprintf (stderr, "calibrate_input: end\n");
01268     #endif
01269     return;
01270 }

```

### 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 1561 of file [mpcotool.c](#).

```

01563 {
01564     unsigned int i, j, k, s[calibrate->nbest];
01565     double e[calibrate->nbest];
01566     #if DEBUG
01567     fprintf (stderr, "calibrate_merge: start\n");
01568     #endif
01569     i = j = k = 0;
01570     do
01571     {
01572         if (i == calibrate->nsaveds)
01573         {
01574             s[k] = simulation_best[j];
01575             e[k] = error_best[j];
01576             ++j;
01577             ++k;
01578             if (j == nsaveds)
01579                 break;
01580         }
01581         else if (j == nsaveds)
01582         {
01583             s[k] = calibrate->simulation_best[i];
01584             e[k] = calibrate->error_best[i];
01585             ++i;
01586             ++k;
01587             if (i == calibrate->nsaveds)
01588                 break;
01589         }
01590         else if (calibrate->error_best[i] > error_best[j])
01591         {
01592             s[k] = simulation_best[j];
01593             e[k] = error_best[j];
01594             ++j;
01595             ++k;
01596         }
01597         else
01598         {
01599             s[k] = calibrate->simulation_best[i];
01600             e[k] = calibrate->error_best[i];
01601             ++i;
01602             ++k;
01603         }
01604     }
01605     while (k < calibrate->nbest);
01606     calibrate->nsaveds = k;
01607     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01608     memcpy (calibrate->error_best, e, k * sizeof (double));
01609     #if DEBUG
01610     fprintf (stderr, "calibrate_merge: end\n");
01611     #endif
01612 }
```

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

## Returns

Objective function value.

Definition at line 1283 of file [mpcotool.c](#).

```

01284 {
01285     unsigned int i;
01286     double e;
01287     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01288         *buffer3, *buffer4;
01289     FILE *file_result;
01290
01291     #if DEBUG
01292         fprintf (stderr, "calibrate_parse: start\n");
01293         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01294             experiment);
01295     #endif
01296
01297     // Opening input files
01298     for (i = 0; i < calibrate->ninputs; ++i)
01299     {
01300         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01301     #if DEBUG
01302         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01303     #endif
01304         calibrate_input (simulation, &input[i][0],
01305             calibrate->file[i][experiment]);
01306     }
01307     for (; i < MAX_NINPUTS; ++i)
01308         strcpy (&input[i][0], "");
01309     #if DEBUG
01310         fprintf (stderr, "calibrate_parse: parsing end\n");
01311     #endif
01312
01313     // Performing the simulation
01314     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01315     buffer2 = g_path_get_dirname (calibrate->simulator);
01316     buffer3 = g_path_get_basename (calibrate->simulator);
01317     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01318     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
01319         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01320         input[6], input[7], output);
01321     g_free (buffer4);
01322     g_free (buffer3);
01323     g_free (buffer2);
01324     #if DEBUG
01325         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01326     #endif
01327     system (buffer);
01328
01329     // Checking the objective value function
01330     if (calibrate->evaluator)
01331     {
01332         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01333         buffer2 = g_path_get_dirname (calibrate->evaluator);
01334         buffer3 = g_path_get_basename (calibrate->evaluator);
01335         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01336         snprintf (buffer, 512, "%s\ " %s %s %s",
01337             buffer4, output, calibrate->experiment[experiment], result);
01338         g_free (buffer4);
01339         g_free (buffer3);
01340         g_free (buffer2);
01341     #if DEBUG
01342         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01343     #endif
01344         system (buffer);
01345         file_result = g_fopen (result, "r");
01346         e = atof (fgets (buffer, 512, file_result));
01347         fclose (file_result);
01348     }
01349     else
01350     {
01351         strcpy (result, "");
01352         file_result = g_fopen (output, "r");
01353         e = atof (fgets (buffer, 512, file_result));
01354         fclose (file_result);
01355     }
01356
01357     // Removing files
01358     #if !DEBUG
01359     for (i = 0; i < calibrate->ninputs; ++i)
01360     {
01361         if (calibrate->file[i][0])
01362         {

```

```

01363         snprintf (buffer, 512, RM " %s", &input[i][0]);
01364         system (buffer);
01365     }
01366 }
01367 snprintf (buffer, 512, RM " %s %s", output, result);
01368 system (buffer);
01369 #endif
01370
01371 #if DEBUG
01372 fprintf (stderr, "calibrate_parse: end\n");
01373 #endif
01374
01375 // Returning the objective function
01376 return e * calibrate->weight[experiment];
01377 }

```

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 1415 of file [mpcotool.c](#).

```

01416 {
01417     unsigned int i;
01418     char buffer[64];
01419     #if DEBUG
01420     fprintf (stderr, "calibrate_save_variables: start\n");
01421     #endif
01422     for (i = 0; i < calibrate->nvariables; ++i)
01423     {
01424         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01425         fprintf (calibrate->file_variables, buffer,
01426                 calibrate->value[simulation * calibrate->
01427                               nvariables + i]);
01428     }
01429     fprintf (calibrate->file_variables, "%.14le\n", error);
01430     #if DEBUG
01431     fprintf (stderr, "calibrate_save_variables: end\n");
01432     #endif
01433 }

```

#### 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.
-------------------	--------------------

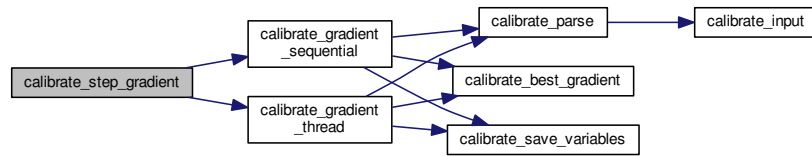
Definition at line 1923 of file `mpcotool.c`.

```

01924 {
01925     GThread *thread[nthreads_gradient];
01926     ParallelData data[nthreads_gradient];
01927     unsigned int i, j, k, b;
01928     #if DEBUG
01929     fprintf (stderr, "calibrate_step_gradient: start\n");
01930     #endif
01931     for (i = 0; i < calibrate->nestimates; ++i)
01932     {
01933         k = (simulation + i) * calibrate->nvariables;
01934         b = calibrate->simulation_best[0] * calibrate->
nvariables;
01935         #if DEBUG
01936         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01937                 simulation + i, calibrate->simulation_best[0]);
01938         #endif
01939         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01940         {
01941             #if DEBUG
01942             fprintf (stderr,
01943                     "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01944                     i, j, calibrate->value[b]);
01945             #endif
01946             calibrate->value[k]
01947             = calibrate->value[b] + calibrate_estimate_gradient (j
, i);
01948             calibrate->value[k] = fmin (fmax (calibrate->
value[k],
01949                                             calibrate->rangeminabs[j]),
01950                                         calibrate->rangemaxabs[j]);
01951             #if DEBUG
01952             fprintf (stderr,
01953                     "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01954                     i, j, calibrate->value[k]);
01955             #endif
01956         }
01957     }
01958     if (nthreads_gradient == 1)
01959         calibrate_gradient_sequential (simulation);
01960     else
01961     {
01962         for (i = 0; i <= nthreads_gradient; ++i)
01963         {
01964             calibrate->thread_gradient[i]
01965             = simulation + calibrate->nstart_gradient
01966             + i * (calibrate->nend_gradient - calibrate->
nstart_gradient)
01967             / nthreads_gradient;
01968             #if DEBUG
01969             fprintf (stderr,
01970                     "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01971                     i, calibrate->thread_gradient[i]);
01972             #endif
01973         }
01974         for (i = 0; i < nthreads_gradient; ++i)
01975         {
01976             data[i].thread = i;
01977             thread[i] = g_thread_new
01978             (NULL, (void (*) ) calibrate_gradient_thread, &data[i]);
01979         }
01980         for (i = 0; i < nthreads_gradient; ++i)
01981             g_thread_join (thread[i]);
01982     }
01983     #if DEBUG
01984     fprintf (stderr, "calibrate_step_gradient: end\n");
01985     #endif
01986 }

```

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

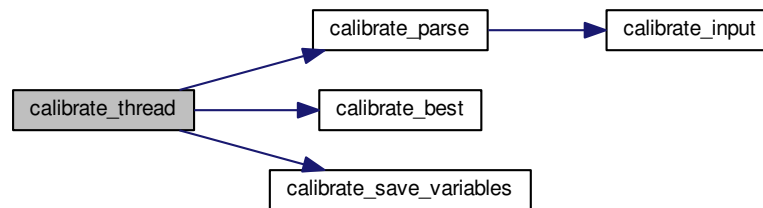
Definition at line 1517 of file [mpcotool.c](#).

```

01518 {
01519     unsigned int i, j, thread;
01520     double e;
01521     #if DEBUG
01522     fprintf (stderr, "calibrate_thread: start\n");
01523     #endif
01524     thread = data->thread;
01525     #if DEBUG
01526     fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01527             calibrate->thread[thread], calibrate->thread[thread + 1]);
01528     #endif
01529     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01530     {
01531         e = 0.;
01532         for (j = 0; j < calibrate->nexperiments; ++j)
01533             e += calibrate_parse (i, j);
01534         g_mutex_lock (mutex);
01535         calibrate_best (i, e);
01536         calibrate_save_variables (i, e);
01537         g_mutex_unlock (mutex);
01538     #if DEBUG
01539     fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01540     #endif
01541     }
01542     #if DEBUG
01543     fprintf (stderr, "calibrate_thread: end\n");
01544     #endif
01545     g_thread_exit (NULL);
01546     return NULL;
01547 }

```

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 548 of file `mpcotool.c`.

```

00549 {
00550     char buffer2[64];
00551     char *buffert[MAX_NINPUTS] =
00552         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00553     xmlDoc *doc;
00554     xmlNode *node, *child;
00555     xmlChar *buffer;
00556     char *msg;
00557     int error_code;
00558     unsigned int i;
00559
00560     #if DEBUG
00561     fprintf (stderr, "input_open: start\n");
00562     #endif
00563
00564     // Resetting input data
00565     buffer = NULL;
00566     input_new ();
00567
00568     // Parsing the input file
00569     #if DEBUG
00570     fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00571     #endif
00572     doc = xmlParseFile (filename);
00573     if (!doc)
00574     {
00575         msg = gettext ("Unable to parse the input file");
00576         goto exit_on_error;
00577     }
00578
00579     // Getting the root node
00580     #if DEBUG
00581     fprintf (stderr, "input_open: getting the root node\n");
00582     #endif
00583     node = xmlDocGetRootElement (doc);
00584     if (xmlStrcmp (node->name, XML_CALIBRATE))
00585     {
00586         msg = gettext ("Bad root XML node");
00587         goto exit_on_error;
00588     }
00589

```

```

00590 // Getting results file names
00591 input->result = (char *) xmlGetProp (node, XML_RESULT);
00592 if (!input->result)
00593     input->result = (char *) xmlStrdup (result_name);
00594 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00595 if (!input->variables)
00596     input->variables = (char *) xmlStrdup (variables_name);
00597
00598 // Opening simulator program name
00599 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00600 if (!input->simulator)
00601 {
00602     msg = gettext ("Bad simulator program");
00603     goto exit_on_error;
00604 }
00605
00606 // Opening evaluator program name
00607 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00608
00609 // Obtaining pseudo-random numbers generator seed
00610 input->seed
00611     = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00612                                     &error_code);
00613 if (error_code)
00614 {
00615     msg = gettext ("Bad pseudo-random numbers generator seed");
00616     goto exit_on_error;
00617 }
00618
00619 // Opening algorithm
00620 buffer = xmlGetProp (node, XML_ALGORITHM);
00621 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00622 {
00623     input->algorithm = ALGORITHM_MONTE_CARLO;
00624
00625     // Obtaining simulations number
00626     input->nsimulations
00627         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00628     if (error_code)
00629     {
00630         msg = gettext ("Bad simulations number");
00631         goto exit_on_error;
00632     }
00633 }
00634 else if (!xmlStrcmp (buffer, XML_SWEEP))
00635     input->algorithm = ALGORITHM_SWEEP;
00636 else if (!xmlStrcmp (buffer, XML_GENETIC))
00637 {
00638     input->algorithm = ALGORITHM_GENETIC;
00639
00640     // Obtaining population
00641     if (xmlHasProp (node, XML_NPOPULATION))
00642     {
00643         input->nsimulations
00644             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00645         if (error_code || input->nsimulations < 3)
00646         {
00647             msg = gettext ("Invalid population number");
00648             goto exit_on_error;
00649         }
00650     }
00651     else
00652     {
00653         msg = gettext ("No population number");
00654         goto exit_on_error;
00655     }
00656
00657     // Obtaining generations
00658     if (xmlHasProp (node, XML_NGENERATIONS))
00659     {
00660         input->niterations
00661             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00662         if (error_code || !input->niterations)
00663         {
00664             msg = gettext ("Invalid generations number");
00665             goto exit_on_error;
00666         }
00667     }
00668     else
00669     {
00670         msg = gettext ("No generations number");
00671         goto exit_on_error;
00672     }
00673
00674     // Obtaining mutation probability
00675     if (xmlHasProp (node, XML_MUTATION))

```

```

00676     {
00677         input->mutation_ratio
00678         = xml_node_get_float (node, XML_MUTATION, &error_code);
00679         if (error_code || input->mutation_ratio < 0.
00680             || input->mutation_ratio >= 1.)
00681         {
00682             msg = gettext ("Invalid mutation probability");
00683             goto exit_on_error;
00684         }
00685     }
00686     else
00687     {
00688         msg = gettext ("No mutation probability");
00689         goto exit_on_error;
00690     }
00691
00692     // Obtaining reproduction probability
00693     if (xmlHasProp (node, XML_REPRODUCTION))
00694     {
00695         input->reproduction_ratio
00696         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00697         if (error_code || input->reproduction_ratio < 0.
00698             || input->reproduction_ratio >= 1.0)
00699         {
00700             msg = gettext ("Invalid reproduction probability");
00701             goto exit_on_error;
00702         }
00703     }
00704     else
00705     {
00706         msg = gettext ("No reproduction probability");
00707         goto exit_on_error;
00708     }
00709
00710     // Obtaining adaptation probability
00711     if (xmlHasProp (node, XML_ADAPTATION))
00712     {
00713         input->adaptation_ratio
00714         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00715         if (error_code || input->adaptation_ratio < 0.
00716             || input->adaptation_ratio >= 1.)
00717         {
00718             msg = gettext ("Invalid adaptation probability");
00719             goto exit_on_error;
00720         }
00721     }
00722     else
00723     {
00724         msg = gettext ("No adaptation probability");
00725         goto exit_on_error;
00726     }
00727
00728     // Checking survivals
00729     i = input->mutation_ratio * input->nsimulations;
00730     i += input->reproduction_ratio * input->
00731 nsimulations;
00732     i += input->adaptation_ratio * input->
00733 nsimulations;
00734     if (i > input->nsimulations - 2)
00735     {
00736         msg = gettext
00737             ("No enough survival entities to reproduce the population");
00738         goto exit_on_error;
00739     }
00740     else
00741     {
00742         msg = gettext ("Unknown algorithm");
00743         goto exit_on_error;
00744     }
00745     xmlFree (buffer);
00746     buffer = NULL;
00747
00748     if (input->algorithm == ALGORITHM_MONTE_CARLO
00749         || input->algorithm == ALGORITHM_SWEEP)
00750     {
00751         // Obtaining iterations number
00752         input->niterations
00753         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00754         if (error_code == 1)
00755             input->niterations = 1;
00756         else if (error_code)
00757         {
00758             msg = gettext ("Bad iterations number");
00759             goto exit_on_error;
00760         }
00761     }

```

```

00761
00762     // Obtaining best number
00763     input->nbest
00764     = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
00765     if (error_code || !input->nbest)
00766     {
00767         msg = gettext ("Invalid best number");
00768         goto exit_on_error;
00769     }
00770
00771     // Obtaining tolerance
00772     input->tolerance
00773     = xml_node_get_float_with_default (node,
XML_TOLERANCE, 0.,
00774                                     &error_code);
00775     if (error_code || input->tolerance < 0.)
00776     {
00777         msg = gettext ("Invalid tolerance");
00778         goto exit_on_error;
00779     }
00780
00781     // Getting gradient method parameters
00782     if (xmlHasProp (node, XML_NSTEPS))
00783     {
00784         input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
00785         if (error_code || !input->nsteps)
00786         {
00787             msg = gettext ("Invalid steps number");
00788             goto exit_on_error;
00789         }
00790         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00791         if (!xmlStrcmp (buffer, XML_COORDINATES))
00792             input->gradient_method =
GRADIENT_METHOD_COORDINATES;
00793         else if (!xmlStrcmp (buffer, XML_RANDOM))
00794         {
00795             input->gradient_method =
GRADIENT_METHOD_RANDOM;
00796             input->nestimates
00797             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00798             if (error_code || !input->nestimates)
00799             {
00800                 msg = gettext ("Invalid estimates number");
00801                 goto exit_on_error;
00802             }
00803         }
00804         else
00805         {
00806             msg = gettext ("Unknown method to estimate the gradient");
00807             goto exit_on_error;
00808         }
00809         xmlFree (buffer);
00810         buffer = NULL;
00811         input->relaxation
00812         = xml_node_get_float_with_default (node,
XML_RELAXATION,
00813                                     DEFAULT_RELAXATION, &error_code);
00814         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00815         {
00816             msg = gettext ("Invalid relaxation parameter");
00817             goto exit_on_error;
00818         }
00819     }
00820     else
00821         input->nsteps = 0;
00822 }
00823
00824 // Reading the experimental data
00825 for (child = node->children; child; child = child->next)
00826 {
00827     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00828         break;
00829 #if DEBUG
00830     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00831 #endif
00832     if (xmlHasProp (child, XML_NAME))
00833         buffer = xmlGetProp (child, XML_NAME);
00834     else
00835     {
00836         snprintf (buffer2, 64, "%s %u: %s",
00837                 gettext ("Experiment"),
00838                 input->nexperiments + 1, gettext ("no data file name"));
00839         msg = buffer2;
00840         goto exit_on_error;

```

```

00841     }
00842     #if DEBUG
00843     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00844     #endif
00845     input->weight = g_realloc (input->weight,
00846                               (1 + input->nexperiments) * sizeof (double));
00847     input->weight[input->nexperiments]
00848     = xml_node_get_float_with_default (child,
XML_WEIGHT, 1., &error_code);
00849     if (error_code)
00850     {
00851         snprintf (buffer2, 64, "%s %s: %s",
00852                 gettext ("Experiment"), buffer, gettext ("bad weight"));
00853         msg = buffer2;
00854         goto exit_on_error;
00855     }
00856     #if DEBUG
00857     fprintf (stderr, "input_open: weight=%lg\n",
00858             input->weight[input->nexperiments]);
00859     #endif
00860     if (!input->nexperiments)
00861         input->ninputs = 0;
00862     #if DEBUG
00863     fprintf (stderr, "input_open: template[0]\n");
00864     #endif
00865     if (xmlHasProp (child, XML_TEMPLATE1))
00866     {
00867         input->template[0]
00868         = (char **) g_realloc (input->template[0],
00869                               (1 + input->nexperiments) * sizeof (char *));
00870         buffert[0] = (char *) xmlGetProp (child, template[0]);
00871     #if DEBUG
00872     fprintf (stderr, "input_open: experiment=%u templatel=%s\n",
00873             input->nexperiments, buffert[0]);
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
00882     {
00883         snprintf (buffer2, 64, "%s %s: %s",
00884                 gettext ("Experiment"), buffer, gettext ("no template"));
00885         msg = buffer2;
00886         goto exit_on_error;
00887     }
00888     for (i = 1; i < MAX_NINPUTS; ++i)
00889     {
00890     #if DEBUG
00891     fprintf (stderr, "input_open: template%u\n", i + 1);
00892     #endif
00893     if (xmlHasProp (child, template[i]))
00894     {
00895         if (input->nexperiments && input->ninputs <= i)
00896         {
00897             snprintf (buffer2, 64, "%s %s: %s",
00898                     gettext ("Experiment"),
00899                     buffer, gettext ("bad templates number"));
00900             msg = buffer2;
00901             while (i-- > 0)
00902                 xmlFree (buffert[i]);
00903             goto exit_on_error;
00904         }
00905         input->template[i] = (char **)
00906             g_realloc (input->template[i],
00907                       (1 + input->nexperiments) * sizeof (char *));
00908         buffert[i] = (char *) xmlGetProp (child, template[i]);
00909     #if DEBUG
00910     fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00911             input->nexperiments, i + 1,
00912             input->template[i][input->nexperiments]);
00913     #endif
00914     if (!input->nexperiments)
00915         ++input->ninputs;
00916     #if DEBUG
00917     fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00918     #endif
00919     }
00920     else if (input->nexperiments && input->ninputs > i)
00921     {
00922         snprintf (buffer2, 64, "%s %s: %s%u",
00923                 gettext ("Experiment"),
00924                 buffer, gettext ("no template"), i + 1);
00925         msg = buffer2;
00926         while (i-- > 0)

```

```

00927         xmlFree (buffert[i]);
00928         goto exit_on_error;
00929     }
00930     else
00931         break;
00932 }
00933 input->experiment
00934 = g_realloc (input->experiment,
00935             (1 + input->nexperiments) * sizeof (char *));
00936 input->experiment[input->nexperiments] = (char *) buffer;
00937 for (i = 0; i < input->ninputs; ++i)
00938     input->template[i][input->nexperiments] = buffert[i];
00939 ++input->nexperiments;
00940 #if DEBUG
00941     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00942 #endif
00943 }
00944 if (!input->nexperiments)
00945 {
00946     msg = gettext ("No calibration experiments");
00947     goto exit_on_error;
00948 }
00949 buffer = NULL;
00950
00951 // Reading the variables data
00952 for (; child; child = child->next)
00953 {
00954     if (xmlStrcmp (child->name, XML_VARIABLE))
00955     {
00956         snprintf (buffer2, 64, "%s %u: %s",
00957                 gettext ("Variable"),
00958                 input->nvariables + 1, gettext ("bad XML node"));
00959         msg = buffer2;
00960         goto exit_on_error;
00961     }
00962     if (xmlHasProp (child, XML_NAME))
00963         buffer = xmlGetProp (child, XML_NAME);
00964     else
00965     {
00966         snprintf (buffer2, 64, "%s %u: %s",
00967                 gettext ("Variable"),
00968                 input->nvariables + 1, gettext ("no name"));
00969         msg = buffer2;
00970         goto exit_on_error;
00971     }
00972     if (xmlHasProp (child, XML_MINIMUM))
00973     {
00974         input->rangemin = g_realloc
00975             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00976         input->rangeminabs = g_realloc
00977             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00978         input->rangemin[input->nvariables]
00979             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00980         if (error_code)
00981         {
00982             snprintf (buffer2, 64, "%s %s: %s",
00983                     gettext ("Variable"), buffer, gettext ("bad minimum"));
00984             msg = buffer2;
00985             goto exit_on_error;
00986         }
00987         input->rangeminabs[input->nvariables]
00988             = xml_node_get_float_with_default (child,
00989 XML_ABSOLUTE_MINIMUM,
00990                                             -G_MAXDOUBLE, &error_code);
00991         if (error_code)
00992         {
00993             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00994                     gettext ("bad absolute minimum"));
00995             msg = buffer2;
00996             goto exit_on_error;
00997         }
00998         if (input->rangemin[input->nvariables]
00999             < input->rangeminabs[input->nvariables])
01000         {
01001             snprintf (buffer2, 64, "%s %s: %s",
01002                     gettext ("Variable"),
01003                     buffer, gettext ("minimum range not allowed"));
01004             msg = buffer2;
01005             goto exit_on_error;
01006         }
01007     }
01008     else
01009     {
01010         snprintf (buffer2, 64, "%s %s: %s",
01011                 gettext ("Variable"), buffer, gettext ("no minimum range"));
01012         msg = buffer2;
01013         goto exit_on_error;

```



```

01013     }
01014     if (xmlHasProp (child, XML_MAXIMUM))
01015     {
01016         input->rangemax = g_realloc
01017             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01018         input->rangemaxabs = g_realloc
01019             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01020         input->rangemax[input->nvariables]
01021             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01022         if (error_code)
01023         {
01024             snprintf (buffer2, 64, "%s %s: %s",
01025                 gettext ("Variable"), buffer, gettext ("bad maximum"));
01026             msg = buffer2;
01027             goto exit_on_error;
01028         }
01029         input->rangemaxabs[input->nvariables]
01030             = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
01031                 G_MAXDOUBLE, &error_code);
01032         if (error_code)
01033         {
01034             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01035                 gettext ("bad absolute maximum"));
01036             msg = buffer2;
01037             goto exit_on_error;
01038         }
01039         if (input->rangemax[input->nvariables]
01040             > input->rangemaxabs[input->nvariables])
01041         {
01042             snprintf (buffer2, 64, "%s %s: %s",
01043                 gettext ("Variable"),
01044                 buffer, gettext ("maximum range not allowed"));
01045             msg = buffer2;
01046             goto exit_on_error;
01047         }
01048     }
01049     else
01050     {
01051         snprintf (buffer2, 64, "%s %s: %s",
01052             gettext ("Variable"), buffer, gettext ("no maximum range"));
01053         msg = buffer2;
01054         goto exit_on_error;
01055     }
01056     if (input->rangemax[input->nvariables]
01057         < input->rangemin[input->nvariables])
01058     {
01059         snprintf (buffer2, 64, "%s %s: %s",
01060             gettext ("Variable"), buffer, gettext ("bad range"));
01061         msg = buffer2;
01062         goto exit_on_error;
01063     }
01064     input->precision = g_realloc
01065         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01066     input->precision[input->nvariables]
01067         = xml_node_get_uint_with_default (child,
XML_PRECISION,
01068             DEFAULT_PRECISION, &error_code);
01069     if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
01070     {
01071         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01072             gettext ("bad precision"));
01073         msg = buffer2;
01074         goto exit_on_error;
01075     }
01076     if (input->algorithm == ALGORITHM_SWEEP)
01077     {
01078         if (xmlHasProp (child, XML_NSWEEPS))
01079         {
01080             input->nsweeps = (unsigned int *)
01081                 g_realloc (input->nsweeps,
01082                     (1 + input->nvariables) * sizeof (unsigned int));
01083             input->nsweeps[input->nvariables]
01084                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01085             if (error_code || !input->nsweeps[input->
nvariables])
01086             {
01087                 snprintf (buffer2, 64, "%s %s: %s",
01088                     gettext ("Variable"),
01089                     buffer, gettext ("bad sweeps"));
01090                 msg = buffer2;
01091                 goto exit_on_error;
01092             }
01093         }
01094     }
01095     {

```

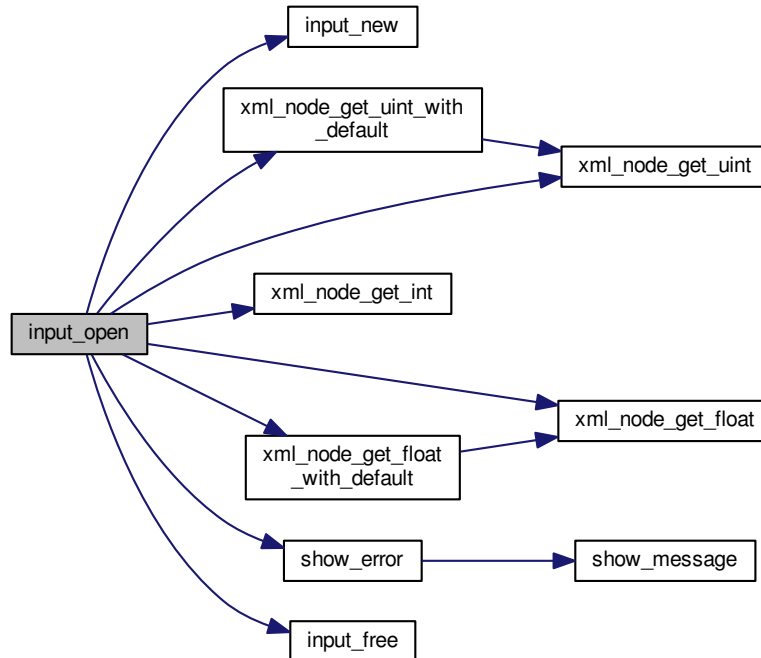
```

01096         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01097                     gettext ("no sweeps number"));
01098         msg = buffer2;
01099         goto exit_on_error;
01100     }
01101     #if DEBUG
01102     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01103             input->nsweeps[input->nvariables],
01104             input->nsimulations);
01105     #endif
01106     if (input->algorithm == ALGORITHM_GENETIC)
01107     {
01108         // Obtaining bits representing each variable
01109         if (xmlHasProp (child, XML_NBITS))
01110         {
01111             input->nbits = (unsigned int *)
01112                 g_realloc (input->nbits,
01113                             (1 + input->nvariables) * sizeof (unsigned int));
01114             i = xml_node_get_uint (child, XML_NBITS, &error_code);
01115             if (error_code || !i)
01116             {
01117                 snprintf (buffer2, 64, "%s %s: %s",
01118                             gettext ("Variable"),
01119                             buffer, gettext ("invalid bits number"));
01120                 msg = buffer2;
01121                 goto exit_on_error;
01122             }
01123             input->nbits[input->nvariables] = i;
01124         }
01125         else
01126         {
01127             snprintf (buffer2, 64, "%s %s: %s",
01128                         gettext ("Variable"),
01129                         buffer, gettext ("no bits number"));
01130             msg = buffer2;
01131             goto exit_on_error;
01132         }
01133     }
01134     else if (input->nsteps)
01135     {
01136         input->step = (double *)
01137             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01138         input->step[input->nvariables]
01139             = xml_node_get_float (child, XML_STEP, &error_code);
01140         if (error_code || input->step[input->nvariables] < 0.)
01141         {
01142             snprintf (buffer2, 64, "%s %s: %s",
01143                         gettext ("Variable"),
01144                         buffer, gettext ("bad step size"));
01145             msg = buffer2;
01146             goto exit_on_error;
01147         }
01148     }
01149     input->label = g_realloc
01150         (input->label, (1 + input->nvariables) * sizeof (char *));
01151     input->label[input->nvariables] = (char *) buffer;
01152     ++input->nvariables;
01153 }
01154 if (!input->nvariables)
01155 {
01156     msg = gettext ("No calibration variables");
01157     goto exit_on_error;
01158 }
01159 buffer = NULL;
01160
01161 // Getting the working directory
01162 input->directory = g_path_get_dirname (filename);
01163 input->name = g_path_get_basename (filename);
01164
01165 // Closing the XML document
01166 xmlFreeDoc (doc);
01167
01168 #if DEBUG
01169 fprintf (stderr, "input_open: end\n");
01170 #endif
01171 return 1;
01172
01173 exit_on_error:
01174 xmlFree (buffer);
01175 xmlFreeDoc (doc);
01176 show_error (msg);
01177 input_free ();
01178 #if DEBUG
01179 fprintf (stderr, "input_open: end\n");
01180 #endif
01181 return 0;

```

```
01182 }
```

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 366 of file [mpcotool.c](#).

```

00367 {
00368     double x = 0.;
00369     xmlChar *buffer;
00370     buffer = xmlGetProp (node, prop);
00371     if (!buffer)
00372         *error_code = 1;
00373     else
00374     {
00375         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00376             *error_code = 2;
00377         else
00378             *error_code = 0;
00379         xmlFree (buffer);
00380     }
00381     return x;
00382 }
```

#### 5.7.3.15 double xml\_node\_get\_float\_with\_default ( xmlNode \* node, const xmlChar \* prop, double default\_value, int \* error\_code )

Function to get a floating point number of a XML node property with a default value.

**Parameters**

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

**Returns**

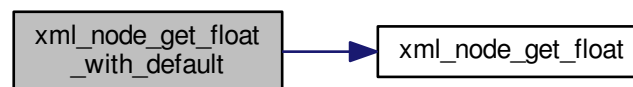
Floating point number value.

Definition at line 400 of file [mpcotool.c](#).

```

00402 {
00403     double x;
00404     if (xmlHasProp (node, prop))
00405         x = xml_node_get_float (node, prop, error_code);
00406     else
00407     {
00408         x = default_value;
00409         *error_code = 0;
00410     }
00411     return x;
00412 }
```

Here is the call graph for this function:



**5.7.3.16** `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 }

```

### 5.7.3.17 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 }

```

### 5.7.3.18 unsigned int xml\_node\_get\_uint\_with\_default ( xmlDoc \* node, const xmlChar \* prop, unsigned int default\_value, int \* error\_code )

Function to get an unsigned integer number of a XML node property with a default value.

#### Parameters

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

#### Returns

Unsigned integer number value.

Definition at line 339 of file [mpcotool.c](#).

```

00341 {
00342     unsigned int i;
00343     if (xmlHasProp (node, prop))
00344         i = xml_node_get_uint (node, prop, error_code);
00345     else
00346     {
00347         i = default_value;

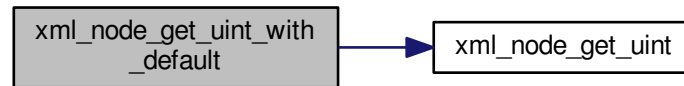
```

```

00348     *error_code = 0;
00349 }
00350 return i;
00351 }

```

Here is the call graph for this function:



#### 5.7.3.19 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 463 of file [mpcotool.c](#).

```

00464 {
00465     xmlChar buffer[64];
00466     snprintf ((char *) buffer, 64, "%.14lg", value);
00467     xmlSetProp (node, prop, buffer);
00468 }

```

#### 5.7.3.20 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 425 of file [mpcotool.c](#).

```

00426 {
00427     xmlChar buffer[64];
00428     snprintf ((char *) buffer, 64, "%d", value);
00429     xmlSetProp (node, prop, buffer);
00430 }

```

#### 5.7.3.21 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 444 of file [mpcotool.c](#).

```

00445 {
00446     xmlChar buffer[64];
00447     snprintf ((char *) buffer, 64, "%u", value);
00448     xmlSetProp (node, prop, buffer);
00449 }
```

## 5.8 mpcotool.h

```

00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burquete 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;
00073     char *directory;
00074     char *name;
00075     double *rangemin;
00076     double *rangemax;
00077     double *rangeminabs;
00078     double *rangemaxabs;
00079     double *weight;
00080     double *step;
00081     unsigned int *precision;
00082     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 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00196                                             const xmlChar * prop,
00197                                             unsigned int default_value,
00198                                             int *error_code);
00199 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00200                           int *error_code);
00201 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
00202                                     , double default_value, int *error_code);
00203 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00204 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00205                        unsigned int value);
00206 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00207 void input_new ();
00208 void input_free ();
00209 int input_open (char *filename);
00210 void calibrate_input (unsigned int simulation, char *input,
00211                     GMappedFile * template);
00212 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00213 void calibrate_print ();
00214 void calibrate_save_variables (unsigned int simulation, double error);
00215 void calibrate_best (unsigned int simulation, double value);
00216 void calibrate_sequential ();
00217 void *calibrate_thread (ParallelData * data);
00218 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00219                     double *error_best);
00220 #if HAVE_MPI
00221 void calibrate_synchronise ();
00222 #endif
00223 void calibrate_sweep ();
00224 void calibrate_MonteCarlo ();
00225 void calibrate_best_gradient (unsigned int simulation, double value);
00226 void calibrate_gradient_sequential ();
00227 void *calibrate_gradient_thread (ParallelData * data);
00228 double calibrate_variable_step_gradient (unsigned int variable);
00229 void calibrate_step_gradient (unsigned int simulation);
00230 void calibrate_gradient ();
00231 double calibrate_genetic_objective (Entity * entity);
00232 void calibrate_genetic ();
00233 void calibrate_save_old ();
00234 void calibrate_merge_old ();
00235 void calibrate_refine ();
00236 void calibrate_step ();
00237 void calibrate_iterate ();
00238 void calibrate_open ();
00239
00240 #endif

```



# Index

ALGORITHM\_GENETIC  
    mpcotoool.h, 139  
ALGORITHM\_MONTE\_CARLO  
    mpcotoool.h, 139  
ALGORITHM\_SWEEP  
    mpcotoool.h, 139  
Algorithm  
    mpcotoool.h, 139  
  
Calibrate, 13  
    thread\_gradient, 15  
calibrate\_best  
    mpcotoool.c, 46  
    mpcotoool.h, 140  
calibrate\_best\_gradient  
    mpcotoool.c, 46  
    mpcotoool.h, 140  
calibrate\_estimate\_gradient\_coordinates  
    mpcotoool.c, 47  
calibrate\_estimate\_gradient\_random  
    mpcotoool.c, 47  
calibrate\_genetic\_objective  
    mpcotoool.c, 48  
    mpcotoool.h, 141  
calibrate\_gradient\_sequential  
    mpcotoool.c, 49  
calibrate\_gradient\_thread  
    mpcotoool.c, 49  
    mpcotoool.h, 141  
calibrate\_input  
    mpcotoool.c, 51  
    mpcotoool.h, 142  
calibrate\_merge  
    mpcotoool.c, 52  
    mpcotoool.h, 143  
calibrate\_parse  
    mpcotoool.c, 53  
    mpcotoool.h, 144  
calibrate\_save\_variables  
    mpcotoool.c, 55  
    mpcotoool.h, 146  
calibrate\_step\_gradient  
    mpcotoool.c, 55  
    mpcotoool.h, 146  
calibrate\_thread  
    mpcotoool.c, 57  
    mpcotoool.h, 148  
config.h, 25  
cores\_number  
    interface.h, 31  
    mpcotoool.c, 58  
Experiment, 15  
  
format  
    mpcotoool.c, 82  
  
GRADIENT\_METHOD\_COORDINATES  
    mpcotoool.h, 139  
GRADIENT\_METHOD\_RANDOM  
    mpcotoool.h, 139  
GradientMethod  
    mpcotoool.h, 139  
  
Input, 16  
input\_open  
    mpcotoool.c, 58  
    mpcotoool.h, 149  
input\_save  
    interface.h, 31  
    mpcotoool.c, 66  
input\_save\_gradient  
    mpcotoool.c, 68  
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, 38  
  
main  
    mpcotoool.c, 70  
mpcotoool.c, 41  
    calibrate\_best, 46  
    calibrate\_best\_gradient, 46  
    calibrate\_estimate\_gradient\_coordinates, 47  
    calibrate\_estimate\_gradient\_random, 47  
    calibrate\_genetic\_objective, 48  
    calibrate\_gradient\_sequential, 49  
    calibrate\_gradient\_thread, 49  
    calibrate\_input, 51  
    calibrate\_merge, 52  
    calibrate\_parse, 53  
    calibrate\_save\_variables, 55  
    calibrate\_step\_gradient, 55  
    calibrate\_thread, 57  
    cores\_number, 58  
    format, 82  
    input\_open, 58

- input\_save, 66
- input\_save\_gradient, 68
- main, 70
- precision, 82
- show\_error, 72
- show\_message, 73
- template, 83
- window\_get\_algorithm, 73
- window\_get\_gradient, 74
- window\_read, 74
- window\_save, 76
- window\_template\_experiment, 78
- xml\_node\_get\_float, 78
- xml\_node\_get\_float\_with\_default, 79
- xml\_node\_get\_int, 80
- xml\_node\_get\_uint, 80
- xml\_node\_get\_uint\_with\_default, 81
- xml\_node\_set\_float, 81
- xml\_node\_set\_int, 82
- xml\_node\_set\_uint, 82
- mpcotool.h, 136
  - ALGORITHM\_GENETIC, 139
  - ALGORITHM\_MONTE\_CARLO, 139
  - ALGORITHM\_SWEEP, 139
  - Algorithm, 139
  - calibrate\_best, 140
  - calibrate\_best\_gradient, 140
  - calibrate\_genetic\_objective, 141
  - calibrate\_gradient\_thread, 141
  - calibrate\_input, 142
  - calibrate\_merge, 143
  - calibrate\_parse, 144
  - calibrate\_save\_variables, 146
  - calibrate\_step\_gradient, 146
  - calibrate\_thread, 148
  - GRADIENT\_METHOD\_COORDINATES, 139
  - GRADIENT\_METHOD\_RANDOM, 139
  - GradientMethod, 139
  - input\_open, 149
  - show\_error, 157
  - show\_message, 157
  - xml\_node\_get\_float, 159
  - xml\_node\_get\_float\_with\_default, 159
  - xml\_node\_get\_int, 160
  - xml\_node\_get\_uint, 161
  - xml\_node\_get\_uint\_with\_default, 161
  - xml\_node\_set\_float, 162
  - xml\_node\_set\_int, 162
  - xml\_node\_set\_uint, 162
- Options, 17
- ParallelData, 18
- precision
  - mpcotool.c, 82
- Running, 19
- show\_error
  - mpcotool.c, 72
  - mpcotool.h, 157
- show\_message
  - mpcotool.c, 73
  - mpcotool.h, 157
- template
  - mpcotool.c, 83
- thread\_gradient
  - Calibrate, 15
- Variable, 19
- Window, 20
- window\_get\_algorithm
  - interface.h, 33
  - mpcotool.c, 73
- window\_get\_gradient
  - interface.h, 34
  - mpcotool.c, 74
- window\_read
  - interface.h, 34
  - mpcotool.c, 74
- window\_save
  - interface.h, 36
  - mpcotool.c, 76
- window\_template\_experiment
  - interface.h, 38
  - mpcotool.c, 78
- xml\_node\_get\_float
  - mpcotool.c, 78
  - mpcotool.h, 159
- xml\_node\_get\_float\_with\_default
  - mpcotool.c, 79
  - mpcotool.h, 159
- xml\_node\_get\_int
  - mpcotool.c, 80
  - mpcotool.h, 160
- xml\_node\_get\_uint
  - mpcotool.c, 80
  - mpcotool.h, 161
- xml\_node\_get\_uint\_with\_default
  - mpcotool.c, 81
  - mpcotool.h, 161
- xml\_node\_set\_float
  - mpcotool.c, 81
  - mpcotool.h, 162
- xml\_node\_set\_int
  - mpcotool.c, 82
  - mpcotool.h, 162
- xml\_node\_set\_uint
  - mpcotool.c, 82
  - mpcotool.h, 162