

Calibrator

1.2.4

Generated by Doxygen 1.8.9.1

Sun Jan 10 2016 07:35:22

Contents

1	MPCOTool	1
2	Data Structure Index	9
2.1	Data Structures	9
3	File Index	11
3.1	File List	11
4	Data Structure Documentation	13
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
5	File Documentation	25
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	xml_node_get_int	80
5.5.2.29	xml_node_get_uint	80
5.5.2.30	xml_node_get_uint_with_default	81
5.5.2.31	xml_node_set_float	81
5.5.2.32	xml_node_set_int	82
5.5.2.33	xml_node_set_uint	82
5.5.3	Variable Documentation	82
5.5.3.1	format	82
5.5.3.2	precision	83
5.5.3.3	template	83
5.6	mpcotool.c	83
5.7	mpcotool.h File Reference	136
5.7.1	Detailed Description	139
5.7.2	Enumeration Type Documentation	139
5.7.2.1	Algorithm	139
5.7.2.2	GradientMethod	139
5.7.3	Function Documentation	140
5.7.3.1	calibrate_best	140
5.7.3.2	calibrate_best_gradient	140
5.7.3.3	calibrate_genetic_objective	141
5.7.3.4	calibrate_gradient_thread	141
5.7.3.5	calibrate_input	142
5.7.3.6	calibrate_merge	144
5.7.3.7	calibrate_parse	144
5.7.3.8	calibrate_save_variables	146
5.7.3.9	calibrate_step_gradient	146
5.7.3.10	calibrate_thread	148
5.7.3.11	input_open	149
5.7.3.12	show_error	157
5.7.3.13	show_message	158
5.7.3.14	xml_node_get_float	159
5.7.3.15	xml_node_get_float_with_default	159
5.7.3.16	xml_node_get_int	160
5.7.3.17	xml_node_get_uint	161
5.7.3.18	xml_node_get_uint_with_default	161
5.7.3.19	xml_node_set_float	162
5.7.3.20	xml_node_set_int	162
5.7.3.21	xml_node_set_uint	162
5.8	mpcotool.h	163

Index	167
-----------------------	-----

Chapter 1

MPCOTool

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

VERSIONS

- 1.2.4: Stable and recommended version.
- 1.3.9: Developing version to do new features.

AUTHORS

- Javier Burguete Tolosa (jburguete@eead.csic.es)
- Borja Latorre Garcés (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:

- 1.2.4/configure.ac: configure generator.
- 1.2.4/Makefile.in: Makefile generator.
- 1.2.4/config.h.in: config header generator.
- 1.2.4/mpcotool.c: main source code.
- 1.2.4/mpcotool.h: main header code.
- 1.2.4/interface.h: interface header code.
- 1.2.4/build: script to build all.
- 1.2.4/logo.png: logo figure.
- 1.2.4/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/1.2.4
```

```
$ 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/1.2.4):

```
$ 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 ../1.2.4
$ 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:

Calibrate	Struct to define the calibration data	13
Experiment	Struct to define experiment data	15
Input	Struct to define the calibration input file	16
Options	Struct to define the options dialog	17
ParallelData	Struct to pass to the GThreads parallelized function	18
Running	Struct to define the running dialog	19
Variable	Struct to define variable data	19
Window	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:

config.h	Configuration header file	25
interface.h	Header file of the interface	29
mpcotool.c	Source file of the mpcotool	41
mpcotool.h	Header file of the mpcotool	136

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

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

Data Fields

- `GMappedFile ** file [MAX_NINPUTS]`
Matrix of input template files.
- `char ** template [MAX_NINPUTS]`
Matrix of template names of input files.
- `char ** experiment`
Array of experimental data file names.
- `char ** label`
Array of variable names.
- `gsl_rng * rng`
GSL random number generator.
- `GeneticVariable * genetic_variable`
Array of variables for the genetic algorithm.
- `FILE * file_result`
Result file.
- `FILE * file_variables`
Variables file.
- `char * result`
Name of the result file.
- `char * variables`
Name of the variables file.
- `char * simulator`
Name of the simulator program.
- `char * evaluator`
Name of the program to evaluate the objective function.
- `double * value`
Array of variable values.
- `double * rangemin`
Array of minimum variable values.
- `double * rangemax`

- *Array of maximum variable values.*
double * [rangeminabs](#)
- *Array of absolute minimum variable values.*
double * [rangemaxabs](#)
- *Array of absolute maximum variable values.*
double * [error_best](#)
- *Array of the best minimum errors.*
double * [weight](#)
- *Array of the experiment weights.*
double * [step](#)
- *Array of gradient based method step sizes.*
double * [gradient](#)
- *Vector of gradient estimation.*
double * [value_old](#)
- *Array of the best variable values on the previous step.*
double * [error_old](#)
- *Array of the best minimum errors on the previous step.*
unsigned int * [precision](#)
- *Array of variable precisions.*
unsigned int * [nsweeps](#)
- *Array of sweeps of the sweep algorithm.*
unsigned int * [thread](#)
- *Array of simulation numbers to calculate on the thread.*
unsigned int * [thread_gradient](#)
- *Array of best simulation numbers.*
unsigned int * [simulation_best](#)
- *Algorithm tolerance.*
double [tolerance](#)
- *Mutation probability.*
double [mutation_ratio](#)
- *Reproduction probability.*
double [reproduction_ratio](#)
- *Adaptation probability.*
double [adaptation_ratio](#)
- *Relaxation parameter.*
double [relaxation](#)
- *Calculation time.*
double [calculation_time](#)
- *Seed of the pseudo-random numbers generator.*
unsigned long int [seed](#)
- *Variables number.*
unsigned int [nvariables](#)
- *Experiments number.*
unsigned int [nexperiments](#)
- *Number of input files to the simulator.*
unsigned int [ninputs](#)
- *Simulations number per experiment.*
unsigned int [nsimulations](#)
- *Method to estimate the gradient.*
unsigned int [gradient_method](#)
- *Method to estimate the gradient.*
unsigned int [nsteps](#)

- unsigned int [nesteemates](#)
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:

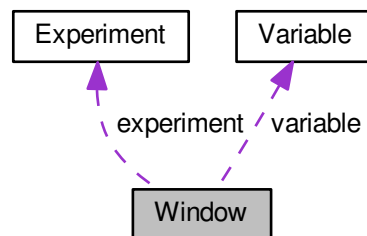
- [interface.h](#)

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.*
- GtkWidget * [grid_files](#)
Files GtkWidget.
- GtkWidget * [label_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [button_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [check_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [button_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [label_result](#)
Result file GtkWidget.
- GtkWidget * [entry_result](#)
Result file GtkWidget.
- GtkWidget * [label_variables](#)
Variables file GtkWidget.
- GtkWidget * [entry_variables](#)
Variables file GtkWidget.
- GtkWidget * [frame_algorithm](#)
GtkFrame to set the algorithm.
- GtkWidget * [grid_algorithm](#)
GtkWidget to set the algorithm.
- GtkWidget * [button_algorithm](#) [NALGORITHMS]
Array of GtkWidget to set the algorithm.
- GtkWidget * [label_simulations](#)
GtkWidget to set the simulations number.
- GtkWidget * [spin_simulations](#)
GtkSpinButton to set the simulations number.
- GtkWidget * [label_iterations](#)
GtkWidget to set the iterations number.
- GtkWidget * [spin_iterations](#)
GtkSpinButton to set the iterations number.
- GtkWidget * [label_tolerance](#)
GtkWidget to set the tolerance.
- GtkWidget * [spin_tolerance](#)
GtkSpinButton to set the tolerance.
- GtkWidget * [label_bests](#)
GtkWidget to set the best number.
- GtkWidget * [spin_bests](#)
GtkSpinButton to set the best number.
- GtkWidget * [label_population](#)
GtkWidget to set the population number.
- GtkWidget * [spin_population](#)
GtkSpinButton to set the population number.
- GtkWidget * [label_generations](#)
GtkWidget to set the generations number.
- GtkWidget * [spin_generations](#)
GtkSpinButton to set the generations number.
- GtkWidget * [label_mutation](#)
GtkWidget 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](#)

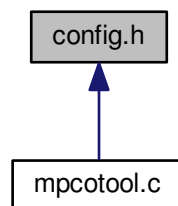
Chapter 5

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"

```

```

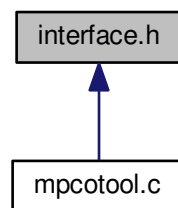
00111 #define XML_STEP (const xmlChar*)"step"
00112 #define XML_SWEEP (const xmlChar*)"sweep"
00113 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00114 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00116 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00118 #define XML_TEMPLATE4 (const xmlChar*)"template4"
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 [4876](#) of file [mpcotoool.c](#).

```

04877 {
04878     #ifdef G_OS_WIN32
04879         SYSTEM_INFO sysinfo;
04880         GetSystemInfo (&sysinfo);
04881         return sysinfo.dwNumberOfProcessors;
04882     #else
04883         return (int) sysconf (_SC_NPROCESSORS_ONLN);
04884     #endif
04885 }
```

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 2700 of file `mpcotool.c`.

```

02701 {
02702     unsigned int i, j;
02703     char *buffer;
02704     xmlDoc *doc;
02705     xmlNode *node, *child;
02706     GFile *file, *file2;
02707
02708     #if DEBUG
02709         fprintf (stderr, "input_save: start\n");
02710     #endif
02711
02712     // Getting the input file directory
02713     input->name = g_path_get_basename (filename);
02714     input->directory = g_path_get_dirname (filename);
02715     file = g_file_new_for_path (input->directory);
02716
02717     // Opening the input file
02718     doc = xmlNewDoc ((const xmlChar *) "1.0");
02719
02720     // Setting root XML node
02721     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02722     xmlDocSetRootElement (doc, node);
02723
02724     // Adding properties to the root XML node
02725     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02726         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02727     if (xmlStrcmp ((const xmlChar *) input->variables,
02728         variables_name))
02729         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02730         variables);
02731     file2 = g_file_new_for_path (input->simulator);
02732     buffer = g_file_get_relative_path (file, file2);
02733     g_object_unref (file2);
02734     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02735     g_free (buffer);
02736     if (input->evaluator)
02737     {
02738         file2 = g_file_new_for_path (input->evaluator);
02739         buffer = g_file_get_relative_path (file, file2);
02740         g_object_unref (file2);
02741         if (xmlStrlen ((xmlChar *) buffer))
02742             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02743         g_free (buffer);
02744     }
02745     if (input->seed != DEFAULT_RANDOM_SEED)
02746         xml_node_set_uint (node, XML_SEED, input->seed);
02747
02748     // Setting the algorithm
02749     buffer = (char *) g_malloc (64);
02750     switch (input->algorithm)
02751     {
02752     case ALGORITHM_MONTE_CARLO:
02753         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02754         snprintf (buffer, 64, "%u", input->nsimulations);
02755         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02756         snprintf (buffer, 64, "%u", input->niterations);
02757         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02758         snprintf (buffer, 64, "%.3lg", input->tolerance);
02759         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02760         snprintf (buffer, 64, "%u", input->nbest);
02761         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02762         input_save_gradient (node);
02763         break;
02764     case ALGORITHM_SWEEP:
02765         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02766         snprintf (buffer, 64, "%u", input->niterations);
02767         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02768         snprintf (buffer, 64, "%.3lg", input->tolerance);
02769         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02770         snprintf (buffer, 64, "%u", input->nbest);
02771         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02772         input_save_gradient (node);
02773         break;
02774     default:
02775         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02776         snprintf (buffer, 64, "%u", input->nsimulations);
02777         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02778         snprintf (buffer, 64, "%u", input->niterations);
02779         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02780         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);

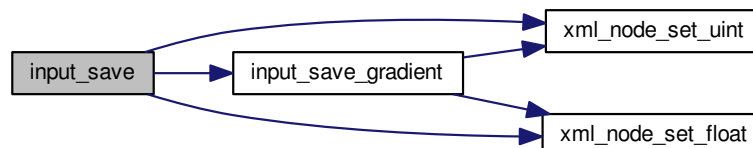
```

```

02779     xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02780     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02781     xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02782     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02783     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02784     break;
02785 }
02786 g_free (buffer);
02787
02788 // Setting the experimental data
02789 for (i = 0; i < input->nexperiments; ++i)
02790 {
02791     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02792     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02793     if (input->weight[i] != 1.)
02794         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02795     for (j = 0; j < input->ninputs; ++j)
02796         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02797 }
02798
02799 // Setting the variables data
02800 for (i = 0; i < input->nvariables; ++i)
02801 {
02802     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02803     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02804     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02805     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02806         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02807     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02808     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02809         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02810     if (input->precision[i] != DEFAULT_PRECISION)
02811         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02812     if (input->algorithm == ALGORITHM_SWEEP)
02813         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02814     else if (input->algorithm == ALGORITHM_GENETIC)
02815         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02816     if (input->nsteps)
02817         xml_node_set_float (child, XML_STEP, input->
step[i]);
02818 }
02819
02820 // Saving the XML file
02821 xmlSaveFormatFile (filename, doc, 1);
02822
02823 // Freeing memory
02824 xmlFreeDoc (doc);
02825
02826 #if DEBUG
02827 fprintf (stderr, "input_save: end\n");
02828 #endif
02829 }

```

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 2933 of file [mpcotool.c](#).

```

02934 {
02935     unsigned int i;
02936     #if DEBUG
02937         fprintf (stderr, "window_get_algorithm: start\n");
02938     #endif
02939     for (i = 0; i < NALGORITHMS; ++i)
02940         if (gtk_toggle_button_get_active
02941             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02942             break;
02943     #if DEBUG
02944         fprintf (stderr, "window_get_algorithm: %u\n", i);
02945         fprintf (stderr, "window_get_algorithm: end\n");
02946     #endif
02947     return i;
02948 }
```

5.3.2.4 int window_get_gradient ()

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2956 of file [mpcotool.c](#).

```

02957 {
02958     unsigned int i;
02959     #if DEBUG
02960         fprintf (stderr, "window_get_gradient: start\n");
02961     #endif
02962     for (i = 0; i < NGRADIENTS; ++i)
02963         if (gtk_toggle_button_get_active
02964             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02965             break;
02966     #if DEBUG
02967         fprintf (stderr, "window_get_gradient: %u\n", i);
02968         fprintf (stderr, "window_get_gradient: end\n");
02969     #endif
02970     return i;
02971 }
```

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 4053 of file [mpcotool.c](#).


```

04054 {
04055     unsigned int i;
04056     char *buffer;
04057     #if DEBUG
04058         fprintf (stderr, "window_read: start\n");
04059     #endif
04060     // Reading new input file
04061     input_free ();
04062     if (!input_open (filename))
04063         return 0;
04064     // Setting GTK+ widgets data
04065     gtk_entry_set_text (window->entry_result, input->result);
04066     gtk_entry_set_text (window->entry_variables, input->
variables);
04067     buffer = g_build_filename (input->directory, input->
simulator, NULL);
04070     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
04071     g_free (buffer);
04072     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
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     gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
04080     switch (input->algorithm)
04081     {
04082     case ALGORITHM_MONTE_CARLO:
04083         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
04084     case ALGORITHM_SWEEP:
04085         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
04086         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
04087         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
04088         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
input->nsteps);
04089         if (input->nsteps)
04090         {
04091             gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
04092             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
04093             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
04094             switch (input->gradient_method)
04095             {
04096             case GRADIENT_METHOD_RANDOM:
04097                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
04098             }
04099         }
04100         break;
04101     default:
04102         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
04103         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
04104         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
04105         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
04106         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
04107     }
04108     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
04109     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
04110     gtk_combo_box_text_remove_all (window->combo_experiment);
04111     for (i = 0; i < input->nexperiments; ++i)
04112         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
04113     g_signal_handler_unblock

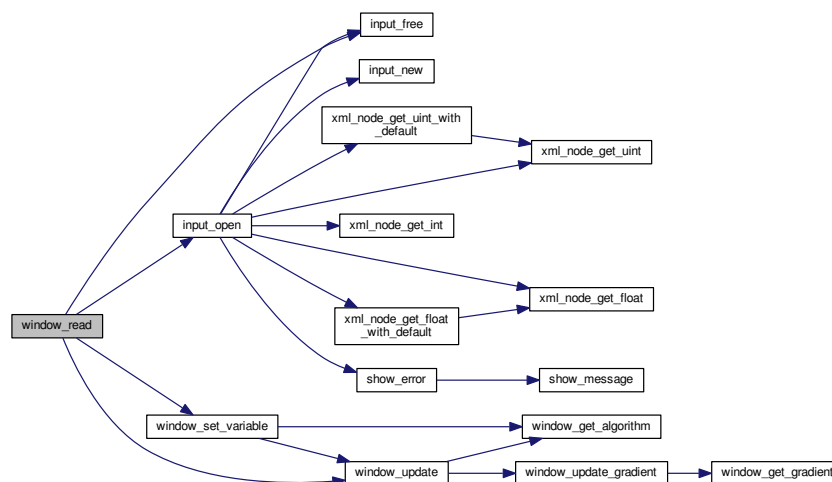
```

```

04132     (window->button_experiment, window->
id_experiment_name);
04133     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
04134     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04135     g_signal_handler_block (window->combo_variable, window->
id_variable);
04136     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
04137     gtk_combo_box_text_remove_all (window->combo_variable);
04138     for (i = 0; i < input->nvariables; ++i)
04139         gtk_combo_box_text_append_text (window->combo_variable,
input->label[i]);
04140     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
04141     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
04142     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04143     window_set_variable ();
04144     window_update ();
04145
04146 #if DEBUG
04147     fprintf (stderr, "window_read: end\n");
04148 #endif
04149     return 1;
04150 }

```

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 3011 of file [mpcotool.c](#).

```

03012 {
03013     GtkFileChooserDialog *dlg;
03014     GtkFileFilter *filter;
03015     char *buffer;
03016
03017 #if DEBUG
03018     fprintf (stderr, "window_save: start\n");
03019 #endif
03020

```

```

03021 // Opening the saving dialog
03022 dlg = (GtkFileChooserDialog *)
03023     gtk_file_chooser_dialog_new (gettext ("Save file"),
03024                                 window->window,
03025                                 GTK_FILE_CHOOSER_ACTION_SAVE,
03026                                 gettext ("_Cancel"),
03027                                 GTK_RESPONSE_CANCEL,
03028                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03029 gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03030 buffer = g_build_filename (input->directory, input->name, NULL);
03031 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03032 g_free (buffer);
03033
03034 // Adding XML filter
03035 filter = (GtkFileFilter *) gtk_file_filter_new ();
03036 gtk_file_filter_set_name (filter, "XML");
03037 gtk_file_filter_add_pattern (filter, "*.xml");
03038 gtk_file_filter_add_pattern (filter, "*.XML");
03039 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03040
03041 // If OK response then saving
03042 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03043 {
03044
03045     // Adding properties to the root XML node
03046     input->simulator = gtk_file_chooser_get_filename
03047         (GTK_FILE_CHOOSER (window->button_simulator));
03048     if (gtk_toggle_button_get_active
03049         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03050         input->evaluator = gtk_file_chooser_get_filename
03051             (GTK_FILE_CHOOSER (window->button_evaluator));
03052     else
03053         input->evaluator = NULL;
03054     input->result
03055         = (char *) xmlStrdup ((const xmlChar *)
03056                               gtk_entry_get_text (window->entry_result));
03057     input->variables
03058         = (char *) xmlStrdup ((const xmlChar *)
03059                               gtk_entry_get_text (window->entry_variables));
03060
03061     // Setting the algorithm
03062     switch (window_get_algorithm ())
03063     {
03064     case ALGORITHM_MONTE_CARLO:
03065         input->algorithm = ALGORITHM_MONTE_CARLO;
03066         input->nsimulations
03067             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03068         input->niterations
03069             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03070         input->tolerance = gtk_spin_button_get_value (window->
03071 spin_tolerance);
03072         input->nbest = gtk_spin_button_get_value_as_int (window->
03073 spin_bests);
03074         window_save_gradient ();
03075         break;
03076     case ALGORITHM_SWEEP:
03077         input->algorithm = ALGORITHM_SWEEP;
03078         input->niterations
03079             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03080         input->tolerance = gtk_spin_button_get_value (window->
03081 spin_tolerance);
03082         input->nbest = gtk_spin_button_get_value_as_int (window->
03083 spin_bests);
03084         window_save_gradient ();
03085         break;
03086     default:
03087         input->algorithm = ALGORITHM_GENETIC;
03088         input->nsimulations
03089             = gtk_spin_button_get_value_as_int (window->spin_population);
03090         input->niterations
03091             = gtk_spin_button_get_value_as_int (window->spin_generations);
03092         input->mutation_ratio
03093             = gtk_spin_button_get_value (window->spin_mutation);
03094         input->reproduction_ratio
03095             = gtk_spin_button_get_value (window->spin_reproduction);
03096         input->adaptation_ratio
03097             = gtk_spin_button_get_value (window->spin_adaptation);
03098         break;
03099     }
03100
03101     // Saving the XML file
03102     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03103     input_save (buffer);
03104
03105     // Closing and freeing memory
03106     g_free (buffer);
03107     gtk_widget_destroy (GTK_WIDGET (dlg));

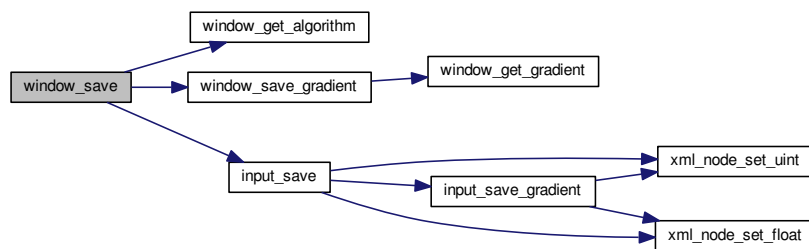
```

```

03104 #if DEBUG
03105     fprintf (stderr, "window_save: end\n");
03106 #endif
03107     return 1;
03108 }
03109
03110 // Closing and freeing memory
03111 gtk_widget_destroy (GTK_WIDGET (dlg));
03112 #if DEBUG
03113     fprintf (stderr, "window_save: end\n");
03114 #endif
03115     return 0;
03116 }

```

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 3657 of file `mpcotool.c`.

```

03658 {
03659     unsigned int i, j;
03660     char *buffer;
03661     GFile *file1, *file2;
03662     #if DEBUG
03663         fprintf (stderr, "window_template_experiment: start\n");
03664     #endif
03665     i = (size_t) data;
03666     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03667     file1
03668         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03669     file2 = g_file_new_for_path (input->directory);
03670     buffer = g_file_get_relative_path (file2, file1);
03671     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03672     g_free (buffer);
03673     g_object_unref (file2);
03674     g_object_unref (file1);
03675     #if DEBUG
03676         fprintf (stderr, "window_template_experiment: end\n");
03677     #endif
03678 }

```

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

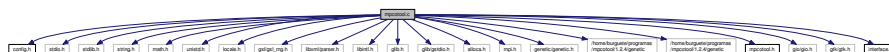
Source file of the mpcotool.

```

#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 1444 of file [mpcotool.c](#).

```

01445 {
01446     unsigned int i, j;
01447     double e;
01448     #if DEBUG
01449         fprintf (stderr, "calibrate_best: start\n");
01450         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01451                 calibrate->nsaveds, calibrate->nbest);
01452     #endif
01453     if (calibrate->nsaveds < calibrate->nbest
01454         || value < calibrate->error_best[calibrate->nsaveds - 1])
01455     {
01456         if (calibrate->nsaveds < calibrate->nbest)
01457             ++calibrate->nsaveds;
01458         calibrate->error_best[calibrate->nsaveds - 1] = value;
01459         calibrate->simulation_best[calibrate->
01460             nsaveds - 1] = simulation;
01461         for (i = calibrate->nsaveds; --i;)
01462         {
01463             if (calibrate->error_best[i] < calibrate->
01464                 error_best[i - 1])
01465             {
01466                 j = calibrate->simulation_best[i];
01467                 e = calibrate->error_best[i];
01468                 calibrate->simulation_best[i] = calibrate->
01469                     simulation_best[i - 1];
01470                 calibrate->error_best[i] = calibrate->
01471                     error_best[i - 1];
01472                 calibrate->simulation_best[i - 1] = j;
01473                 calibrate->error_best[i - 1] = e;
01474             }
01475             else
01476                 break;
01477         }
01478     }
01479     #if DEBUG
01480         fprintf (stderr, "calibrate_best: end\n");
01481     #endif
01482 }
```

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 1757 of file [mpcotool.c](#).

```

01758 {
01759     #if DEBUG
01760         fprintf (stderr, "calibrate_best_gradient: start\n");
01761         fprintf (stderr,
01762             "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01763             simulation, value, calibrate->error\_best[0]);
01764     #endif
01765     if (value < calibrate->error\_best[0])
01766     {
01767         calibrate->error\_best[0] = value;
01768         calibrate->simulation\_best[0] = simulation;
01769     #if DEBUG
01770         fprintf (stderr,
01771             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01772             simulation, value);
01773     #endif
01774     }
01775     #if DEBUG
01776         fprintf (stderr, "calibrate_best_gradient: end\n");
01777     #endif
01778 }
```

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 1894 of file [mpcotool.c](#).

```

01896 {
01897     double x;
01898     #if DEBUG
01899         fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01900     #endif
01901     x = calibrate->gradient[variable];
01902     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01903     {
01904         if (estimate & 1)
01905             x += calibrate->step[variable];
01906         else
01907             x -= calibrate->step[variable];
01908     }
01909     #if DEBUG
01910         fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01911             variable, x);
01912         fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01913     #endif
01914     return x;
01915 }
```

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 1867 of file `mpcotool.c`.

```

01869 {
01870     double x;
01871     #if DEBUG
01872     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01873     #endif
01874     x = calibrate->gradient[variable]
01875         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
01876         step[variable];
01877     #if DEBUG
01878     fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01879             variable, x);
01880     #endif
01881     return x;
01882 }

```

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.

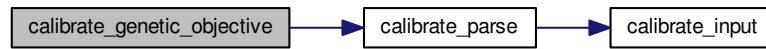
Definition at line 2060 of file `mpcotool.c`.

```

02061 {
02062     unsigned int j;
02063     double objective;
02064     char buffer[64];
02065     #if DEBUG
02066     fprintf (stderr, "calibrate_genetic_objective: start\n");
02067     #endif
02068     for (j = 0; j < calibrate->nvariables; ++j)
02069     {
02070         calibrate->value[entity->id * calibrate->nvariables + j]
02071             = genetic_get_variable (entity, calibrate->genetic_variable + j);
02072     }
02073     for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02074     {
02075         objective += calibrate_parse (entity->id, j);
02076         g_mutex_lock (mutex);
02077         for (j = 0; j < calibrate->nvariables; ++j)
02078         {
02079             snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02080             fprintf (calibrate->file_variables, buffer,
02081                     genetic_get_variable (entity, calibrate->
02082                     genetic_variable + j));
02083         }
02084         fprintf (calibrate->file_variables, "%.14le\n", objective);
02085         g_mutex_unlock (mutex);
02086     }
02087     #if DEBUG
02088     fprintf (stderr, "calibrate_genetic_objective: end\n");
02089     #endif
02090     return objective;
02091 }

```

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

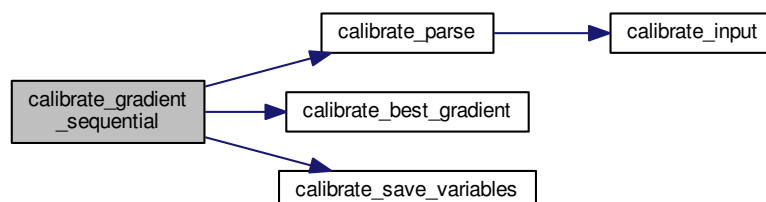
<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 1787 of file [mpcotool.c](#).

```

01788 {
01789     unsigned int i, j, k;
01790     double e;
01791     #if DEBUG
01792         fprintf (stderr, "calibrate_gradient_sequential: start\n");
01793         fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01794                 "nend_gradient=%u\n",
01795                 calibrate->nstart_gradient, calibrate->
01796                 nend_gradient);
01797     #endif
01798     for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01799     {
01800         k = simulation + i;
01801         e = 0.;
01802         for (j = 0; j < calibrate->nexperiments; ++j)
01803             e += calibrate_parse (k, j);
01804         calibrate_best_gradient (k, e);
01805         calibrate_save_variables (k, e);
01806     #if DEBUG
01807         fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01808     #endif
01809     }
01810     #if DEBUG
01811         fprintf (stderr, "calibrate_gradient_sequential: end\n");
01812     #endif
01813 }
  
```

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

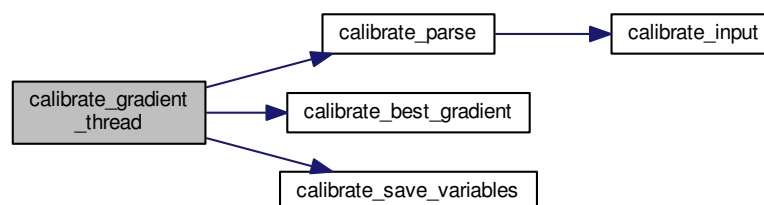
Definition at line 1822 of file [mpcotool.c](#).

```

01823 {
01824     unsigned int i, j, thread;
01825     double e;
01826     #if DEBUG
01827         fprintf (stderr, "calibrate_gradient_thread: start\n");
01828     #endif
01829     thread = data->thread;
01830     #if DEBUG
01831         fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01832                 thread,
01833                 calibrate->thread_gradient[thread],
01834                 calibrate->thread_gradient[thread + 1]);
01835     #endif
01836     for (i = calibrate->thread_gradient[thread];
01837          i < calibrate->thread_gradient[thread + 1]; ++i)
01838     {
01839         e = 0.;
01840         for (j = 0; j < calibrate->nexperiments; ++j)
01841             e += calibrate_parse (i, j);
01842         g_mutex_lock (mutex);
01843         calibrate_best_gradient (i, e);
01844         calibrate_save_variables (i, e);
01845         g_mutex_unlock (mutex);
01846     #if DEBUG
01847         fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01848     #endif
01849     }
01850     #if DEBUG
01851         fprintf (stderr, "calibrate_gradient_thread: end\n");
01852     #endif
01853     g_thread_exit (NULL);
01854     return NULL;
01855 }

```

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 1197 of file [mpcotool.c](#).

```

01198 {
01199     unsigned int i;
01200     char buffer[32], value[32], *buffer2, *buffer3, *content;
01201     FILE *file;
01202     gsize length;
01203     GRegex *regex;
01204
01205     #if DEBUG
01206         fprintf (stderr, "calibrate_input: start\n");
01207     #endif
01208
01209     // Checking the file
01210     if (!template)
01211         goto calibrate_input_end;
01212
01213     // Opening template
01214     content = g_mapped_file_get_contents (template);
01215     length = g_mapped_file_get_length (template);
01216     #if DEBUG
01217         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01218                 content);
01219     #endif
01220     file = g_fopen (input, "w");
01221
01222     // Parsing template
01223     for (i = 0; i < calibrate->nvariables; ++i)
01224     {
01225         #if DEBUG
01226             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01227         #endif
01228         snprintf (buffer, 32, "@variable%u@", i + 1);
01229         regex = g_regex_new (buffer, 0, 0, NULL);
01230         if (i == 0)
01231         {
01232             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01233                                               calibrate->label[i], 0, NULL);
01234         #if DEBUG
01235             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01236         #endif
01237         }
01238         else
01239         {
01240             length = strlen (buffer3);
01241             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01242                                               calibrate->label[i], 0, NULL);
01243             g_free (buffer3);
01244         }
01245         g_regex_unref (regex);
01246         length = strlen (buffer2);
01247         snprintf (buffer, 32, "@value%u@", i + 1);
01248         regex = g_regex_new (buffer, 0, 0, NULL);
01249         snprintf (value, 32, format[calibrate->precision[i]],
01250                 calibrate->value[simulation * calibrate->
01251 nvariables + i]);
01252         #if DEBUG
01253             fprintf (stderr, "calibrate_input: value=%s\n", value);
01254         #endif
01255         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01256                                           0, NULL);
01257         g_free (buffer2);
01258         g_regex_unref (regex);
01259     }
01260
01261     // Saving input file
01262     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01263     g_free (buffer3);
01264     fclose (file);
01265
01266     calibrate_input_end:
01267     #if DEBUG
01268         fprintf (stderr, "calibrate_input: end\n");
01269     #endif
01270     return;
01271 }

```

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 1562 of file `mpcotool.c`.

```

01564 {
01565     unsigned int i, j, k, s[calibrate->nbest];
01566     double e[calibrate->nbest];
01567     #if DEBUG
01568     fprintf (stderr, "calibrate_merge: start\n");
01569     #endif
01570     i = j = k = 0;
01571     do
01572     {
01573         if (i == calibrate->nsaveds)
01574         {
01575             s[k] = simulation_best[j];
01576             e[k] = error_best[j];
01577             ++j;
01578             ++k;
01579             if (j == nsaveds)
01580                 break;
01581         }
01582         else if (j == nsaveds)
01583         {
01584             s[k] = calibrate->simulation_best[i];
01585             e[k] = calibrate->error_best[i];
01586             ++i;
01587             ++k;
01588             if (i == calibrate->nsaveds)
01589                 break;
01590         }
01591         else if (calibrate->error_best[i] > error_best[j])
01592         {
01593             s[k] = simulation_best[j];
01594             e[k] = error_best[j];
01595             ++j;
01596             ++k;
01597         }
01598         else
01599         {
01600             s[k] = calibrate->simulation_best[i];
01601             e[k] = calibrate->error_best[i];
01602             ++i;
01603             ++k;
01604         }
01605     }
01606     while (k < calibrate->nbest);
01607     calibrate->nsaveds = k;
01608     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01609     memcpy (calibrate->error_best, e, k * sizeof (double));
01610     #if DEBUG
01611     fprintf (stderr, "calibrate_merge: end\n");
01612     #endif
01613 }
```

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 1284 of file `mpcotool.c`.

```

01285 {
01286     unsigned int i;
01287     double e;
01288     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01289         *buffer3, *buffer4;
01290     FILE *file_result;
01291
01292     #if DEBUG
01293         fprintf (stderr, "calibrate_parse: start\n");
01294         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01295             experiment);
01296     #endif
01297
01298     // Opening input files
01299     for (i = 0; i < calibrate->ninputs; ++i)
01300     {
01301         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01302     #if DEBUG
01303         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01304     #endif
01305         calibrate_input (simulation, &input[i][0],
01306             calibrate->file[i][experiment]);
01307     }
01308     for (; i < MAX_NINPUTS; ++i)
01309         strcpy (&input[i][0], "");
01310     #if DEBUG
01311         fprintf (stderr, "calibrate_parse: parsing end\n");
01312     #endif
01313
01314     // Performing the simulation
01315     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01316     buffer2 = g_path_get_dirname (calibrate->simulator);
01317     buffer3 = g_path_get_basename (calibrate->simulator);
01318     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01319     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
01320         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01321         input[6], input[7], output);
01322     g_free (buffer4);
01323     g_free (buffer3);
01324     g_free (buffer2);
01325     #if DEBUG
01326         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01327     #endif
01328     system (buffer);
01329
01330     // Checking the objective value function
01331     if (calibrate->evaluator)
01332     {
01333         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01334         buffer2 = g_path_get_dirname (calibrate->evaluator);
01335         buffer3 = g_path_get_basename (calibrate->evaluator);
01336         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01337         snprintf (buffer, 512, "%s\ " %s %s %s",
01338             buffer4, output, calibrate->experiment[experiment], result);
01339         g_free (buffer4);
01340         g_free (buffer3);
01341         g_free (buffer2);
01342     #if DEBUG
01343         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01344     #endif
01345         system (buffer);
01346         file_result = g_fopen (result, "r");
01347         e = atof (fgets (buffer, 512, file_result));
01348         fclose (file_result);
01349     }
01350     else
01351     {
01352         strcpy (result, "");
01353         file_result = g_fopen (output, "r");
01354         e = atof (fgets (buffer, 512, file_result));
01355         fclose (file_result);
01356     }
01357
01358     // Removing files
01359     #if !DEBUG
01360     for (i = 0; i < calibrate->ninputs; ++i)
01361     {
01362         if (calibrate->file[i][0])
01363         {

```

```

01364         snprintf (buffer, 512, RM " %s", &input[i][0]);
01365         system (buffer);
01366     }
01367 }
01368     snprintf (buffer, 512, RM " %s %s", output, result);
01369     system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373     fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376 // Returning the objective function
01377 return e * calibrate->weight[experiment];
01378 }

```

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 1416 of file [mpcotool.c](#).

```

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

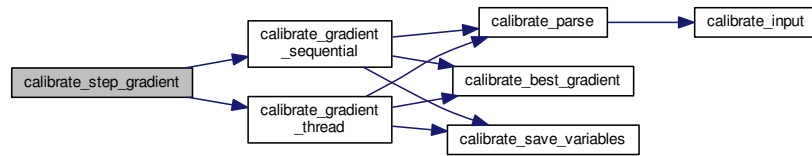
Definition at line 1924 of file `mpcotool.c`.

```

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

```

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

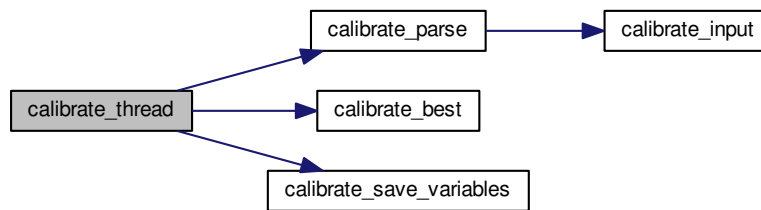
Definition at line 1518 of file [mpcotool.c](#).

```

01519 {
01520     unsigned int i, j, thread;
01521     double e;
01522     #if DEBUG
01523         fprintf (stderr, "calibrate_thread: start\n");
01524     #endif
01525     thread = data->thread;
01526     #if DEBUG
01527         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01528                 calibrate->thread[thread], calibrate->thread[thread + 1]);
01529     #endif
01530     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01531     {
01532         e = 0.;
01533         for (j = 0; j < calibrate->nexperiments; ++j)
01534             e += calibrate_parse (i, j);
01535         g_mutex_lock (mutex);
01536         calibrate_best (i, e);
01537         calibrate_save_variables (i, e);
01538         g_mutex_unlock (mutex);
01539     #if DEBUG
01540         fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01541     #endif
01542     }
01543     #if DEBUG
01544         fprintf (stderr, "calibrate_thread: end\n");
01545     #endif
01546     g_thread_exit (NULL);
01547     return NULL;
01548 }

```

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 [4876](#) of file [mpcotool.c](#).

```

04877 {
04878     #ifdef G_OS_WIN32
04879         SYSTEM_INFO sysinfo;
04880         GetSystemInfo (&sysinfo);
04881         return sysinfo.dwNumberOfProcessors;
04882     #else
04883         return (int) sysconf ( _SC_NPROCESSORS_ONLN );
04884     #endif
04885 }
  
```

5.5.2.15 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 [549](#) of file [mpcotool.c](#).

```

00550 {
00551     char buffer2[64];
00552     char *buffert[MAX_NINPUTS] =
00553         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00554     xmlDoc *doc;
00555     xmlNode *node, *child;
00556     xmlChar *buffer;
00557     char *msg;
00558     int error_code;
00559     unsigned int i;
00560
00561     #if DEBUG
00562         fprintf (stderr, "input_open: start\n");
00563     #endif
  
```



```

00564
00565 // Resetting input data
00566 buffer = NULL;
00567 input_new ();
00568
00569 // Parsing the input file
00570 #if DEBUG
00571 fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00572 #endif
00573 doc = xmlParseFile (filename);
00574 if (!doc)
00575 {
00576     msg = gettext ("Unable to parse the input file");
00577     goto exit_on_error;
00578 }
00579
00580 // Getting the root node
00581 #if DEBUG
00582 fprintf (stderr, "input_open: getting the root node\n");
00583 #endif
00584 node = xmlDocGetRootElement (doc);
00585 if (xmlStrcmp (node->name, XML_CALIBRATE))
00586 {
00587     msg = gettext ("Bad root XML node");
00588     goto exit_on_error;
00589 }
00590
00591 // Getting results file names
00592 input->result = (char *) xmlGetProp (node, XML_RESULT);
00593 if (!input->result)
00594     input->result = (char *) xmlStrdup (result_name);
00595 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00596 if (!input->variables)
00597     input->variables = (char *) xmlStrdup (variables_name);
00598
00599 // Opening simulator program name
00600 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00601 if (!input->simulator)
00602 {
00603     msg = gettext ("Bad simulator program");
00604     goto exit_on_error;
00605 }
00606
00607 // Opening evaluator program name
00608 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00609
00610 // Obtaining pseudo-random numbers generator seed
00611 input->seed
00612     = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00613                                     &error_code);
00614 if (error_code)
00615 {
00616     msg = gettext ("Bad pseudo-random numbers generator seed");
00617     goto exit_on_error;
00618 }
00619
00620 // Opening algorithm
00621 buffer = xmlGetProp (node, XML_ALGORITHM);
00622 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00623 {
00624     input->algorithm = ALGORITHM_MONTE_CARLO;
00625 }
00626 // Obtaining simulations number
00627 input->nsimulations
00628     = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00629 if (error_code)
00630 {
00631     msg = gettext ("Bad simulations number");
00632     goto exit_on_error;
00633 }
00634 }
00635 else if (!xmlStrcmp (buffer, XML_SWEEP))
00636     input->algorithm = ALGORITHM_SWEEP;
00637 else if (!xmlStrcmp (buffer, XML_GENETIC))
00638 {
00639     input->algorithm = ALGORITHM_GENETIC;
00640 }
00641 // Obtaining population
00642 if (xmlHasProp (node, XML_NPOPULATION))
00643 {
00644     input->nsimulations
00645         = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00646     if (error_code || input->nsimulations < 3)
00647     {
00648         msg = gettext ("Invalid population number");
00649         goto exit_on_error;

```

```

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

```

```

00735         msg = gettext
00736             ("No enough survival entities to reproduce the population");
00737         goto exit_on_error;
00738     }
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
00752     // Obtaining iterations number
00753     input->niterations
00754         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00755     if (error_code == 1)
00756         input->niterations = 1;
00757     else if (error_code)
00758     {
00759         msg = gettext ("Bad iterations number");
00760         goto exit_on_error;
00761     }
00762
00763     // Obtaining best number
00764     input->nbest
00765         = xml_node_get_uint_with_default (node,
00766     XML_NBEST, 1, &error_code);
00767     if (error_code || !input->nbest)
00768     {
00769         msg = gettext ("Invalid best number");
00770         goto exit_on_error;
00771     }
00772
00773     // Obtaining tolerance
00774     input->tolerance
00775         = xml_node_get_float_with_default (node,
00776     XML_TOLERANCE, 0.,
00777                                         &error_code);
00778     if (error_code || input->tolerance < 0.)
00779     {
00780         msg = gettext ("Invalid tolerance");
00781         goto exit_on_error;
00782     }
00783
00784     // Getting gradient method parameters
00785     if (xmlHasProp (node, XML_NSTEPS))
00786     {
00787         input->nsteps = xml_node_get_uint (node,
00788     XML_NSTEPS, &error_code);
00789         if (error_code || !input->nsteps)
00790         {
00791             msg = gettext ("Invalid steps number");
00792             goto exit_on_error;
00793         }
00794         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00795         if (!xmlStrcmp (buffer, XML_COORDINATES))
00796             input->gradient_method =
00797     GRADIENT_METHOD_COORDINATES;
00798         else if (!xmlStrcmp (buffer, XML_RANDOM))
00799         {
00800             input->gradient_method =
00801     GRADIENT_METHOD_RANDOM;
00802             input->nestimates
00803                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00804             if (error_code || !input->nestimates)
00805             {
00806                 msg = gettext ("Invalid estimates number");
00807                 goto exit_on_error;
00808             }
00809         }
00810         else
00811         {
00812             msg = gettext ("Unknown method to estimate the gradient");
00813             goto exit_on_error;
00814         }
00815         xmlFree (buffer);
00816         buffer = NULL;
00817         input->relaxation
00818             = xml_node_get_float_with_default (node,
00819     XML_RELAXATION,
00820     DEFAULT_RELAXATION, &error_code);
00821         if (error_code || input->relaxation < 0. || input->

```

```

        relaxation > 2.)
00816     {
00817         msg = gettext ("Invalid relaxation parameter");
00818         goto exit_on_error;
00819     }
00820 }
00821 else
00822     input->nsteps = 0;
00823 }
00824
00825 // Reading the experimental data
00826 for (child = node->children; child; child = child->next)
00827 {
00828     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833     if (xmlHasProp (child, XML_NAME))
00834         buffer = xmlGetProp (child, XML_NAME);
00835     else
00836     {
00837         snprintf (buffer2, 64, "%s %u: %s",
00838             gettext ("Experiment"),
00839             input->nexperiments + 1, gettext ("no data file name"));
00840         msg = buffer2;
00841         goto exit_on_error;
00842     }
00843 #if DEBUG
00844     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845 #endif
00846     input->weight = g_realloc (input->weight,
00847         (1 + input->nexperiments) * sizeof (double));
00848     input->weight[input->nexperiments]
00849         = xml_node_get_float_with_default (child,
00850             XML_WEIGHT, 1., &error_code);
00851     if (error_code)
00852     {
00853         snprintf (buffer2, 64, "%s %s: %s",
00854             gettext ("Experiment"), buffer, gettext ("bad weight"));
00855         msg = buffer2;
00856         goto exit_on_error;
00857     }
00858 #if DEBUG
00859     fprintf (stderr, "input_open: weight=%lg\n",
00860         input->weight[input->nexperiments]);
00861 #endif
00862     if (!input->nexperiments)
00863         input->ninputs = 0;
00864 #if DEBUG
00865     fprintf (stderr, "input_open: template[0]\n");
00866 #endif
00867     if (xmlHasProp (child, XML_TEMPLATE1))
00868     {
00869         input->template[0]
00870             = (char **) g_realloc (input->template[0],
00871                 (1 + input->nexperiments) * sizeof (char *));
00872         buffert[0] = (char *) xmlGetProp (child, template[0]);
00873 #if DEBUG
00874         fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00875             input->nexperiments, buffert[0]);
00876 #endif
00877         if (!input->nexperiments)
00878             ++input->ninputs;
00879 #if DEBUG
00880         fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00881 #endif
00882     }
00883     else
00884     {
00885         snprintf (buffer2, 64, "%s %s: %s",
00886             gettext ("Experiment"), buffer, gettext ("no template"));
00887         msg = buffer2;
00888         goto exit_on_error;
00889     }
00890     for (i = 1; i < MAX_NINPUTS; ++i)
00891     {
00892         #if DEBUG
00893             fprintf (stderr, "input_open: template%u\n", i + 1);
00894         #endif
00895         if (xmlHasProp (child, template[i]))
00896         {
00897             if (input->nexperiments && input->ninputs <= i)
00898             {
00899                 snprintf (buffer2, 64, "%s %s: %s",
00900                     gettext ("Experiment"),
00901                     buffer, gettext ("bad templates number"));

```

```

00901         msg = buffer2;
00902         while (i-- > 0)
00903             xmlFree (buffert[i]);
00904         goto exit_on_error;
00905     }
00906     input->template[i] = (char **)
00907         g_realloc (input->template[i],
00908             (1 + input->nexperiments) * sizeof (char *));
00909     buffert[i] = (char *) xmlGetProp (child, template[i]);
00910 #if DEBUG
00911     fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00912         input->nexperiments, i + 1,
00913         input->template[i][input->nexperiments]);
00914 #endif
00915     if (!input->nexperiments)
00916         ++input->ninputs;
00917 #if DEBUG
00918     fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00919 #endif
00920 }
00921 else if (input->nexperiments && input->ninputs >= i)
00922 {
00923     snprintf (buffer2, 64, "%s %s: %s",
00924         gettext ("Experiment"),
00925         buffer, gettext ("no template"), i + 1);
00926     msg = buffer2;
00927     while (i-- > 0)
00928         xmlFree (buffert[i]);
00929     goto exit_on_error;
00930 }
00931 else
00932     break;
00933 }
00934 input->experiment
00935     = g_realloc (input->experiment,
00936         (1 + input->nexperiments) * sizeof (char *));
00937 input->experiment[input->nexperiments] = (char *) buffer;
00938 for (i = 0; i < input->ninputs; ++i)
00939     input->template[i][input->nexperiments] = buffert[i];
00940 ++input->nexperiments;
00941 #if DEBUG
00942     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00943 #endif
00944 }
00945 if (!input->nexperiments)
00946 {
00947     msg = gettext ("No calibration experiments");
00948     goto exit_on_error;
00949 }
00950 buffer = NULL;
00951
00952 // Reading the variables data
00953 for (; child; child = child->next)
00954 {
00955     if (xmlStrcmp (child->name, XML_VARIABLE))
00956     {
00957         snprintf (buffer2, 64, "%s %u: %s",
00958             gettext ("Variable"),
00959             input->nvariables + 1, gettext ("bad XML node"));
00960         msg = buffer2;
00961         goto exit_on_error;
00962     }
00963     if (xmlHasProp (child, XML_NAME))
00964         buffer = xmlGetProp (child, XML_NAME);
00965     else
00966     {
00967         snprintf (buffer2, 64, "%s %u: %s",
00968             gettext ("Variable"),
00969             input->nvariables + 1, gettext ("no name"));
00970         msg = buffer2;
00971         goto exit_on_error;
00972     }
00973     if (xmlHasProp (child, XML_MINIMUM))
00974     {
00975         input->rangemin = g_realloc
00976             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00977         input->rangeminabs = g_realloc
00978             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00979         input->rangemin[input->nvariables]
00980             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00981         if (error_code)
00982         {
00983             snprintf (buffer2, 64, "%s %s: %s",
00984                 gettext ("Variable"), buffer, gettext ("bad minimum"));
00985             msg = buffer2;
00986             goto exit_on_error;
00987         }
00988     }

```

```

00988         input->rangeminabs[input->nvariables]
00989         = xml_node_get_float_with_default (child,
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;
01014     }
01015     if (xmlHasProp (child, XML_MAXIMUM))
01016     {
01017         input->rangemax = g_realloc
01018             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01019         input->rangemaxabs = g_realloc
01020             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01021         input->rangemax[input->nvariables]
01022         = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01023         if (error_code)
01024         {
01025             snprintf (buffer2, 64, "%s %s: %s",
01026                     gettext ("Variable"), buffer, gettext ("bad maximum"));
01027             msg = buffer2;
01028             goto exit_on_error;
01029         }
01030         input->rangemaxabs[input->nvariables]
01031         = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
01032                                         G_MAXDOUBLE, &error_code);
01033         if (error_code)
01034         {
01035             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01036                     gettext ("bad absolute maximum"));
01037             msg = buffer2;
01038             goto exit_on_error;
01039         }
01040         if (input->rangemax[input->nvariables]
01041             > input->rangemaxabs[input->nvariables])
01042         {
01043             snprintf (buffer2, 64, "%s %s: %s",
01044                     gettext ("Variable"),
01045                     buffer, gettext ("maximum range not allowed"));
01046             msg = buffer2;
01047             goto exit_on_error;
01048         }
01049     }
01050     else
01051     {
01052         snprintf (buffer2, 64, "%s %s: %s",
01053                 gettext ("Variable"), buffer, gettext ("no maximum range"));
01054         msg = buffer2;
01055         goto exit_on_error;
01056     }
01057     if (input->rangemax[input->nvariables]
01058         < input->rangemin[input->nvariables])
01059     {
01060         snprintf (buffer2, 64, "%s %s: %s",
01061                 gettext ("Variable"), buffer, gettext ("bad range"));
01062         msg = buffer2;
01063         goto exit_on_error;
01064     }
01065     input->precision = g_realloc
01066         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01067     input->precision[input->nvariables]
01068     = xml_node_get_uint_with_default (child,
XML_PRECISION,
01069                                     DEFAULT_PRECISION, &error_code);
01070     if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)

```

```

01071     {
01072         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01073             gettext ("bad precision"));
01074         msg = buffer2;
01075         goto exit_on_error;
01076     }
01077     if (input->algorithm == ALGORITHM_SWEEP)
01078     {
01079         if (xmlHasProp (child, XML_NSWEEPS))
01080         {
01081             input->nsweeps = (unsigned int *)
01082                 g_realloc (input->nsweeps,
01083                     (1 + input->nvariables) * sizeof (unsigned int));
01084             input->nsweeps[input->nvariables]
01085                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01086             if (error_code || !input->nsweeps[input->
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         #if DEBUG
01103         fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01104             input->nsweeps[input->nvariables],
01105             input->nsimulations);
01106         #endif
01107     }
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     if (!input->nvariables)

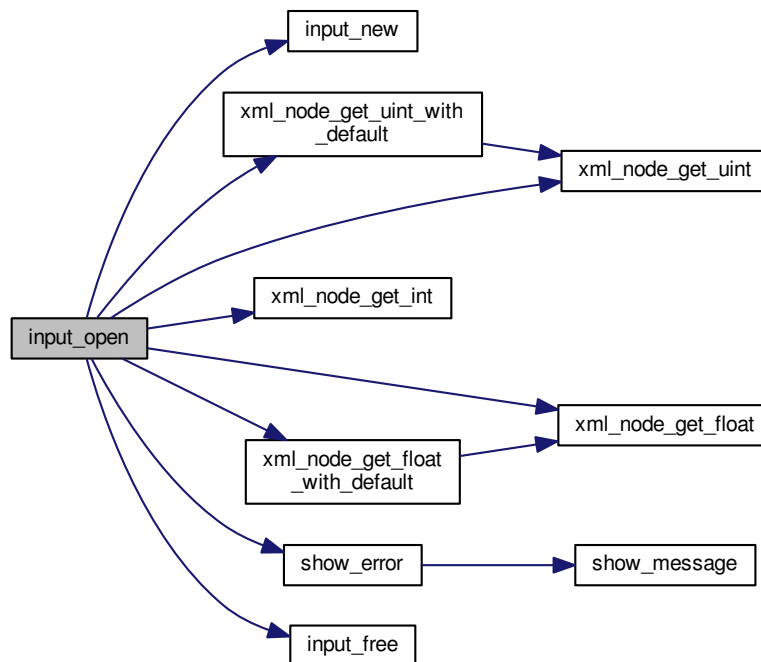
```

```

01156     {
01157         msg = gettext ("No calibration variables");
01158         goto exit_on_error;
01159     }
01160     buffer = NULL;
01161
01162     // Getting the working directory
01163     input->directory = g_path_get_dirname (filename);
01164     input->name = g_path_get_basename (filename);
01165
01166     // Closing the XML document
01167     xmlFreeDoc (doc);
01168
01169     #if DEBUG
01170     fprintf (stderr, "input_open: end\n");
01171     #endif
01172     return 1;
01173
01174 exit_on_error:
01175     xmlFree (buffer);
01176     xmlFreeDoc (doc);
01177     show_error (msg);
01178     input_free ();
01179     #if DEBUG
01180     fprintf (stderr, "input_open: end\n");
01181     #endif
01182     return 0;
01183 }

```

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 2700 of file `mpcotool.c`.

```

02701 {
02702     unsigned int i, j;
02703     char *buffer;
02704     xmlDoc *doc;
02705     xmlNode *node, *child;
02706     GFile *file, *file2;
02707
02708     #if DEBUG
02709         fprintf (stderr, "input_save: start\n");
02710     #endif
02711
02712     // Getting the input file directory
02713     input->name = g_path_get_basename (filename);
02714     input->directory = g_path_get_dirname (filename);
02715     file = g_file_new_for_path (input->directory);
02716
02717     // Opening the input file
02718     doc = xmlNewDoc ((const xmlChar *) "1.0");
02719
02720     // Setting root XML node
02721     node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02722     xmlDocSetRootElement (doc, node);
02723
02724     // Adding properties to the root XML node
02725     if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02726         xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02727     if (xmlStrcmp ((const xmlChar *) input->variables,
02728         variables_name))
02729         xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
02730         variables);
02731     file2 = g_file_new_for_path (input->simulator);
02732     buffer = g_file_get_relative_path (file, file2);
02733     g_object_unref (file2);
02734     xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02735     g_free (buffer);
02736     if (input->evaluator)
02737     {
02738         file2 = g_file_new_for_path (input->evaluator);
02739         buffer = g_file_get_relative_path (file, file2);
02740         g_object_unref (file2);
02741         if (xmlStrlen ((xmlChar *) buffer))
02742             xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02743         g_free (buffer);
02744     }
02745     if (input->seed != DEFAULT_RANDOM_SEED)
02746         xml_node_set_uint (node, XML_SEED, input->seed);
02747
02748     // Setting the algorithm
02749     buffer = (char *) g_malloc (64);
02750     switch (input->algorithm)
02751     {
02752     case ALGORITHM_MONTE_CARLO:
02753         xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02754         snprintf (buffer, 64, "%u", input->nsimulations);
02755         xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02756         snprintf (buffer, 64, "%u", input->niterations);
02757         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02758         snprintf (buffer, 64, "%.3lg", input->tolerance);
02759         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02760         snprintf (buffer, 64, "%u", input->nbest);
02761         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02762         input_save_gradient (node);
02763         break;
02764     case ALGORITHM_SWEEP:
02765         xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02766         snprintf (buffer, 64, "%u", input->niterations);
02767         xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02768         snprintf (buffer, 64, "%.3lg", input->tolerance);
02769         xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02770         snprintf (buffer, 64, "%u", input->nbest);
02771         xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02772         input_save_gradient (node);
02773         break;
02774     default:
02775         xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02776         snprintf (buffer, 64, "%u", input->nsimulations);
02777         xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02778         snprintf (buffer, 64, "%u", input->niterations);
02779         xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02780         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);

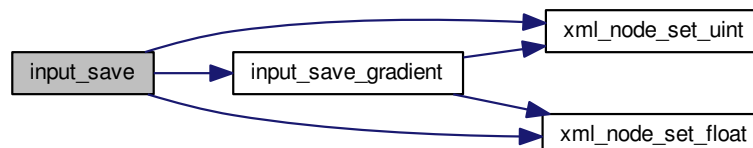
```

```

02779     xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02780     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02781     xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02782     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02783     xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02784     break;
02785 }
02786 g_free (buffer);
02787
02788 // Setting the experimental data
02789 for (i = 0; i < input->nexperiments; ++i)
02790 {
02791     child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02792     xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02793     if (input->weight[i] != 1.)
02794         xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
02795     for (j = 0; j < input->ninputs; ++j)
02796         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02797 }
02798
02799 // Setting the variables data
02800 for (i = 0; i < input->nvariables; ++i)
02801 {
02802     child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02803     xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02804     xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
02805     if (input->rangeminabs[i] != -G_MAXDOUBLE)
02806         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
input->rangeminabs[i]);
02807     xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
02808     if (input->rangemaxabs[i] != G_MAXDOUBLE)
02809         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
input->rangemaxabs[i]);
02810     if (input->precision[i] != DEFAULT_PRECISION)
02811         xml_node_set_uint (child, XML_PRECISION,
input->precision[i]);
02812     if (input->algorithm == ALGORITHM_SWEEP)
02813         xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
02814     else if (input->algorithm == ALGORITHM_GENETIC)
02815         xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
02816     if (input->nsteps)
02817         xml_node_set_float (child, XML_STEP, input->
step[i]);
02818 }
02819
02820 // Saving the XML file
02821 xmlSaveFormatFile (filename, doc, 1);
02822
02823 // Freeing memory
02824 xmlFreeDoc (doc);
02825
02826 #if DEBUG
02827 fprintf (stderr, "input_save: end\n");
02828 #endif
02829 }

```

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

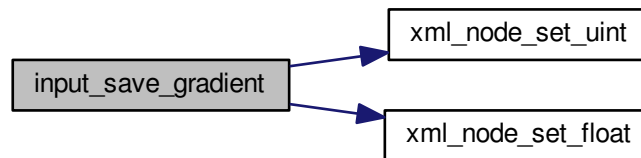
Definition at line 2668 of file [mpcotool.c](#).

```

02669 {
02670 #if DEBUG
02671     fprintf (stderr, "input_save_gradient: start\n");
02672 #endif
02673     if (input->nsteps)
02674     {
02675         xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
02676         if (input->relaxation != DEFAULT_RELAXATION)
02677             xml_node_set_float (node, XML_RELAXATION,
input->relaxation);
02678         switch (input->gradient_method)
02679         {
02680             case GRADIENT_METHOD_COORDINATES:
02681                 xmlSetProp (node, XML_GRADIENT_METHOD,
XML_COORDINATES);
02682                 break;
02683             default:
02684                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02685                 xml_node_set_uint (node, XML_NESTIMATES,
input->nestimates);
02686         }
02687     }
02688 #if DEBUG
02689     fprintf (stderr, "input_save_gradient: end\n");
02690 #endif
02691 }

```

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 4897 of file [mpcotool.c](#).

```

04898 {
04899 #if HAVE_GTK
04900     char *buffer;

```

```

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

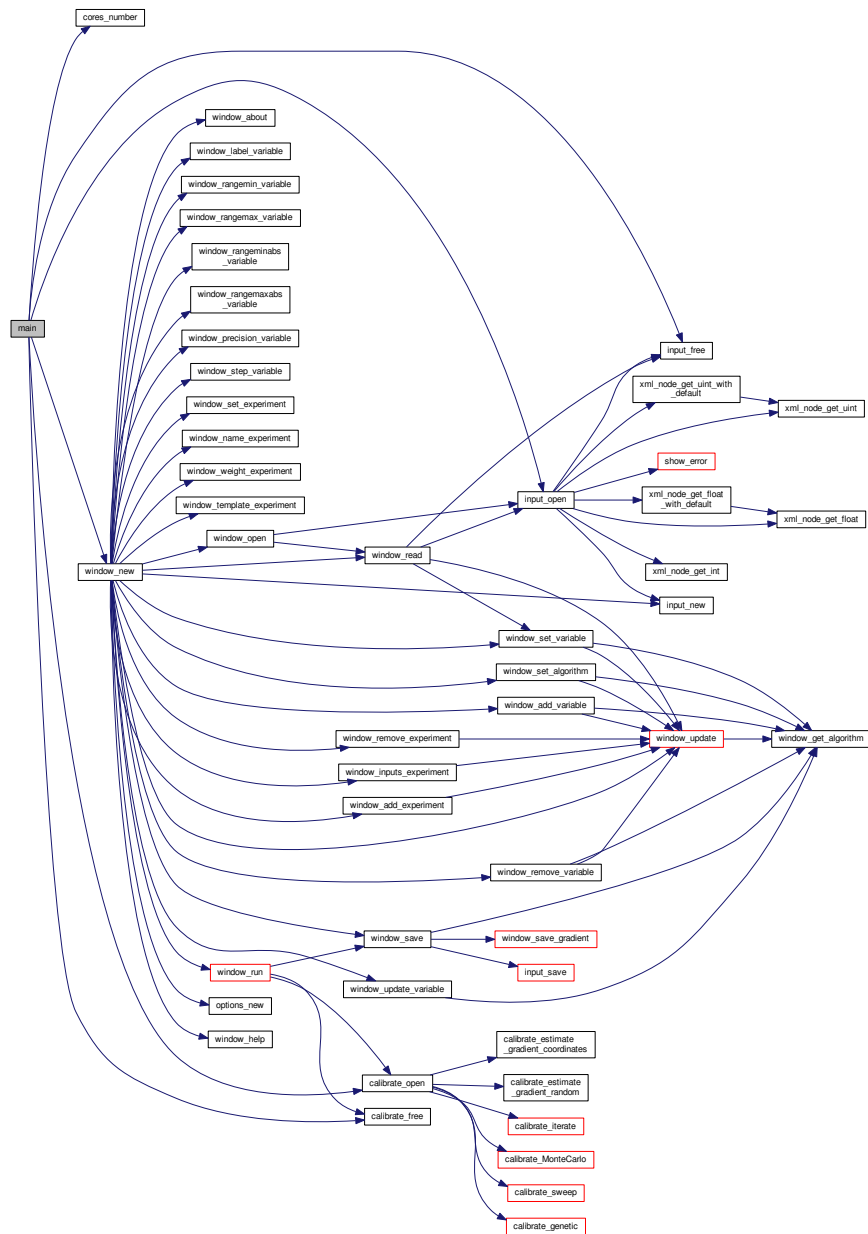
```

```

04987
04988 // Closing
04989 return 0;
04990 }

```

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

```
00258 {
00259     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00260 }
```

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

```
00228 {
00229     #if HAVE_GTK
00230     GtkMessageDialog *dlg;
00231
00232     // Creating the dialog
00233     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00234         (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00235
00236     // Setting the dialog title
00237     gtk_window_set_title (GTK_WINDOW (dlg), title);
00238
00239     // Showing the dialog and waiting response
00240     gtk_dialog_run (GTK_DIALOG (dlg));
00241
00242     // Closing and freeing memory
00243     gtk_widget_destroy (GTK_WIDGET (dlg));
00244
00245     #else
00246     printf ("%s: %s\n", title, msg);
00247     #endif
00248 }
```

5.5.2.21 int window_get_algorithm ()

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line [2933](#) of file [mpcotool.c](#).

```

02934 {
02935     unsigned int i;
02936     #if DEBUG
02937     fprintf (stderr, "window_get_algorithm: start\n");
02938     #endif
02939     for (i = 0; i < NALGORITHMS; ++i)
02940         if (gtk_toggle_button_get_active
02941             (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02942             break;
02943     #if DEBUG
02944     fprintf (stderr, "window_get_algorithm: %u\n", i);
02945     fprintf (stderr, "window_get_algorithm: end\n");
02946     #endif
02947     return i;
02948 }
```

5.5.2.22 int window_get_gradient ()

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line [2956](#) of file [mpcotool.c](#).

```

02957 {
02958     unsigned int i;
02959     #if DEBUG
02960     fprintf (stderr, "window_get_gradient: start\n");
02961     #endif
02962     for (i = 0; i < NGRADIENTS; ++i)
02963         if (gtk_toggle_button_get_active
02964             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02965             break;
02966     #if DEBUG
02967     fprintf (stderr, "window_get_gradient: %u\n", i);
02968     fprintf (stderr, "window_get_gradient: end\n");
02969     #endif
02970     return i;
02971 }
```

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 [4053](#) of file [mpcotool.c](#).

```

04054 {
04055     unsigned int i;
04056     char *buffer;
04057     #if DEBUG
04058     fprintf (stderr, "window_read: start\n");
04059     #endif
```



```

04060
04061 // Reading new input file
04062 input_free ();
04063 if (!input_open (filename))
04064     return 0;
04065
04066 // Setting GTK+ widgets data
04067 gtk_entry_set_text (window->entry_result, input->result);
04068 gtk_entry_set_text (window->entry_variables, input->
variables);
04069 buffer = g_build_filename (input->directory, input->
simulator, NULL);
04070 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
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     g_free (buffer);
04079 }
04080
04081 gtk_toggle_button_set_active
04082 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
04083
04084 switch (input->algorithm)
04085 {
04086     case ALGORITHM_MONTE_CARLO:
04087         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
04088     case ALGORITHM_SWEEP:
04089         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
04090         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
04091         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
04092         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_gradient),
input->nsteps);
04093
04094         if (input->nsteps)
04095         {
04096             gtk_toggle_button_set_active
04097                 (GTK_TOGGLE_BUTTON (window->button_gradient
[input->gradient_method]), TRUE);
04098             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
04099             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
04100             switch (input->gradient_method)
04101             {
04102                 case GRADIENT_METHOD_RANDOM:
04103                     gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
04104             }
04105         }
04106         break;
04107     default:
04108         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
04109         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
04110         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
04111         gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
04112         gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
04113     }
04114 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
04115 g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
04116 gtk_combo_box_text_remove_all (window->combo_experiment);
04117 for (i = 0; i < input->nexperiments; ++i)
04118     gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i]);
04119 g_signal_handler_unblock
04120 (window->button_experiment, window->
id_experiment_name);
04121 g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
04122 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04123 g_signal_handler_block (window->combo_variable, window->

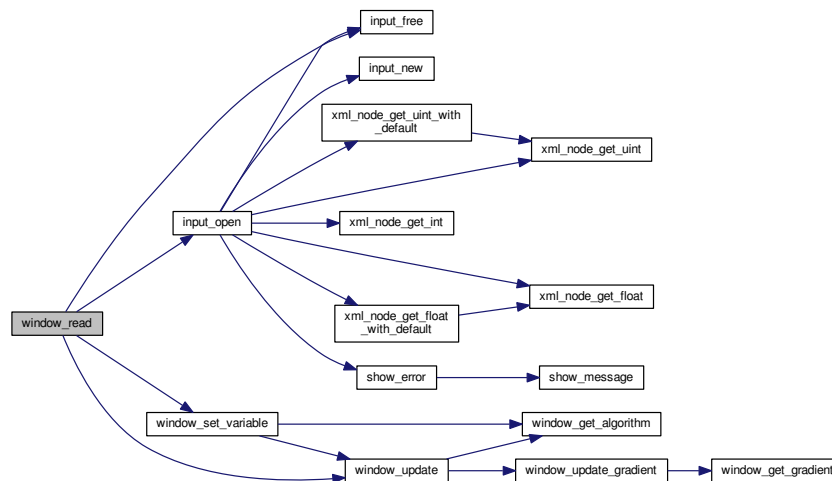
```

```

    id_variable);
04136   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
04137   gtk_combo_box_text_remove_all (window->combo_variable);
04138   for (i = 0; i < input->nvariables; ++i)
04139       gtk_combo_box_text_append_text (window->combo_variable,
    input->label[i]);
04140   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
04141   g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
04142   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04143   window_set_variable ();
04144   window_update ();
04145
04146   #if DEBUG
04147       fprintf (stderr, "window_read: end\n");
04148   #endif
04149   return 1;
04150 }

```

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 3011 of file `mpcotool.c`.

```

03012 {
03013   GtkFileChooserDialog *dlg;
03014   GtkFileFilter *filter;
03015   char *buffer;
03016
03017   #if DEBUG
03018       fprintf (stderr, "window_save: start\n");
03019   #endif
03020
03021   // Opening the saving dialog
03022   dlg = (GtkFileChooserDialog *)
03023       gtk_file_chooser_dialog_new (gettext ("Save file"),
03024                                   window->window,
03025                                   GTK_FILE_CHOOSER_ACTION_SAVE,
03026                                   gettext ("_Cancel"),

```

```

03027                                     GTK_RESPONSE_CANCEL,
03028                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
03029 gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03030 buffer = g_build_filename (input->directory, input->name, NULL);
03031 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03032 g_free (buffer);
03033
03034 // Adding XML filter
03035 filter = (GtkFileFilter *) gtk_file_filter_new ();
03036 gtk_file_filter_set_name (filter, "XML");
03037 gtk_file_filter_add_pattern (filter, "*.xml");
03038 gtk_file_filter_add_pattern (filter, "*.XML");
03039 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03040
03041 // If OK response then saving
03042 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03043 {
03044
03045     // Adding properties to the root XML node
03046     input->simulator = gtk_file_chooser_get_filename
03047         (GTK_FILE_CHOOSER (window->button_simulator));
03048     if (gtk_toggle_button_get_active
03049         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03050         input->evaluator = gtk_file_chooser_get_filename
03051             (GTK_FILE_CHOOSER (window->button_evaluator));
03052     else
03053         input->evaluator = NULL;
03054     input->result
03055         = (char *) xmlStrdup ((const xmlChar *)
03056             gtk_entry_get_text (window->entry_result));
03057     input->variables
03058         = (char *) xmlStrdup ((const xmlChar *)
03059             gtk_entry_get_text (window->entry_variables));
03060
03061     // Setting the algorithm
03062     switch (window_get_algorithm ())
03063     {
03064     case ALGORITHM_MONTE_CARLO:
03065         input->algorithm = ALGORITHM_MONTE_CARLO;
03066         input->nsimulations
03067             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03068         input->niterations
03069             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03070         input->tolerance = gtk_spin_button_get_value (window->
03071 spin_tolerance);
03072         input->nbest = gtk_spin_button_get_value_as_int (window->
03073 spin_bests);
03074         window_save_gradient ();
03075         break;
03076     case ALGORITHM_SWEEP:
03077         input->algorithm = ALGORITHM_SWEEP;
03078         input->niterations
03079             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03080         input->tolerance = gtk_spin_button_get_value (window->
03081 spin_tolerance);
03082         input->nbest = gtk_spin_button_get_value_as_int (window->
03083 spin_bests);
03084         window_save_gradient ();
03085         break;
03086     default:
03087         input->algorithm = ALGORITHM_GENETIC;
03088         input->nsimulations
03089             = gtk_spin_button_get_value_as_int (window->spin_population);
03090         input->niterations
03091             = gtk_spin_button_get_value_as_int (window->spin_generations);
03092         input->mutation_ratio
03093             = gtk_spin_button_get_value (window->spin_mutation);
03094         input->reproduction_ratio
03095             = gtk_spin_button_get_value (window->spin_reproduction);
03096         input->adaptation_ratio
03097             = gtk_spin_button_get_value (window->spin_adaptation);
03098         break;
03099     }
03100
03101     // Saving the XML file
03102     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03103     input_save (buffer);
03104
03105     // Closing and freeing memory
03106     g_free (buffer);
03107     gtk_widget_destroy (GTK_WIDGET (dlg));
03108     #if DEBUG
03109     fprintf (stderr, "window_save: end\n");
03110     #endif
03111     return 1;
03112 }
03113

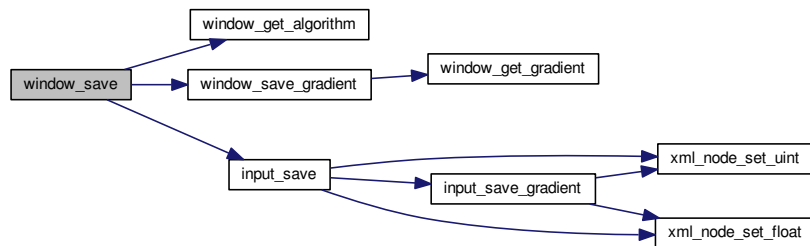
```

```

03110 // Closing and freeing memory
03111 gtk_widget_destroy (GTK_WIDGET (dlg));
03112 #if DEBUG
03113 fprintf (stderr, "window_save: end\n");
03114 #endif
03115 return 0;
03116 }

```

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 3657 of file [mpcotool.c](#).

```

03658 {
03659     unsigned int i, j;
03660     char *buffer;
03661     GFile *file1, *file2;
03662     #if DEBUG
03663     fprintf (stderr, "window_template_experiment: start\n");
03664     #endif
03665     i = (size_t) data;
03666     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03667     file1
03668     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03669     file2 = g_file_new_for_path (input->directory);
03670     buffer = g_file_get_relative_path (file2, file1);
03671     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03672     g_free (buffer);
03673     g_object_unref (file2);
03674     g_object_unref (file1);
03675     #if DEBUG
03676     fprintf (stderr, "window_template_experiment: end\n");
03677     #endif
03678 }

```

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

```

00368 {
00369     double x = 0.;
00370     xmlChar *buffer;
00371     buffer = xmlGetProp (node, prop);
00372     if (!buffer)
00373         *error_code = 1;
00374     else
00375     {
00376         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00377             *error_code = 2;
00378         else
00379             *error_code = 0;
00380         xmlFree (buffer);
00381     }
00382     return x;
00383 }
```

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

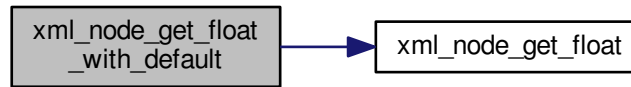
Floating point number value.

Definition at line 401 of file [mpcotool.c](#).

```

00403 {
00404     double x;
00405     if (xmlHasProp (node, prop))
00406         x = xml_node_get_float (node, prop, error_code);
00407     else
00408     {
00409         x = default_value;
00410         *error_code = 0;
00411     }
00412     return x;
00413 }
```

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

```

00276 {
00277     int i = 0;
00278     xmlChar *buffer;
00279     buffer = xmlGetProp (node, prop);
00280     if (!buffer)
00281         *error_code = 1;
00282     else
00283     {
00284         if (sscanf ((char *) buffer, "%d", &i) != 1)
00285             *error_code = 2;
00286         else
00287             *error_code = 0;
00288         xmlFree (buffer);
00289     }
00290     return i;
00291 }
  
```

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

```

00307 {
00308     unsigned int i = 0;
00309     xmlChar *buffer;
00310     buffer = xmlGetProp (node, prop);
00311     if (!buffer)
00312         *error_code = 1;
00313     else
00314     {
00315         if (sscanf ((char *) buffer, "%u", &i) != 1)
00316             *error_code = 2;
00317         else
00318             *error_code = 0;
00319         xmlFree (buffer);
00320     }
00321     return i;
00322 }

```

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.

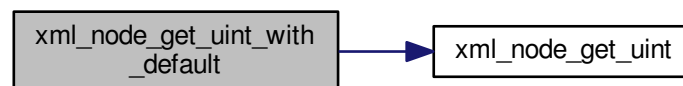
Definition at line 340 of file [mpcotool.c](#).

```

00342 {
00343     unsigned int i;
00344     if (xmlHasProp (node, prop))
00345         i = xml_node_get_uint (node, prop, error_code);
00346     else
00347     {
00348         i = default_value;
00349         *error_code = 0;
00350     }
00351     return i;
00352 }

```

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

```
00465 {
00466     xmlChar buffer[64];
00467     snprintf ((char *) buffer, 64, "%.14lg", value);
00468     xmlSetProp (node, prop, buffer);
00469 }
```

5.5.2.32 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 426 of file [mpcotool.c](#).

```
00427 {
00428     xmlChar buffer[64];
00429     snprintf ((char *) buffer, 64, "%d", value);
00430     xmlSetProp (node, prop, buffer);
00431 }
```

5.5.2.33 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 445 of file [mpcotool.c](#).

```
00446 {
00447     xmlChar buffer[64];
00448     snprintf ((char *) buffer, 64, "%u", value);
00449     xmlSetProp (node, prop, buffer);
00450 }
```

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

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

Array of C-strings with variable formats.

Definition at line 118 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 123 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     "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00120     "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
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 /*
00168 const char * logo[] = {
00169 "32 32 3 1",
00170 "      c #FFFFFFFFFFFF",
00171 "      c #00000000FFFF",
00172 "X      c #FFFF00000000",
00173 "      .      .      .      .      .      ",
00174 "      .      .      .      .      .      ",
00175 "      .      .      .      .      .      ",
00176 "      .      .      .      .      .      ",
00177 "      .      .      .      .      .      ",
00178 "      .      .      .      .      .      ",
00179 "      .      .      .      .      .      ",
00180 "      .      .      .      .      .      ",
00181 "      .      .      .      .      .      ",
00182 "      .      .      .      .      .      ",
00183 "      .      .      .      .      .      ",
00184 "      .      .      .      .      .      ",
00185 "      .      .      .      .      .      ",
00186 "      .      .      .      .      .      ",
00187 "      .      .      .      .      .      ",
00188 "      .      .      .      .      .      ",
00189 "      .      .      .      .      .      ",
00190 "      .      .      .      .      .      ",
00191 "      .      .      .      .      .      ",
00192 "      .      .      .      .      .      ",
00193 "      .      .      .      .      .      ",
00194 "      .      .      .      .      .      ",
00195 "      .      .      .      .      .      ",
00196 "      .      .      .      .      .      ",
00197 "      .      .      .      .      .      ",
00198 "      .      .      .      .      .      ",
00199 "      .      .      .      .      .      ",
00200 "      .      .      .      .      .      ",
00201 "      .      .      .      .      .      ",
00202 "      .      .      .      .      .      ",
00203 "      .      .      .      .      .      ",
00204 "      .      .      .      .      .      ",
00205 */
00206
00207 #if HAVE_GTK
00208 Options options[1];
00210 Running running[1];
00212 Window window[1];
00214 #endif
00215
00226 void
00227 show_message (char *title, char *msg, int type)
00228 {
00229 #if HAVE_GTK
00230     GtkMessageDialog *dlg;
00231
00232     // Creating the dialog
00233     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00234         (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00235
00236     // Setting the dialog title
00237     gtk_window_set_title (GTK_WINDOW (dlg), title);
00238
00239     // Showing the dialog and waiting response
00240     gtk_dialog_run (GTK_DIALOG (dlg));
00241
00242     // Closing and freeing memory
00243     gtk_widget_destroy (GTK_WIDGET (dlg));
00244
00245 #else
00246     printf ("%s: %s\n", title, msg);
00247 #endif
00248 }
00249
00256 void
00257 show_error (char *msg)

```

```

00258 {
00259     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00260 }
00261
00262 int
00275 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00276 {
00277     int i = 0;
00278     xmlChar *buffer;
00279     buffer = xmlGetProp (node, prop);
00280     if (!buffer)
00281         *error_code = 1;
00282     else
00283     {
00284         if (sscanf ((char *) buffer, "%d", &i) != 1)
00285             *error_code = 2;
00286         else
00287             *error_code = 0;
00288         xmlFree (buffer);
00289     }
00290     return i;
00291 }
00292
00305 unsigned int
00306 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00307 {
00308     unsigned int i = 0;
00309     xmlChar *buffer;
00310     buffer = xmlGetProp (node, prop);
00311     if (!buffer)
00312         *error_code = 1;
00313     else
00314     {
00315         if (sscanf ((char *) buffer, "%u", &i) != 1)
00316             *error_code = 2;
00317         else
00318             *error_code = 0;
00319         xmlFree (buffer);
00320     }
00321     return i;
00322 }
00323
00339 unsigned int
00340 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00341                                unsigned int default_value, int *error_code)
00342 {
00343     unsigned int i;
00344     if (xmlHasProp (node, prop))
00345         i = xml_node_get_uint (node, prop, error_code);
00346     else
00347     {
00348         i = default_value;
00349         *error_code = 0;
00350     }
00351     return i;
00352 }
00353
00366 double
00367 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00368 {
00369     double x = 0.;
00370     xmlChar *buffer;
00371     buffer = xmlGetProp (node, prop);
00372     if (!buffer)
00373         *error_code = 1;
00374     else
00375     {
00376         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00377             *error_code = 2;
00378         else
00379             *error_code = 0;
00380         xmlFree (buffer);
00381     }
00382     return x;
00383 }
00384
00400 double
00401 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00402                                 double default_value, int *error_code)
00403 {
00404     double x;
00405     if (xmlHasProp (node, prop))
00406         x = xml_node_get_float (node, prop, error_code);
00407     else
00408     {
00409         x = default_value;
00410         *error_code = 0;

```

```

00411     }
00412     return x;
00413 }
00414
00425 void
00426 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00427 {
00428     xmlChar buffer[64];
00429     snprintf ((char *) buffer, 64, "%d", value);
00430     xmlSetProp (node, prop, buffer);
00431 }
00432
00444 void
00445 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00446 {
00447     xmlChar buffer[64];
00448     snprintf ((char *) buffer, 64, "%u", value);
00449     xmlSetProp (node, prop, buffer);
00450 }
00451
00463 void
00464 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00465 {
00466     xmlChar buffer[64];
00467     snprintf ((char *) buffer, 64, "%.14lg", value);
00468     xmlSetProp (node, prop, buffer);
00469 }
00470
00475 void
00476 input_new ()
00477 {
00478     unsigned int i;
00479     #if DEBUG
00480     fprintf (stderr, "input_new: start\n");
00481     #endif
00482     input->nvariables = input->nexperiments = input->ninputs = input->
nsteps = 0;
00483     input->simulator = input->evaluator = input->directory = input->
name
00484     = input->result = input->variables = NULL;
00485     input->experiment = input->label = NULL;
00486     input->precision = input->nsweeps = input->nbits = NULL;
00487     input->rangemin = input->rangemax = input->rangeminabs = input->
rangemaxabs
00488     = input->weight = input->step = NULL;
00489     for (i = 0; i < MAX_NINPUTS; ++i)
00490         input->template[i] = NULL;
00491     #if DEBUG
00492     fprintf (stderr, "input_new: end\n");
00493     #endif
00494 }
00495
00500 void
00501 input_free ()
00502 {
00503     unsigned int i, j;
00504     #if DEBUG
00505     fprintf (stderr, "input_free: start\n");
00506     #endif
00507     g_free (input->name);
00508     g_free (input->directory);
00509     for (i = 0; i < input->nexperiments; ++i)
00510     {
00511         xmlFree (input->experiment[i]);
00512         for (j = 0; j < input->ninputs; ++j)
00513             xmlFree (input->template[j][i]);
00514         g_free (input->template[j]);
00515     }
00516     g_free (input->experiment);
00517     for (i = 0; i < input->ninputs; ++i)
00518         g_free (input->template[i]);
00519     for (i = 0; i < input->nvariables; ++i)
00520         xmlFree (input->label[i]);
00521     g_free (input->label);
00522     g_free (input->precision);
00523     g_free (input->rangemin);
00524     g_free (input->rangemax);
00525     g_free (input->rangeminabs);
00526     g_free (input->rangemaxabs);
00527     g_free (input->weight);
00528     g_free (input->step);
00529     g_free (input->nsweeps);
00530     g_free (input->nbits);
00531     xmlFree (input->evaluator);
00532     xmlFree (input->simulator);
00533     xmlFree (input->result);
00534     xmlFree (input->variables);

```

```

00535     input->nexperiments = input->ninputs = input->nvariables = input->
        nsteps = 0;
00536     #if DEBUG
00537         fprintf (stderr, "input_free: end\n");
00538     #endif
00539 }
00540
00541 int
00542 input_open (char *filename)
00543 {
00544     char buffer2[64];
00545     char *buffert[MAX_NINPUTS] =
00546     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00547     xmlDoc *doc;
00548     xmlNode *node, *child;
00549     xmlChar *buffer;
00550     char *msg;
00551     int error_code;
00552     unsigned int i;
00553
00554     #if DEBUG
00555         fprintf (stderr, "input_open: start\n");
00556     #endif
00557
00558     // Resetting input data
00559     buffer = NULL;
00560     input_new ();
00561
00562     // Parsing the input file
00563     #if DEBUG
00564         fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00565     #endif
00566     doc = xmlParseFile (filename);
00567     if (!doc)
00568     {
00569         msg = gettext ("Unable to parse the input file");
00570         goto exit_on_error;
00571     }
00572
00573     // Getting the root node
00574     #if DEBUG
00575         fprintf (stderr, "input_open: getting the root node\n");
00576     #endif
00577     node = xmlDocGetRootElement (doc);
00578     if (xmlStrcmp (node->name, XML_CALIBRATE))
00579     {
00580         msg = gettext ("Bad root XML node");
00581         goto exit_on_error;
00582     }
00583
00584     // Getting results file names
00585     input->result = (char *) xmlGetProp (node, XML_RESULT);
00586     if (!input->result)
00587     {
00588         input->result = (char *) xmlStrdup (result_name);
00589     }
00590     input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00591     if (!input->variables)
00592     {
00593         input->variables = (char *) xmlStrdup (variables_name);
00594     }
00595
00596     // Opening simulator program name
00597     input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00598     if (!input->simulator)
00599     {
00600         msg = gettext ("Bad simulator program");
00601         goto exit_on_error;
00602     }
00603
00604     // Opening evaluator program name
00605     input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00606
00607     // Obtaining pseudo-random numbers generator seed
00608     input->seed
00609     = xml_node_get_uint_with_default (node,
00610     XML_SEED, DEFAULT_RANDOM_SEED,
00611     &error_code);
00612     if (error_code)
00613     {
00614         msg = gettext ("Bad pseudo-random numbers generator seed");
00615         goto exit_on_error;
00616     }
00617
00618     // Opening algorithm
00619     buffer = xmlGetProp (node, XML_ALGORITHM);
00620     if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00621     {
00622         input->algorithm = ALGORITHM_MONTE_CARLO;
00623     }
00624
00625     // Obtaining simulations number

```

```

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

```

```

00714         input->adaptation_ratio
00715         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00716         if (error_code || input->adaptation_ratio < 0.
00717             || input->adaptation_ratio >= 1.)
00718         {
00719             msg = gettext ("Invalid adaptation probability");
00720             goto exit_on_error;
00721         }
00722     }
00723     else
00724     {
00725         msg = gettext ("No adaptation probability");
00726         goto exit_on_error;
00727     }
00728
00729     // Checking survivals
00730     i = input->mutation_ratio * input->nsimulations;
00731     i += input->reproduction_ratio * input->nsimulations;
00732     i += input->adaptation_ratio * input->nsimulations;
00733     if (i > input->nsimulations - 2)
00734     {
00735         msg = gettext
00736             ("No enough survival entities to reproduce the population");
00737         goto exit_on_error;
00738     }
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 }
00762
00763 // Obtaining best number
00764 input->nbest
00765 = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
00766 if (error_code || !input->nbest)
00767 {
00768     msg = gettext ("Invalid best number");
00769     goto exit_on_error;
00770 }
00771
00772 // Obtaining tolerance
00773 input->tolerance
00774 = xml_node_get_float_with_default (node,
XML_TOLERANCE, 0.,
&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
00791     buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00792     if (!xmlStrcmp (buffer, XML_COORDINATES))
00793         input->gradient_method = GRADIENT_METHOD_COORDINATES;
00794     else if (!xmlStrcmp (buffer, XML_RANDOM))
00795     {
00796         input->gradient_method = GRADIENT_METHOD_RANDOM;
00797         input->nestimates

```



```

00798         = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00799         if (error_code || !input->nestimates)
00800         {
00801             msg = gettext ("Invalid estimates number");
00802             goto exit_on_error;
00803         }
00804     }
00805     else
00806     {
00807         msg = gettext ("Unknown method to estimate the gradient");
00808         goto exit_on_error;
00809     }
00810     xmlFree (buffer);
00811     buffer = NULL;
00812     input->relaxation
00813         = xml_node_get_float_with_default (node,
XML_RELAXATION,
00814                                           DEFAULT_RELAXATION, &error_code);
00815     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00816     {
00817         msg = gettext ("Invalid relaxation parameter");
00818         goto exit_on_error;
00819     }
00820 }
00821 else
00822     input->nsteps = 0;
00823 }
00824
00825 // Reading the experimental data
00826 for (child = node->children; child; child = child->next)
00827 {
00828     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833     if (xmlHasProp (child, XML_NAME))
00834         buffer = xmlGetProp (child, XML_NAME);
00835     else
00836     {
00837         snprintf (buffer2, 64, "%s %u: %s",
00838                 gettext ("Experiment"),
00839                 input->nexperiments + 1, gettext ("no data file name"));
00840         msg = buffer2;
00841         goto exit_on_error;
00842     }
00843 #if DEBUG
00844     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845 #endif
00846     input->weight = g_realloc (input->weight,
00847                               (1 + input->nexperiments) * sizeof (double));
00848     input->weight[input->nexperiments]
00849         = xml_node_get_float_with_default (child,
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"));
00901                 msg = buffer2;
00902                 while (i-- > 0)
00903                     xmlFree (buffert[i]);
00904                 goto exit_on_error;
00905             }
00906             input->template[i] = (char **)
00907                 g_realloc (input->template[i],
00908                     (1 + input->nexperiments) * sizeof (char *));
00909             buffert[i] = (char *) xmlGetProp (child, template[i]);
00910 #if DEBUG
00911             fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00912                 input->nexperiments, i + 1,
00913                 input->template[i][input->nexperiments]);
00914 #endif
00915             if (!input->nexperiments)
00916                 ++input->ninputs;
00917 #if DEBUG
00918             fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00919 #endif
00920         }
00921         else if (input->nexperiments && input->ninputs >= i)
00922         {
00923             snprintf (buffer2, 64, "%s %s: %s%u",
00924                 gettext ("Experiment"),
00925                 buffer, gettext ("no template"), i + 1);
00926             msg = buffer2;
00927             while (i-- > 0)
00928                 xmlFree (buffert[i]);
00929             goto exit_on_error;
00930         }
00931         else
00932             break;
00933     }
00934     input->experiment
00935     = g_realloc (input->experiment,
00936         (1 + input->nexperiments) * sizeof (char *));
00937     input->experiment[input->nexperiments] = (char *) buffer;
00938     for (i = 0; i < input->ninputs; ++i)
00939         input->template[i][input->nexperiments] = buffert[i];
00940     ++input->nexperiments;
00941 #if DEBUG
00942     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00943 #endif
00944     }
00945     if (!input->nexperiments)
00946     {
00947         msg = gettext ("No calibration experiments");
00948         goto exit_on_error;
00949     }
00950     buffer = NULL;
00951
00952     // Reading the variables data
00953     for (; child; child = child->next)
00954     {
00955         if (xmlStrcmp (child->name, XML_VARIABLE))
00956         {
00957             snprintf (buffer2, 64, "%s %u: %s",
00958                 gettext ("Variable"),
00959                 input->nvariables + 1, gettext ("bad XML node"));
00960             msg = buffer2;
00961             goto exit_on_error;
00962         }
00963         if (xmlHasProp (child, XML_NAME))
00964             buffer = xmlGetProp (child, XML_NAME);
00965         else
00966         {
00967             snprintf (buffer2, 64, "%s %u: %s",
00968                 gettext ("Variable"),

```

```

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

```

```

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

```

```

01138         g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01139         input->step[input->nvariables]
01140         = xml_node_get_float (child, XML_STEP, &error_code);
01141         if (error_code || input->step[input->nvariables] < 0.)
01142         {
01143             snprintf (buffer2, 64, "%s %s: %s",
01144                     gettext ("Variable"),
01145                     buffer, gettext ("bad step size"));
01146             msg = buffer2;
01147             goto exit_on_error;
01148         }
01149     }
01150     input->label = g_realloc
01151     (input->label, (1 + input->nvariables) * sizeof (char *));
01152     input->label[input->nvariables] = (char *) buffer;
01153     ++input->nvariables;
01154 }
01155 if (!input->nvariables)
01156 {
01157     msg = gettext ("No calibration variables");
01158     goto exit_on_error;
01159 }
01160 buffer = NULL;
01161
01162 // Getting the working directory
01163 input->directory = g_path_get_dirname (filename);
01164 input->name = g_path_get_basename (filename);
01165
01166 // Closing the XML document
01167 xmlFreeDoc (doc);
01168
01169 #if DEBUG
01170 fprintf (stderr, "input_open: end\n");
01171 #endif
01172 return 1;
01173
01174 exit_on_error:
01175 xmlFree (buffer);
01176 xmlFreeDoc (doc);
01177 show_error (msg);
01178 input_free ();
01179 #if DEBUG
01180 fprintf (stderr, "input_open: end\n");
01181 #endif
01182 return 0;
01183 }
01184
01196 void
01197 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01198 {
01199     unsigned int i;
01200     char buffer[32], value[32], *buffer2, *buffer3, *content;
01201     FILE *file;
01202     gsize length;
01203     GRegex *regex;
01204
01205     #if DEBUG
01206     fprintf (stderr, "calibrate_input: start\n");
01207     #endif
01208
01209     // Checking the file
01210     if (!template)
01211         goto calibrate_input_end;
01212
01213     // Opening template
01214     content = g_mapped_file_get_contents (template);
01215     length = g_mapped_file_get_length (template);
01216     #if DEBUG
01217     fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01218             content);
01219     #endif
01220     file = g_fopen (input, "w");
01221
01222     // Parsing template
01223     for (i = 0; i < calibrate->nvariables; ++i)
01224     {
01225         #if DEBUG
01226         fprintf (stderr, "calibrate_input: variable=%u\n", i);
01227         #endif
01228         snprintf (buffer, 32, "@variable%u@", i + 1);
01229         regex = g_regex_new (buffer, 0, 0, NULL);
01230         if (i == 0)
01231         {
01232             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01233                 calibrate->label[i], 0, NULL);
01234             #if DEBUG
01235             fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);

```

```

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

```

```

01333     snprintf (result, 32, "result-%u-%u", simulation, experiment);
01334     buffer2 = g_path_get_dirname (calibrate->evaluator);
01335     buffer3 = g_path_get_basename (calibrate->evaluator);
01336     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01337     snprintf (buffer, 512, "\\\"%s\\\" %s %s %s",
01338             buffer4, output, calibrate->experiment[experiment], result);
01339     g_free (buffer4);
01340     g_free (buffer3);
01341     g_free (buffer2);
01342 #if DEBUG
01343     fprintf (stderr, "calibrate_parse: %s\n", buffer);
01344 #endif
01345     system (buffer);
01346     file_result = g_fopen (result, "r");
01347     e = atof (fgets (buffer, 512, file_result));
01348     fclose (file_result);
01349 }
01350 else
01351 {
01352     strcpy (result, "");
01353     file_result = g_fopen (output, "r");
01354     e = atof (fgets (buffer, 512, file_result));
01355     fclose (file_result);
01356 }
01357
01358 // Removing files
01359 #if !DEBUG
01360 for (i = 0; i < calibrate->ninputs; ++i)
01361 {
01362     if (calibrate->file[i][0])
01363     {
01364         snprintf (buffer, 512, RM " %s", &input[i][0]);
01365         system (buffer);
01366     }
01367 }
01368     snprintf (buffer, 512, RM " %s %s", output, result);
01369     system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373     fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376 // Returning the objective function
01377 return e * calibrate->weight[experiment];
01378 }
01379
01384 void
01385 calibrate_print ()
01386 {
01387     unsigned int i;
01388     char buffer[512];
01389 #if HAVE_MPI
01390     if (calibrate->mpi_rank)
01391         return;
01392 #endif
01393     printf ("%s\n", gettext ("Best result"));
01394     fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01395     printf ("error = %.15le\n", calibrate->error_old[0]);
01396     fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
01397 error_old[0]);
01397     for (i = 0; i < calibrate->nvariables; ++i)
01398     {
01399         snprintf (buffer, 512, "%s = %s\n",
01400             calibrate->label[i], format[calibrate->precision[i]]);
01401         printf (buffer, calibrate->value_old[i]);
01402         fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01403     }
01404     fflush (calibrate->file_result);
01405 }
01406
01415 void
01416 calibrate_save_variables (unsigned int simulation, double error)
01417 {
01418     unsigned int i;
01419     char buffer[64];
01420 #if DEBUG
01421     fprintf (stderr, "calibrate_save_variables: start\n");
01422 #endif
01423     for (i = 0; i < calibrate->nvariables; ++i)
01424     {
01425         snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01426         fprintf (calibrate->file_variables, buffer,
01427             calibrate->value[simulation * calibrate->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 }
01434
01443 void
01444 calibrate_best (unsigned int simulation, double value)
01445 {
01446     unsigned int i, j;
01447     double e;
01448     #if DEBUG
01449         fprintf (stderr, "calibrate_best: start\n");
01450         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01451             calibrate->nsaveds, calibrate->nbest);
01452     #endif
01453     if (calibrate->nsaveds < calibrate->nbest
01454         || value < calibrate->error_best[calibrate->nsaveds - 1])
01455     {
01456         if (calibrate->nsaveds < calibrate->nbest)
01457             ++calibrate->nsaveds;
01458         calibrate->error_best[calibrate->nsaveds - 1] = value;
01459         calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01460         for (i = calibrate->nsaveds; --i;)
01461         {
01462             if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01463             {
01464                 j = calibrate->simulation_best[i];
01465                 e = calibrate->error_best[i];
01466                 calibrate->simulation_best[i] = calibrate->
simulation_best[i - 1];
01467                 calibrate->error_best[i] = calibrate->error_best[i - 1];
01468                 calibrate->simulation_best[i - 1] = j;
01469                 calibrate->error_best[i - 1] = e;
01470             }
01471             else
01472                 break;
01473         }
01474     }
01475     #if DEBUG
01476         fprintf (stderr, "calibrate_best: end\n");
01477     #endif
01478 }
01479
01484 void
01485 calibrate_sequential ()
01486 {
01487     unsigned int i, j;
01488     double e;
01489     #if DEBUG
01490         fprintf (stderr, "calibrate_sequential: start\n");
01491         fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01492             calibrate->nstart, calibrate->nend);
01493     #endif
01494     for (i = calibrate->nstart; i < calibrate->nend; ++i)
01495     {
01496         e = 0.;
01497         for (j = 0; j < calibrate->nexperiments; ++j)
01498             e += calibrate_parse (i, j);
01499         calibrate_best (i, e);
01500         calibrate_save_variables (i, e);
01501     #if DEBUG
01502         fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01503     #endif
01504     }
01505     #if DEBUG
01506         fprintf (stderr, "calibrate_sequential: end\n");
01507     #endif
01508 }
01509
01517 void *
01518 calibrate_thread (ParallelData * data)
01519 {
01520     unsigned int i, j, thread;
01521     double e;
01522     #if DEBUG
01523         fprintf (stderr, "calibrate_thread: start\n");
01524     #endif
01525     thread = data->thread;
01526     #if DEBUG
01527         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01528             calibrate->thread[thread], calibrate->thread[thread + 1]);
01529     #endif
01530     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01531     {
01532         e = 0.;
01533         for (j = 0; j < calibrate->nexperiments; ++j)
01534             e += calibrate_parse (i, j);
01535         g_mutex_lock (mutex);

```



```

01536     calibrate_best (i, e);
01537     calibrate_save_variables (i, e);
01538     g_mutex_unlock (mutex);
01539 #if DEBUG
01540     fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01541 #endif
01542 }
01543 #if DEBUG
01544     fprintf (stderr, "calibrate_thread: end\n");
01545 #endif
01546     g_thread_exit (NULL);
01547     return NULL;
01548 }
01549
01561 void
01562 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01563                 double *error_best)
01564 {
01565     unsigned int i, j, k, s[calibrate->nbest];
01566     double e[calibrate->nbest];
01567 #if DEBUG
01568     fprintf (stderr, "calibrate_merge: start\n");
01569 #endif
01570     i = j = k = 0;
01571     do
01572     {
01573         if (i == calibrate->nsaveds)
01574         {
01575             s[k] = simulation_best[j];
01576             e[k] = error_best[j];
01577             ++j;
01578             ++k;
01579             if (j == nsaveds)
01580                 break;
01581         }
01582         else if (j == nsaveds)
01583         {
01584             s[k] = calibrate->simulation_best[i];
01585             e[k] = calibrate->error_best[i];
01586             ++i;
01587             ++k;
01588             if (i == calibrate->nsaveds)
01589                 break;
01590         }
01591         else if (calibrate->error_best[i] > error_best[j])
01592         {
01593             s[k] = simulation_best[j];
01594             e[k] = error_best[j];
01595             ++j;
01596             ++k;
01597         }
01598         else
01599         {
01600             s[k] = calibrate->simulation_best[i];
01601             e[k] = calibrate->error_best[i];
01602             ++i;
01603             ++k;
01604         }
01605     }
01606     while (k < calibrate->nbest);
01607     calibrate->nsaveds = k;
01608     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01609     memcpy (calibrate->error_best, e, k * sizeof (double));
01610 #if DEBUG
01611     fprintf (stderr, "calibrate_merge: end\n");
01612 #endif
01613 }
01614
01619 #if HAVE_MPI
01620 void
01621 calibrate_synchronise ()
01622 {
01623     unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01624     double error_best[calibrate->nbest];
01625     MPI_Status mpi_stat;
01626 #if DEBUG
01627     fprintf (stderr, "calibrate_synchronise: start\n");
01628 #endif
01629     if (calibrate->mpi_rank == 0)
01630     {
01631         for (i = 1; i < ntasks; ++i)
01632         {
01633             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01634             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01635                      MPI_COMM_WORLD, &mpi_stat);
01636             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01637                      MPI_COMM_WORLD, &mpi_stat);

```

```

01638         calibrate_merge (nsaveds, simulation_best, error_best);
01639     }
01640 }
01641 else
01642 {
01643     MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01644     MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01645             MPI_COMM_WORLD);
01646     MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01647             MPI_COMM_WORLD);
01648 }
01649 #if DEBUG
01650 fprintf (stderr, "calibrate_synchronise: end\n");
01651 #endif
01652 }
01653 #endif
01654
01659 void
01660 calibrate_sweep ()
01661 {
01662     unsigned int i, j, k, l;
01663     double e;
01664     GThread *thread[nthreads];
01665     ParallelData data[nthreads];
01666 #if DEBUG
01667     fprintf (stderr, "calibrate_sweep: start\n");
01668 #endif
01669     for (i = 0; i < calibrate->nsimulations; ++i)
01670     {
01671         k = i;
01672         for (j = 0; j < calibrate->nvariables; ++j)
01673         {
01674             l = k % calibrate->nsweeps[j];
01675             k /= calibrate->nsweeps[j];
01676             e = calibrate->rangemin[j];
01677             if (calibrate->nsweeps[j] > 1)
01678                 e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01679                     / (calibrate->nsweeps[j] - 1);
01680             calibrate->value[i * calibrate->nvariables + j] = e;
01681         }
01682     }
01683     calibrate->nsaveds = 0;
01684     if (nthreads <= 1)
01685         calibrate_sequential ();
01686     else
01687     {
01688         for (i = 0; i < nthreads; ++i)
01689         {
01690             data[i].thread = i;
01691             thread[i]
01692                 = g_thread_new (NULL, (void (*) ) calibrate_thread, &data[i]);
01693         }
01694         for (i = 0; i < nthreads; ++i)
01695             g_thread_join (thread[i]);
01696     }
01697 #if HAVE_MPI
01698     // Communicating tasks results
01699     calibrate_synchronise ();
01700 #endif
01701 #if DEBUG
01702     fprintf (stderr, "calibrate_sweep: end\n");
01703 #endif
01704 }
01705
01710 void
01711 calibrate_MonteCarlo ()
01712 {
01713     unsigned int i, j;
01714     GThread *thread[nthreads];
01715     ParallelData data[nthreads];
01716 #if DEBUG
01717     fprintf (stderr, "calibrate_MonteCarlo: start\n");
01718 #endif
01719     for (i = 0; i < calibrate->nsimulations; ++i)
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 (*)(void*)) calibrate_thread, &data[i]);
01734     }
01735     for (i = 0; i < nthreads; ++i)
01736         g_thread_join (thread[i]);
01737 }
01738 #if HAVE_MPI
01739 // Communicating tasks results
01740 calibrate_synchronise ();
01741 #endif
01742 #if DEBUG
01743 fprintf (stderr, "calibrate_MonteCarlo: end\n");
01744 #endif
01745 }
01746
01756 void
01757 calibrate_best_gradient (unsigned int simulation, double value)
01758 {
01759     #if DEBUG
01760     fprintf (stderr, "calibrate_best_gradient: start\n");
01761     fprintf (stderr,
01762             "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01763             simulation, value, calibrate->error_best[0]);
01764     #endif
01765     if (value < calibrate->error_best[0])
01766     {
01767         calibrate->error_best[0] = value;
01768         calibrate->simulation_best[0] = simulation;
01769     }
01770     #if DEBUG
01771     fprintf (stderr,
01772             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01773             simulation, value);
01774     #endif
01775     #if DEBUG
01776     fprintf (stderr, "calibrate_best_gradient: end\n");
01777     #endif
01778 }
01779
01786 void
01787 calibrate_gradient_sequential (unsigned int simulation)
01788 {
01789     unsigned int i, j, k;
01790     double e;
01791     #if DEBUG
01792     fprintf (stderr, "calibrate_gradient_sequential: start\n");
01793     fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01794             "nend_gradient=%u\n",
01795             calibrate->nstart_gradient, calibrate->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     }
01806     #if DEBUG
01807     fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01808     #endif
01809     #if DEBUG
01810     fprintf (stderr, "calibrate_gradient_sequential: end\n");
01811     #endif
01812 }
01813
01821 void *
01822 calibrate_gradient_thread (ParallelData * data)
01823 {
01824     unsigned int i, j, thread;
01825     double e;
01826     #if DEBUG
01827     fprintf (stderr, "calibrate_gradient_thread: start\n");
01828     #endif
01829     thread = data->thread;
01830     #if DEBUG
01831     fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01832             thread,
01833             calibrate->thread_gradient[thread],
01834             calibrate->thread_gradient[thread + 1]);
01835     #endif
01836     for (i = calibrate->thread_gradient[thread];
01837          i < calibrate->thread_gradient[thread + 1]; ++i)
01838     {
01839         e = 0.;
01840         for (j = 0; j < calibrate->nexperiments; ++j)
01841             e += calibrate_parse (i, j);
01842     }

```

```

01842     g_mutex_lock (mutex);
01843     calibrate_best_gradient (i, e);
01844     calibrate_save_variables (i, e);
01845     g_mutex_unlock (mutex);
01846     #if DEBUG
01847     fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01848     #endif
01849     }
01850     #if DEBUG
01851     fprintf (stderr, "calibrate_gradient_thread: end\n");
01852     #endif
01853     g_thread_exit (NULL);
01854     return NULL;
01855 }
01856
01866 double
01867 calibrate_estimate_gradient_random (unsigned int variable,
01868                                     unsigned int estimate)
01869 {
01870     double x;
01871     #if DEBUG
01872     fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01873     #endif
01874     x = calibrate->gradient[variable]
01875         + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
01876     #if DEBUG
01877     fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01878             variable, x);
01879     fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01880     #endif
01881     return x;
01882 }
01883
01893 double
01894 calibrate_estimate_gradient_coordinates (unsigned int variable,
01895                                         unsigned int estimate)
01896 {
01897     double x;
01898     #if DEBUG
01899     fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01900     #endif
01901     x = calibrate->gradient[variable];
01902     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01903     {
01904         if (estimate & 1)
01905             x += calibrate->step[variable];
01906         else
01907             x -= calibrate->step[variable];
01908     }
01909     #if DEBUG
01910     fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01911             variable, x);
01912     fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01913     #endif
01914     return x;
01915 }
01916
01923 void
01924 calibrate_step_gradient (unsigned int simulation)
01925 {
01926     GThread *thread[nthreads_gradient];
01927     ParallelData data[nthreads_gradient];
01928     unsigned int i, j, k, b;
01929     #if DEBUG
01930     fprintf (stderr, "calibrate_step_gradient: start\n");
01931     #endif
01932     for (i = 0; i < calibrate->nestimates; ++i)
01933     {
01934         k = (simulation + i) * calibrate->nvariables;
01935         b = calibrate->simulation_best[0] * calibrate->nvariables;
01936         #if DEBUG
01937         fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01938                 simulation + i, calibrate->simulation_best[0]);
01939         #endif
01940         for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01941         {
01942             #if DEBUG
01943             fprintf (stderr,
01944                     "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01945                     i, j, calibrate->value[b]);
01946             #endif
01947             calibrate->value[k]
01948                 = calibrate->value[b] + calibrate_estimate_gradient (j, i);
01949             calibrate->value[k] = fmin (fmax (calibrate->value[k],
01950                                             calibrate->rangeminabs[j]),
01951                                       calibrate->rangemaxabs[j]);
01952             #if DEBUG

```

```

01953         fprintf (stderr,
01954                 "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01955                 i, j, calibrate->value[k]);
01956     #endif
01957     }
01958     }
01959     if (nthreads_gradient == 1)
01960         calibrate_gradient_sequential (simulation);
01961     else
01962     {
01963         for (i = 0; i <= nthreads_gradient; ++i)
01964         {
01965             calibrate->thread_gradient[i]
01966                 = simulation + calibrate->nstart_gradient
01967                 + i * (calibrate->nend_gradient - calibrate->
01968                     nstart_gradient)
01969                 / nthreads_gradient;
01970         #if DEBUG
01971             fprintf (stderr,
01972                     "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01973                     i, calibrate->thread_gradient[i]);
01974         #endif
01975         }
01976         for (i = 0; i < nthreads_gradient; ++i)
01977         {
01978             data[i].thread = i;
01979             thread[i] = g_thread_new
01980                 (NULL, (void (*) ) calibrate_gradient_thread, &data[i]);
01981         }
01982         for (i = 0; i < nthreads_gradient; ++i)
01983             g_thread_join (thread[i]);
01984     }
01985     #if DEBUG
01986     fprintf (stderr, "calibrate_step_gradient: end\n");
01987     #endif
01988 }
01989 void
01990 calibrate_gradient ()
01991 {
01992     unsigned int i, j, k, b, s, adjust;
01993     #if DEBUG
01994     fprintf (stderr, "calibrate_gradient: start\n");
01995     #endif
01996     for (i = 0; i < calibrate->nvariables; ++i)
01997         calibrate->gradient[i] = 0.;
01998     b = calibrate->simulation_best[0] * calibrate->nvariables;
01999     s = calibrate->nsimulations;
02000     adjust = 1;
02001     for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
02002     {
02003         #if DEBUG
02004         fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
02005                 i, calibrate->simulation_best[0]);
02006         #endif
02007         calibrate_step_gradient (s);
02008         k = calibrate->simulation_best[0] * calibrate->nvariables;
02009         #if DEBUG
02010         fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
02011                 i, calibrate->simulation_best[0]);
02012         #endif
02013         if (k == b)
02014         {
02015             if (adjust)
02016             {
02017                 for (j = 0; j < calibrate->nvariables; ++j)
02018                     calibrate->step[j] *= 0.5;
02019                 for (j = 0; j < calibrate->nvariables; ++j)
02020                     calibrate->gradient[j] = 0.;
02021                 adjust = 1;
02022             }
02023             else
02024             {
02025                 for (j = 0; j < calibrate->nvariables; ++j)
02026                 {
02027                     #if DEBUG
02028                     fprintf (stderr,
02029                             "calibrate_gradient: best=%u old%u=%.14le\n",
02030                             j, calibrate->value[k + j], j, calibrate->value[b + j]);
02031                     #endif
02032                     calibrate->gradient[j]
02033                         = (1. - calibrate->relaxation) * calibrate->gradient[j]
02034                         + calibrate->relaxation
02035                         * (calibrate->value[k + j] - calibrate->value[b + j]);
02036                     #if DEBUG
02037                     fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02038                             j, calibrate->gradient[j]);
02039                     #endif
02040                 }
02041             }
02042         }
02043     }

```

```

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

```

```

02144 {
02145     unsigned int i, j;
02146     #if DEBUG
02147         fprintf (stderr, "calibrate_save_old: start\n");
02148         fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02149     #endif
02150     memcpy (calibrate->error_old, calibrate->error_best,
02151             calibrate->nbest * sizeof (double));
02152     for (i = 0; i < calibrate->nbest; ++i)
02153     {
02154         j = calibrate->simulation_best[i];
02155         #if DEBUG
02156             fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02157         #endif
02158         memcpy (calibrate->value_old + i * calibrate->nvariables,
02159                 calibrate->value + j * calibrate->nvariables,
02160                 calibrate->nvariables * sizeof (double));
02161     }
02162     #if DEBUG
02163         for (i = 0; i < calibrate->nvariables; ++i)
02164             fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02165                     i, calibrate->value_old[i]);
02166         fprintf (stderr, "calibrate_save_old: end\n");
02167     #endif
02168 }
02169
02170 void
02171 calibrate_merge_old ()
02172 {
02173     unsigned int i, j, k;
02174     double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
02175 nbest],
02176          *enew, *eold;
02177     #if DEBUG
02178         fprintf (stderr, "calibrate_merge_old: start\n");
02179     #endif
02180     anew = calibrate->error_best;
02181     eold = calibrate->error_old;
02182     i = j = k = 0;
02183     do
02184     {
02185         if (*enew < *eold)
02186         {
02187             memcpy (v + k * calibrate->nvariables,
02188                     calibrate->value
02189                     + calibrate->simulation_best[i] * calibrate->
02190 nvariables,
02191                     calibrate->nvariables * sizeof (double));
02192             e[k] = *enew;
02193             ++k;
02194             ++enew;
02195             ++i;
02196         }
02197         else
02198         {
02199             memcpy (v + k * calibrate->nvariables,
02200                     calibrate->value_old + j * calibrate->nvariables,
02201                     calibrate->nvariables * sizeof (double));
02202             e[k] = *eold;
02203             ++k;
02204             ++eold;
02205             ++j;
02206         }
02207     }
02208     while (k < calibrate->nbest);
02209     memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02210     memcpy (calibrate->error_old, e, k * sizeof (double));
02211     #if DEBUG
02212         fprintf (stderr, "calibrate_merge_old: end\n");
02213     #endif
02214 }
02215
02216 void
02217 calibrate_refine ()
02218 {
02219     unsigned int i, j;
02220     double d;
02221     #if HAVE_MPI
02222     MPI_Status mpi_stat;
02223     #endif
02224     #if DEBUG
02225         fprintf (stderr, "calibrate_refine: start\n");
02226     #endif
02227     #if HAVE_MPI
02228         if (!calibrate->mpi_rank)
02229         {
02230

```

```

02239     for (j = 0; j < calibrate->nvariables; ++j)
02240     {
02241         calibrate->rangemin[j] = calibrate->rangemax[j]
02242         = calibrate->value_old[j];
02243     }
02244     for (i = 0; ++i < calibrate->nbest;)
02245     {
02246         for (j = 0; j < calibrate->nvariables; ++j)
02247         {
02248             calibrate->rangemin[j]
02249             = fmin (calibrate->rangemin[j],
02250                     calibrate->value_old[i * calibrate->nvariables + j]);
02251             calibrate->rangemax[j]
02252             = fmax (calibrate->rangemax[j],
02253                     calibrate->value_old[i * calibrate->nvariables + j]);
02254         }
02255     }
02256     for (j = 0; j < calibrate->nvariables; ++j)
02257     {
02258         d = calibrate->tolerance
02259           * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02260         switch (calibrate->algorithm)
02261         {
02262             case ALGORITHM_MONTE_CARLO:
02263                 d *= 0.5;
02264                 break;
02265             default:
02266                 if (calibrate->nsweeps[j] > 1)
02267                     d /= calibrate->nsweeps[j] - 1;
02268                 else
02269                     d = 0.;
02270         }
02271         calibrate->rangemin[j] -= d;
02272         calibrate->rangemin[j]
02273         = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02274         calibrate->rangemax[j] += d;
02275         calibrate->rangemax[j]
02276         = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02277         printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02278                calibrate->rangemin[j], calibrate->rangemax[j]);
02279         fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02280                 calibrate->label[j], calibrate->rangemin[j],
02281                 calibrate->rangemax[j]);
02282     }
02283     #if HAVE_MPI
02284     for (i = 1; i < ntasks; ++i)
02285     {
02286         MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02287                  1, MPI_COMM_WORLD);
02288         MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02289                  1, MPI_COMM_WORLD);
02290     }
02291     }
02292     else
02293     {
02294         MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02295                  MPI_COMM_WORLD, &mpi_stat);
02296         MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02297                  MPI_COMM_WORLD, &mpi_stat);
02298     }
02299     #endif
02300     #if DEBUG
02301     fprintf (stderr, "calibrate_refine: end\n");
02302     #endif
02303 }
02304
02305 void
02310 calibrate_step ()
02311 {
02312     #if DEBUG
02313     fprintf (stderr, "calibrate_step: start\n");
02314     #endif
02315     calibrate_algorithm ();
02316     if (calibrate->nsteps)
02317         calibrate_gradient ();
02318     #if DEBUG
02319     fprintf (stderr, "calibrate_step: end\n");
02320     #endif
02321 }
02322
02323 void
02328 calibrate_iterate ()
02329 {
02330     unsigned int i;
02331     #if DEBUG
02332     fprintf (stderr, "calibrate_iterate: start\n");
02333     #endif

```



```

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

```

```

02429     calibrate->algorithm = input->algorithm;
02430     switch (calibrate->algorithm)
02431     {
02432         case ALGORITHM_MONTE_CARLO:
02433             calibrate_algorithm = calibrate_MonteCarlo;
02434             break;
02435         case ALGORITHM_SWEEP:
02436             calibrate_algorithm = calibrate_sweep;
02437             break;
02438         default:
02439             calibrate_algorithm = calibrate_genetic;
02440             calibrate->mutation_ratio = input->mutation_ratio;
02441             calibrate->reproduction_ratio = input->
reproduction_ratio;
02442             calibrate->adaptation_ratio = input->adaptation_ratio;
02443     }
02444     calibrate->nvariables = input->nvariables;
02445     calibrate->nsimulations = input->nsimulations;
02446     calibrate->niterations = input->niterations;
02447     calibrate->nbest = input->nbest;
02448     calibrate->tolerance = input->tolerance;
02449     calibrate->nsteps = input->nsteps;
02450     calibrate->nestimates = 0;
02451     if (input->nsteps)
02452     {
02453         calibrate->gradient_method = input->gradient_method;
02454         calibrate->relaxation = input->relaxation;
02455         switch (input->gradient_method)
02456         {
02457             case GRADIENT_METHOD_COORDINATES:
02458                 calibrate->nestimates = 2 * calibrate->nvariables;
02459                 calibrate_estimate_gradient =
calibrate_estimate_gradient_coordinates;
02460                 break;
02461             default:
02462                 calibrate->nestimates = input->nestimates;
02463                 calibrate_estimate_gradient =
calibrate_estimate_gradient_random;
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     // Reading the variables data
02508     #if DEBUG

```

```

02513     fprintf (stderr, "calibrate_open: reading variables\n");
02514 #endif
02515     calibrate->label = input->label;
02516     j = input->nvariables * sizeof (double);
02517     calibrate->rangemin = (double *) g_malloc (j);
02518     calibrate->rangemax = (double *) g_malloc (j);
02519     memcpy (calibrate->rangemin, input->rangemin, j);
02520     memcpy (calibrate->rangemax, input->rangemax, j);
02521     calibrate->rangeminabs = input->rangeminabs;
02522     calibrate->rangemaxabs = input->rangemaxabs;
02523     calibrate->precision = input->precision;
02524     calibrate->nsweeps = input->nsweeps;
02525     calibrate->step = input->step;
02526     nbits = input->nbits;
02527     if (input->algorithm == ALGORITHM_SWEEP)
02528     {
02529         calibrate->nsimulations = 1;
02530         for (i = 0; i < input->nvariables; ++i)
02531         {
02532             if (input->algorithm == ALGORITHM_SWEEP)
02533             {
02534                 calibrate->nsimulations *= input->nsweeps[i];
02535             }
02536             if (input->algorithm == ALGORITHM_SWEEP)
02537             {
02538                 fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
02539                     calibrate->nsweeps[i], calibrate->nsimulations);
02540             }
02541         }
02542         if (calibrate->nsteps)
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             #endif
02582             calibrate->nstart = calibrate->mpi_rank * calibrate->
02583                 nsimulations / ntasks;
02584             calibrate->nend
02585                 = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
02586                 ntasks;
02587             if (calibrate->nsteps)
02588             {
02589                 calibrate->nstart_gradient
02590                     = calibrate->mpi_rank * calibrate->nestimates / ntasks;
02591                 calibrate->nend_gradient
02592                     = (1 + calibrate->mpi_rank) * calibrate->nestimates /
02593                     ntasks;
02594             }
02595             #else
02596             calibrate->nstart = 0;
02597             calibrate->nend = calibrate->nsimulations;
02598         }

```

```

02595     if (calibrate->nsteps)
02596     {
02597         calibrate->nstart_gradient = 0;
02598         calibrate->nend_gradient = calibrate->nestimates;
02599     }
02600 #endif
02601 #if DEBUG
02602     fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02603             calibrate->nend);
02604 #endif
02605
02606     // Calculating simulations to perform for each thread
02607     calibrate->thread
02608     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02609     for (i = 0; i <= nthreads; ++i)
02610     {
02611         calibrate->thread[i] = calibrate->nstart
02612             + i * (calibrate->nend - calibrate->nstart) / nthreads;
02613 #if DEBUG
02614         fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02615                 calibrate->thread[i]);
02616 #endif
02617     }
02618     if (calibrate->nsteps)
02619         calibrate->thread_gradient = (unsigned int *)
02620             alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02621
02622     // Opening result files
02623     calibrate->file_result = g_fopen (calibrate->result, "w");
02624     calibrate->file_variables = g_fopen (calibrate->variables, "w");
02625
02626     // Performing the algorithm
02627     switch (calibrate->algorithm)
02628     {
02629         // Genetic algorithm
02630         case ALGORITHM_GENETIC:
02631             calibrate_genetic ();
02632             break;
02633
02634         // Iterative algorithm
02635         default:
02636             calibrate_iterate ();
02637     }
02638
02639     // Getting calculation time
02640     t = g_date_time_new_now (tz);
02641     calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02642     g_date_time_unref (t);
02643     g_date_time_unref (t0);
02644     g_time_zone_unref (tz);
02645     printf ("%s = %.6lg s\n",
02646            gettext ("Calculation time"), calibrate->calculation_time);
02647     fprintf (calibrate->file_result, "%s = %.6lg s\n",
02648            gettext ("Calculation time"), calibrate->calculation_time);
02649
02650     // Closing result files
02651     fclose (calibrate->file_variables);
02652     fclose (calibrate->file_result);
02653
02654 #if DEBUG
02655     fprintf (stderr, "calibrate_open: end\n");
02656 #endif
02657 }
02658
02659 #if HAVE_GTK
02660
02661 void
02662 input_save_gradient (xmlNode * node)
02663 {
02664     #if DEBUG
02665         fprintf (stderr, "input_save_gradient: start\n");
02666     #endif
02667     if (input->nsteps)
02668     {
02669         xml_node_set_uint (node, XML_NSTEPS, input->
02670 nsteps);
02671         if (input->relaxation != DEFAULT_RELAXATION)
02672             xml_node_set_float (node, XML_RELAXATION, input->
02673 relaxation);
02674         switch (input->gradient_method)
02675         {
02676             case GRADIENT_METHOD_COORDINATES:
02677                 xmlSetProp (node, XML_GRADIENT_METHOD,
02678 XML_COORDINATES);
02679                 break;
02680             default:
02681                 xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02682         }
02683     }
02684 }

```

```

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

```

```

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

```

```

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

```

```

02959 #if DEBUG
02960     fprintf (stderr, "window_get_gradient: start\n");
02961 #endif
02962     for (i = 0; i < NGRADIENTS; ++i)
02963         if (gtk_toggle_button_get_active
02964             (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02965             break;
02966 #if DEBUG
02967     fprintf (stderr, "window_get_gradient: %u\n", i);
02968     fprintf (stderr, "window_get_gradient: end\n");
02969 #endif
02970     return i;
02971 }
02972
02973 void
02974 window_save_gradient ()
02975 {
02976     #if DEBUG
02977     fprintf (stderr, "window_save_gradient: start\n");
02978     #endif
02979     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02980     {
02981         input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02982         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
02983         switch (window_get_gradient ())
02984         {
02985             case GRADIENT_METHOD_COORDINATES:
02986                 input->gradient_method = GRADIENT_METHOD_COORDINATES;
02987                 break;
02988             default:
02989                 input->gradient_method = GRADIENT_METHOD_RANDOM;
02990                 input->nestimates
= gtk_spin_button_get_value_as_int (window->spin_estimates);
02991         }
02992     }
02993     else
02994         input->nsteps = 0;
02995     #if DEBUG
02996     fprintf (stderr, "window_save_gradient: end\n");
02997     #endif
02998 }
02999
03000 int
03001 window_save ()
03002 {
03003     GtkFileChooserDialog *dlg;
03004     GtkFileFilter *filter;
03005     char *buffer;
03006
03007     #if DEBUG
03008     fprintf (stderr, "window_save: start\n");
03009     #endif
03010
03011     // Opening the saving dialog
03012     dlg = (GtkFileChooserDialog *)
gtk_file_chooser_dialog_new (gettext ("Save file"),
03013                             window->window,
03014                             GTK_FILE_CHOOSER_ACTION_SAVE,
03015                             gettext ("Cancel"),
03016                             GTK_RESPONSE_CANCEL,
03017                             gettext ("OK"), GTK_RESPONSE_OK, NULL);
03018     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03019     buffer = g_build_filename (input->directory, input->name, NULL);
03020     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03021     g_free (buffer);
03022
03023     // Adding XML filter
03024     filter = (GtkFileFilter *) gtk_file_filter_new ();
03025     gtk_file_filter_set_name (filter, "XML");
03026     gtk_file_filter_add_pattern (filter, "*.xml");
03027     gtk_file_filter_add_pattern (filter, "*.XML");
03028     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03029
03030     // If OK response then saving
03031     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03032     {
03033         // Adding properties to the root XML node
03034         input->simulator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_simulator));
03035         if (gtk_toggle_button_get_active
(GTK_TOGGLE_BUTTON (window->check_evaluator)))
03036             input->evaluator = gtk_file_chooser_get_filename
(GTK_FILE_CHOOSER (window->button_evaluator));
03037         else
03038             input->evaluator = NULL;
03039     }

```



```

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

```

```

03141     gtk_widget_destroy (GTK_WIDGET (running->dialog));
03142     snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03143     msg2 = g_strdup (buffer);
03144     for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03145     {
03146         snprintf (buffer, 64, "%s = %s\n",
03147                 calibrate->label[i], format[calibrate->precision[i]]);
03148         snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03149         msg = g_strconcat (msg2, buffer2, NULL);
03150         g_free (msg2);
03151     }
03152     snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03153             calibrate->calculation_time);
03154     msg = g_strconcat (msg2, buffer, NULL);
03155     g_free (msg2);
03156     show_message (gettext ("Best result"), msg, INFO_TYPE);
03157     g_free (msg);
03158     calibrate_free ();
03159 #if DEBUG
03160     fprintf (stderr, "window_run: end\n");
03161 #endif
03162 }
03163
03164 void
03165 window_help ()
03166 {
03167     char *buffer, *buffer2;
03168 #if DEBUG
03169     fprintf (stderr, "window_help: start\n");
03170 #endif
03171     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03172                               gettext ("user-manual.pdf"), NULL);
03173     buffer = g_filename_to_uri (buffer2, NULL, NULL);
03174     g_free (buffer2);
03175     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03176 #if DEBUG
03177     fprintf (stderr, "window_help: uri=%s\n", buffer);
03178 #endif
03179     g_free (buffer);
03180 #if DEBUG
03181     fprintf (stderr, "window_help: end\n");
03182 #endif
03183 }
03184
03185 void
03186 window_about ()
03187 {
03188     static const gchar *authors[] = {
03189         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03190         "Borja Latorre Garcés <borja.latorre@csic.es>",
03191         NULL
03192     };
03193 #if DEBUG
03194     fprintf (stderr, "window_about: start\n");
03195 #endif
03196     gtk_show_about_dialog
03197     (window->window,
03198      "program_name", "MPCOTool",
03199      "comments",
03200      gettext ("A software to perform calibrations/optimizations of empirical "
03201              "parameters"),
03202      "authors", authors,
03203      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03204      "version", "1.2.4",
03205      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03206      "logo", window->logo,
03207      "website", "https://github.com/jburguete/mpcotool",
03208      "license-type", GTK_LICENSE_BSD, NULL);
03209 #if DEBUG
03210     fprintf (stderr, "window_about: end\n");
03211 #endif
03212 }
03213
03214 void
03215 window_update_gradient ()
03216 {
03217 #if DEBUG
03218     fprintf (stderr, "window_update_gradient: start\n");
03219 #endif
03220     gtk_widget_show (GTK_WIDGET (window->check_gradient));
03221     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03222     {
03223         gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03224         gtk_widget_show (GTK_WIDGET (window->label_step));
03225         gtk_widget_show (GTK_WIDGET (window->spin_step));
03226     }
03227     switch (window_get_gradient ())

```

```

03241     {
03242     case GRADIENT_METHOD_COORDINATES:
03243         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03244         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03245         break;
03246     default:
03247         gtk_widget_show (GTK_WIDGET (window->label_estimates));
03248         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03249     }
03250 #if DEBUG
03251     fprintf (stderr, "window_update_gradient: end\n");
03252 #endif
03253 }
03254
03255 void
03260 window_update ()
03261 {
03262     unsigned int i;
03263 #if DEBUG
03264     fprintf (stderr, "window_update: start\n");
03265 #endif
03266     gtk_widget_set_sensitive
03267         (GTK_WIDGET (window->button_evaluator),
03268          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03269                                         (window->check_evaluator)));
03270     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03271     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03272     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03273     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03274     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03275     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03276     gtk_widget_hide (GTK_WIDGET (window->label_bests));
03277     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03278     gtk_widget_hide (GTK_WIDGET (window->label_population));
03279     gtk_widget_hide (GTK_WIDGET (window->spin_population));
03280     gtk_widget_hide (GTK_WIDGET (window->label_generations));
03281     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03282     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03283     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03284     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03285     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03286     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03287     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03288     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03289     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03290     gtk_widget_hide (GTK_WIDGET (window->label_bits));
03291     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03292     gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03293     gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03294     gtk_widget_hide (GTK_WIDGET (window->label_step));
03295     gtk_widget_hide (GTK_WIDGET (window->spin_step));
03296     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03297     switch (window_get_algorithm ())
03298     {
03299     case ALGORITHM_MONTE_CARLO:
03300         gtk_widget_show (GTK_WIDGET (window->label_simulations));
03301         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03302         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03303         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03304         if (i > 1)
03305         {
03306             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03307             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03308             gtk_widget_show (GTK_WIDGET (window->label_bests));
03309             gtk_widget_show (GTK_WIDGET (window->spin_bests));
03310         }
03311         window_update_gradient ();
03312         break;
03313     case ALGORITHM_SWEEP:
03314         gtk_widget_show (GTK_WIDGET (window->label_iterations));
03315         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03316         if (i > 1)
03317         {
03318             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03319             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03320             gtk_widget_show (GTK_WIDGET (window->label_bests));
03321             gtk_widget_show (GTK_WIDGET (window->spin_bests));
03322         }
03323         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03324         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03325         gtk_widget_show (GTK_WIDGET (window->check_gradient));
03326         window_update_gradient ();
03327         break;
03328     default:
03329         gtk_widget_show (GTK_WIDGET (window->label_population));
03330         gtk_widget_show (GTK_WIDGET (window->spin_population));
03331         gtk_widget_show (GTK_WIDGET (window->label_generations));

```

```

03332     gtk_widget_show (GTK_WIDGET (window->spin_generations));
03333     gtk_widget_show (GTK_WIDGET (window->label_mutation));
03334     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03335     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03336     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03337     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03338     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03339     gtk_widget_show (GTK_WIDGET (window->label_bits));
03340     gtk_widget_show (GTK_WIDGET (window->spin_bits));
03341 }
03342 gtk_widget_set_sensitive
03343 (GTK_WIDGET (window->button_remove_experiment), input->
n experiments > 1);
03344 gtk_widget_set_sensitive
03345 (GTK_WIDGET (window->button_remove_variable), input->
n variables > 1);
03346 for (i = 0; i < input->ninputs; ++i)
03347 {
03348     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03349     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03350     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03351     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03352     g_signal_handler_block
03353 (window->check_template[i], window->id_template[i]);
03354     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03355     gtk_toggle_button_set_active
03356 (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03357     g_signal_handler_unblock
03358 (window->button_template[i], window->id_input[i]);
03359     g_signal_handler_unblock
03360 (window->check_template[i], window->id_template[i]);
03361 }
03362 if (i > 0)
03363 {
03364     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03365     gtk_widget_set_sensitive
03366 (GTK_WIDGET (window->button_template[i - 1]),
03367      gtk_toggle_button_get_active
03368      GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
03369 }
03370 if (i < MAX_NINPUTS)
03371 {
03372     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03373     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03374     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03375     gtk_widget_set_sensitive
03376 (GTK_WIDGET (window->button_template[i]),
03377      gtk_toggle_button_get_active
03378      GTK_TOGGLE_BUTTON (window->check_template[i]));
03379     g_signal_handler_block
03380 (window->check_template[i], window->id_template[i]);
03381     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
03382     gtk_toggle_button_set_active
03383 (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03384     g_signal_handler_unblock
03385 (window->button_template[i], window->id_input[i]);
03386     g_signal_handler_unblock
03387 (window->check_template[i], window->id_template[i]);
03388 }
03389 while (++i < MAX_NINPUTS)
03390 {
03391     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03392     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03393 }
03394 gtk_widget_set_sensitive
03395 (GTK_WIDGET (window->spin_minabs),
03396  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03397 gtk_widget_set_sensitive
03398 (GTK_WIDGET (window->spin_maxabs),
03399  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03400 #if DEBUG
03401 fprintf (stderr, "window_update: end\n");
03402 #endif
03403 }
03404
03409 void
03410 window_set_algorithm ()
03411 {
03412     int i;
03413     #if DEBUG
03414     fprintf (stderr, "window_set_algorithm: start\n");
03415     #endif
03416     i = window_get_algorithm ();
03417     switch (i)
03418     {

```

```

03419     case ALGORITHM_SWEEP:
03420         input->nsweeps = (unsigned int *) g_realloc
03421             (input->nsweeps, input->nvariables * sizeof (unsigned int));
03422         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03423         if (i < 0)
03424             i = 0;
03425         gtk_spin_button_set_value (window->spin_sweeps,
03426             (gdouble) input->nsweeps[i]);
03427         break;
03428     case ALGORITHM_GENETIC:
03429         input->nbits = (unsigned int *) g_realloc
03430             (input->nbits, input->nvariables * sizeof (unsigned int));
03431         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03432         if (i < 0)
03433             i = 0;
03434         gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
03435             nbits[i]);
03436         window_update ();
03437         #if DEBUG
03438             fprintf (stderr, "window_set_algorithm: end\n");
03439         #endif
03440     }
03441
03442 void
03443 window_set_experiment ()
03444 {
03445     unsigned int i, j;
03446     char *buffer1, *buffer2;
03447     #if DEBUG
03448         fprintf (stderr, "window_set_experiment: start\n");
03449     #endif
03450     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03451     gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03452     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03453     buffer2 = g_build_filename (input->directory, buffer1, NULL);
03454     g_free (buffer1);
03455     g_signal_handler_block
03456         (window->button_experiment, window->id_experiment_name);
03457     gtk_file_chooser_set_filename
03458         (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03459     g_signal_handler_unblock
03460         (window->button_experiment, window->id_experiment_name);
03461     g_free (buffer2);
03462     for (j = 0; j < input->ninputs; ++j)
03463     {
03464         g_signal_handler_block (window->button_template[j], window->
03465             id_input[j]);
03466         buffer2
03467             = g_build_filename (input->directory, input->template[j][i], NULL);
03468         gtk_file_chooser_set_filename
03469             (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03470         g_free (buffer2);
03471         g_signal_handler_unblock
03472             (window->button_template[j], window->id_input[j]);
03473     }
03474     #if DEBUG
03475         fprintf (stderr, "window_set_experiment: end\n");
03476     #endif
03477 }
03478
03479 void
03480 window_remove_experiment ()
03481 {
03482     unsigned int i, j;
03483     #if DEBUG
03484         fprintf (stderr, "window_remove_experiment: start\n");
03485     #endif
03486     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03487     g_signal_handler_block (window->combo_experiment, window->
03488         id_experiment);
03489     gtk_combo_box_text_remove (window->combo_experiment, i);
03490     g_signal_handler_unblock (window->combo_experiment, window->
03491         id_experiment);
03492     xmlFree (input->experiment[i]);
03493     --input->nexperiments;
03494     for (j = i; j < input->nexperiments; ++j)
03495     {
03496         input->experiment[j] = input->experiment[j + 1];
03497         input->weight[j] = input->weight[j + 1];
03498     }
03499     j = input->nexperiments - 1;
03500     if (i > j)
03501         i = j;
03502     for (j = 0; j < input->ninputs; ++j)
03503         g_signal_handler_block (window->button_template[j], window->
03504             id_input[j]);

```

```

03509 g_signal_handler_block
03510 (window->button_experiment, window->id_experiment_name);
03511 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03512 g_signal_handler_unblock
03513 (window->button_experiment, window->id_experiment_name);
03514 for (j = 0; j < input->ninputs; ++j)
03515 g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03516 window_update ();
03517 #if DEBUG
03518 fprintf (stderr, "window_remove_experiment: end\n");
03519 #endif
03520 }
03521
03522 void
03523 window_add_experiment ()
03524 {
03525     unsigned int i, j;
03526     #if DEBUG
03527     fprintf (stderr, "window_add_experiment: start\n");
03528     #endif
03529     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03530 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03531 gtk_combo_box_text_insert_text
03532 (window->combo_experiment, i, input->experiment[i]);
03533 g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03534 input->experiment = (char **) g_realloc
03535 (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03536 input->weight = (double *) g_realloc
03537 (input->weight, (input->nexperiments + 1) * sizeof (double));
03538 for (j = input->nexperiments - 1; j > i; --j)
03539 {
03540     input->experiment[j + 1] = input->experiment[j];
03541     input->weight[j + 1] = input->weight[j];
03542 }
03543 input->experiment[j + 1] =
03544 = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03545 input->weight[j + 1] = input->weight[j];
03546 ++input->nexperiments;
03547 for (j = 0; j < input->ninputs; ++j)
03548 g_signal_handler_block (window->button_template[j], window->
id_input[j]);
03549 g_signal_handler_block
03550 (window->button_experiment, window->id_experiment_name);
03551 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03552 g_signal_handler_unblock
03553 (window->button_experiment, window->id_experiment_name);
03554 for (j = 0; j < input->ninputs; ++j)
03555 g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
03556 window_update ();
03557 #if DEBUG
03558 fprintf (stderr, "window_add_experiment: end\n");
03559 #endif
03560 }
03561
03562 void
03563 window_name_experiment ()
03564 {
03565     unsigned int i;
03566     char *buffer;
03567     GFile *file1, *file2;
03568     #if DEBUG
03569     fprintf (stderr, "window_name_experiment: start\n");
03570     #endif
03571     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03572     file1
03573     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03574     file2 = g_file_new_for_path (input->directory);
03575     buffer = g_file_get_relative_path (file2, file1);
03576 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
03577 gtk_combo_box_text_remove (window->combo_experiment, i);
03578 gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03579 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03580 g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
03581 g_free (buffer);
03582 g_object_unref (file2);
03583 g_object_unref (file1);
03584 #if DEBUG
03585 fprintf (stderr, "window_name_experiment: end\n");
03586 #endif
03587 }
03588
03589

```

```

03601 void
03602 window_weight_experiment ()
03603 {
03604     unsigned int i;
03605     #if DEBUG
03606     fprintf (stderr, "window_weight_experiment: start\n");
03607     #endif
03608     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03609     input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03610     #if DEBUG
03611     fprintf (stderr, "window_weight_experiment: end\n");
03612     #endif
03613 }
03614
03620 void
03621 window_inputs_experiment ()
03622 {
03623     unsigned int j;
03624     #if DEBUG
03625     fprintf (stderr, "window_inputs_experiment: start\n");
03626     #endif
03627     j = input->ninputs - 1;
03628     if (j
03629         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03630                                           (window->check_template[j])))
03631         --input->ninputs;
03632     if (input->ninputs < MAX_NINPUTS
03633         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03634                                           (window->check_template[j])))
03635     {
03636         ++input->ninputs;
03637         for (j = 0; j < input->ninputs; ++j)
03638         {
03639             input->template[j] = (char **)
03640                 g_realloc (input->template[j], input->nvariables * sizeof (char *));
03641         }
03642     }
03643     window_update ();
03644     #if DEBUG
03645     fprintf (stderr, "window_inputs_experiment: end\n");
03646     #endif
03647 }
03648
03656 void
03657 window_template_experiment (void *data)
03658 {
03659     unsigned int i, j;
03660     char *buffer;
03661     GFile *file1, *file2;
03662     #if DEBUG
03663     fprintf (stderr, "window_template_experiment: start\n");
03664     #endif
03665     i = (size_t) data;
03666     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03667     file1
03668         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03669     file2 = g_file_new_for_path (input->directory);
03670     buffer = g_file_get_relative_path (file2, file1);
03671     input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03672     g_free (buffer);
03673     g_object_unref (file2);
03674     g_object_unref (file1);
03675     #if DEBUG
03676     fprintf (stderr, "window_template_experiment: end\n");
03677     #endif
03678 }
03679
03684 void
03685 window_set_variable ()
03686 {
03687     unsigned int i;
03688     #if DEBUG
03689     fprintf (stderr, "window_set_variable: start\n");
03690     #endif
03691     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03692     g_signal_handler_block (window->entry_variable, window->
03693                             id_variable_label);
03694     gtk_entry_set_text (window->entry_variable, input->label[i]);
03695     g_signal_handler_unblock (window->entry_variable, window->
03696                               id_variable_label);
03697     gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03698     gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03699     if (input->rangeminabs[i] != -G_MAXDOUBLE)
03700     {
03701         gtk_spin_button_set_value (window->spin_minabs, input->
03702                                     rangeminabs[i]);
03703     }
03704     gtk_toggle_button_set_active

```

```

03701         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03702     }
03703     else
03704     {
03705         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03706         gtk_toggle_button_set_active
03707         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03708     }
03709     if (input->rangemaxabs[i] != G_MAXDOUBLE)
03710     {
03711         gtk_spin_button_set_value (window->spin_maxabs, input->
rangemaxabs[i]);
03712         gtk_toggle_button_set_active
03713         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03714     }
03715     else
03716     {
03717         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03718         gtk_toggle_button_set_active
03719         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03720     }
03721     gtk_spin_button_set_value (window->spin_precision, input->
precision[i]);
03722     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
03723     if (input->nsteps)
03724         gtk_spin_button_set_value (window->spin_step, input->step[i]);
03725     #if DEBUG
03726     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03727             input->precision[i]);
03728     #endif
03729     switch (window_get_algorithm ())
03730     {
03731     case ALGORITHM_SWEEP:
03732         gtk_spin_button_set_value (window->spin_sweeps,
03733                                     (gdouble) input->nsweeps[i]);
03734     #if DEBUG
03735     fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03736             input->nsweeps[i]);
03737     #endif
03738         break;
03739     case ALGORITHM_GENETIC:
03740         gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
nbits[i]);
03741     #if DEBUG
03742     fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03743             input->nbits[i]);
03744     #endif
03745         break;
03746     }
03747     window_update ();
03748     #if DEBUG
03749     fprintf (stderr, "window_set_variable: end\n");
03750     #endif
03751 }
03752
03753 void
03754 window_remove_variable ()
03755 {
03756     unsigned int i, j;
03757     #if DEBUG
03758     fprintf (stderr, "window_remove_variable: start\n");
03759     #endif
03760     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03761     g_signal_handler_block (window->combo_variable, window->
id_variable);
03762     gtk_combo_box_text_remove (window->combo_variable, i);
03763     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03764     xmlFree (input->label[i]);
03765     --input->nvariables;
03766     for (j = i; j < input->nvariables; ++j)
03767     {
03768         input->label[j] = input->label[j + 1];
03769         input->rangemin[j] = input->rangemin[j + 1];
03770         input->rangemax[j] = input->rangemax[j + 1];
03771         input->rangeminabs[j] = input->rangeminabs[j + 1];
03772         input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03773         input->precision[j] = input->precision[j + 1];
03774         input->step[j] = input->step[j + 1];
03775         switch (window_get_algorithm ())
03776         {
03777         case ALGORITHM_SWEEP:
03778             input->nsweeps[j] = input->nsweeps[j + 1];
03779             break;
03780         case ALGORITHM_GENETIC:
03781             input->nbits[j] = input->nbits[j + 1];

```



```

03786     }
03787 }
03788 j = input->nvariables - 1;
03789 if (i > j)
03790     i = j;
03791 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03792 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03793 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03794 window_update ();
03795 #if DEBUG
03796 fprintf (stderr, "window_remove_variable: end\n");
03797 #endif
03798 }
03799
03800 void
03801 window_add_variable ()
03802 {
03803     unsigned int i, j;
03804     #if DEBUG
03805     fprintf (stderr, "window_add_variable: start\n");
03806     #endif
03807     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03808     g_signal_handler_block (window->combo_variable, window->
id_variable_label);
03809     gtk_combo_box_text_insert_text (window->combo_variable, i, input->
label[i]);
03810     g_signal_handler_unblock (window->combo_variable, window->
id_variable_label);
03811     input->label = (char **) g_realloc
(input->label, (input->nvariables + 1) * sizeof (char *));
03812     input->rangemin = (double *) g_realloc
(input->rangemin, (input->nvariables + 1) * sizeof (double));
03813     input->rangemax = (double *) g_realloc
(input->rangemax, (input->nvariables + 1) * sizeof (double));
03814     input->rangeminabs = (double *) g_realloc
(input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03815     input->rangemaxabs = (double *) g_realloc
(input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03816     input->precision = (unsigned int *) g_realloc
(input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03817     input->step = (double *) g_realloc
(input->step, (input->nvariables + 1) * sizeof (double));
03818     for (j = input->nvariables - 1; j > i; --j)
03819     {
03820         input->label[j + 1] = input->label[j];
03821         input->rangemin[j + 1] = input->rangemin[j];
03822         input->rangemax[j + 1] = input->rangemax[j];
03823         input->rangeminabs[j + 1] = input->rangeminabs[j];
03824         input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03825         input->precision[j + 1] = input->precision[j];
03826         input->step[j + 1] = input->step[j];
03827     }
03828     input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03829     input->rangemin[j + 1] = input->rangemin[j];
03830     input->rangemax[j + 1] = input->rangemax[j];
03831     input->rangeminabs[j + 1] = input->rangeminabs[j];
03832     input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03833     input->precision[j + 1] = input->precision[j];
03834     input->step[j + 1] = input->step[j];
03835     switch (window_get_algorithm ())
03836     {
03837     case ALGORITHM_SWEEP:
03838         input->nsweeps = (unsigned int *) g_realloc
(input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03839         for (j = input->nvariables - 1; j > i; --j)
03840             input->nsweeps[j + 1] = input->nsweeps[j];
03841         input->nsweeps[j + 1] = input->nsweeps[j];
03842         break;
03843     case ALGORITHM_GENETIC:
03844         input->nbits = (unsigned int *) g_realloc
(input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03845         for (j = input->nvariables - 1; j > i; --j)
03846             input->nbits[j + 1] = input->nbits[j];
03847         input->nbits[j + 1] = input->nbits[j];
03848     }
03849     ++input->nvariables;
03850     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03851     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03852     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03853     window_update ();
03854     #if DEBUG
03855     fprintf (stderr, "window_add_variable: end\n");
03856     #endif

```

```

03870 }
03871
03876 void
03877 window_label_variable ()
03878 {
03879     unsigned int i;
03880     const char *buffer;
03881     #if DEBUG
03882     fprintf (stderr, "window_label_variable: start\n");
03883     #endif
03884     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03885     buffer = gtk_entry_get_text (window->entry_variable);
03886     g_signal_handler_block (window->combo_variable, window->
03887         id_variable);
03887     gtk_combo_box_text_remove (window->combo_variable, i);
03888     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03889     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03890     g_signal_handler_unblock (window->combo_variable, window->
03891         id_variable);
03891     #if DEBUG
03892     fprintf (stderr, "window_label_variable: end\n");
03893     #endif
03894 }
03895
03900 void
03901 window_precision_variable ()
03902 {
03903     unsigned int i;
03904     #if DEBUG
03905     fprintf (stderr, "window_precision_variable: start\n");
03906     #endif
03907     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03908     input->precision[i]
03909     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03910     gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03911     gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03912     gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03913     gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03914     #if DEBUG
03915     fprintf (stderr, "window_precision_variable: end\n");
03916     #endif
03917 }
03918
03923 void
03924 window_rangemin_variable ()
03925 {
03926     unsigned int i;
03927     #if DEBUG
03928     fprintf (stderr, "window_rangemin_variable: start\n");
03929     #endif
03930     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03931     input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03932     #if DEBUG
03933     fprintf (stderr, "window_rangemin_variable: end\n");
03934     #endif
03935 }
03936
03941 void
03942 window_rangemax_variable ()
03943 {
03944     unsigned int i;
03945     #if DEBUG
03946     fprintf (stderr, "window_rangemax_variable: start\n");
03947     #endif
03948     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03949     input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03950     #if DEBUG
03951     fprintf (stderr, "window_rangemax_variable: end\n");
03952     #endif
03953 }
03954
03959 void
03960 window_rangeminabs_variable ()
03961 {
03962     unsigned int i;
03963     #if DEBUG
03964     fprintf (stderr, "window_rangeminabs_variable: start\n");
03965     #endif
03966     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03967     input->rangeminabs[i] = gtk_spin_button_get_value (window->
03968         spin_minabs);
03968     #if DEBUG
03969     fprintf (stderr, "window_rangeminabs_variable: end\n");
03970     #endif
03971 }
03972
03977 void

```

```

03978 window_rangemaxabs_variable ()
03979 {
03980     unsigned int i;
03981     #if DEBUG
03982         fprintf (stderr, "window_rangemaxabs_variable: start\n");
03983     #endif
03984     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03985     input->rangemaxabs[i] = gtk_spin_button_get_value (window->
        spin_maxabs);
03986     #if DEBUG
03987         fprintf (stderr, "window_rangemaxabs_variable: end\n");
03988     #endif
03989 }
03990
03995 void
03996 window_step_variable ()
03997 {
03998     unsigned int i;
03999     #if DEBUG
04000         fprintf (stderr, "window_step_variable: start\n");
04001     #endif
04002     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04003     input->step[i] = gtk_spin_button_get_value (window->spin_step);
04004     #if DEBUG
04005         fprintf (stderr, "window_step_variable: end\n");
04006     #endif
04007 }
04008
04013 void
04014 window_update_variable ()
04015 {
04016     int i;
04017     #if DEBUG
04018         fprintf (stderr, "window_update_variable: start\n");
04019     #endif
04020     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04021     if (i < 0)
04022         i = 0;
04023     switch (window_get_algorithm ())
04024     {
04025         case ALGORITHM_SWEEP:
04026             input->nsweeps[i]
04027                 = gtk_spin_button_get_value_as_int (window->spin_sweeps);
04028             #if DEBUG
04029                 fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
04030                     input->nsweeps[i]);
04031             #endif
04032             break;
04033         case ALGORITHM_GENETIC:
04034             input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
04035             #if DEBUG
04036                 fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
04037                     input->nbits[i]);
04038             #endif
04039     }
04040     #if DEBUG
04041         fprintf (stderr, "window_update_variable: end\n");
04042     #endif
04043 }
04044
04052 int
04053 window_read (char *filename)
04054 {
04055     unsigned int i;
04056     char *buffer;
04057     #if DEBUG
04058         fprintf (stderr, "window_read: start\n");
04059     #endif
04060
04061     // Reading new input file
04062     input_free ();
04063     if (!input_open (filename))
04064         return 0;
04065
04066     // Setting GTK+ widgets data
04067     gtk_entry_set_text (window->entry_result, input->result);
04068     gtk_entry_set_text (window->entry_variables, input->variables);
04069     buffer = g_build_filename (input->directory, input->simulator, NULL);
04070     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04071         (window->button_simulator), buffer);
04072     g_free (buffer);
04073     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04074         (size_t) input->evaluator);
04075     if (input->evaluator)
04076     {
04077         buffer = g_build_filename (input->directory, input->evaluator, NULL);
04078         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER

```

```

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

```

```

04159 GtkFileChooserDialog *dlg;
04160 GtkFileFilter *filter;
04161 char *buffer, *directory, *name;
04162
04163 #if DEBUG
04164     fprintf (stderr, "window_open: start\n");
04165 #endif
04166
04167 // Saving a backup of the current input file
04168 directory = g_strdup (input->directory);
04169 name = g_strdup (input->name);
04170
04171 // Opening dialog
04172 dlg = (GtkFileChooserDialog *)
04173     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04174                                 window->window,
04175                                 GTK_FILE_CHOOSER_ACTION_OPEN,
04176                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04177                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04178
04179 // Adding XML filter
04180 filter = (GtkFileFilter *) gtk_file_filter_new ();
04181 gtk_file_filter_set_name (filter, "XML");
04182 gtk_file_filter_add_pattern (filter, "*.xml");
04183 gtk_file_filter_add_pattern (filter, "*.XML");
04184 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
04185
04186 // If OK saving
04187 while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04188 {
04189     // Traying to open the input file
04190     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04191     if (!window_read (buffer))
04192     {
04193         #if DEBUG
04194             fprintf (stderr, "window_open: error reading input file\n");
04195         #endif
04196         g_free (buffer);
04197
04198         // Reading backup file on error
04199         buffer = g_build_filename (directory, name, NULL);
04200         if (!input_open (buffer))
04201         {
04202             // Closing on backup file reading error
04203             #if DEBUG
04204                 fprintf (stderr, "window_read: error reading backup file\n");
04205             #endif
04206             g_free (buffer);
04207             break;
04208         }
04209         g_free (buffer);
04210     }
04211     else
04212     {
04213         g_free (buffer);
04214         break;
04215     }
04216 }
04217
04218 // Freeing and closing
04219 g_free (name);
04220 g_free (directory);
04221 gtk_widget_destroy (GTK_WIDGET (dlg));
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] = {

```

```

04250     gettext ("Coordinates descent gradient estimate method"),
04251     gettext ("Random gradient estimate method")
04252 };
04253
04254 #if DEBUG
04255     fprintf (stderr, "window_new: start\n");
04256 #endif
04257
04258 // Creating the window
04259 window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04260
04261 // Finish when closing the window
04262 g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04263
04264 // Setting the window title
04265 gtk_window_set_title (window->window, "MPCOTool");
04266
04267 // Creating the open button
04268 window->button_open = (GtkToolButton *) gtk_tool_button_new
04269     (gtk_image_new_from_icon_name ("document-open",
04270         GTK_ICON_SIZE_LARGE_TOOLBAR),
04271     gettext ("Open"));
04272 g_signal_connect (window->button_open, "clicked", window_open, NULL);
04273
04274 // Creating the save button
04275 window->button_save = (GtkToolButton *) gtk_tool_button_new
04276     (gtk_image_new_from_icon_name ("document-save",
04277         GTK_ICON_SIZE_LARGE_TOOLBAR),
04278     gettext ("Save"));
04279 g_signal_connect (window->button_save, "clicked", (void (*)(
window_save,
NULL));
04280
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
04337 = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04338 window->button_simulator = (GtkFileChooserButton *)
04339   gtk_file_chooser_button_new (gettext ("Simulator program"),
04340   GTK_FILE_CHOOSER_ACTION_OPEN);
04341 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04342   gettext ("Simulator program executable file"));
04343
04344 // Creating the evaluator program label and entry
04345 window->check_evaluator = (GtkCheckButton *)
04346   gtk_check_button_new_with_mnemonic (gettext ("Evaluator program"));
04347 g_signal_connect (window->check_evaluator, "toggled",
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->
label_simulator),
04370   0, 0, 1, 1);
04371 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
04372   1, 0, 1, 1);
04373 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
04374   2, 0, 1, 1);
04375 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
04376   3, 0, 1, 1);
04377 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
04378   0, 1, 1, 1);
04379 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
04380   1, 1, 1, 1);
04381 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
04382   2, 1, 1, 1);
04383 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
04384   3, 1, 1, 1);
04385
04386 // Creating the algorithm properties
04387 window->label_simulations = (GtkLabel *) gtk_label_new
04388   (gettext ("Simulations number"));
04389 window->spin_simulations
04390 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04391 gtk_widget_set_tooltip_text
04392   (GTK_WIDGET (window->spin_simulations),
04393   gettext ("Number of simulations to perform for each iteration"));
04394 window->label_iterations = (GtkLabel *)
04395   gtk_label_new (gettext ("Iterations number"));
04396 window->spin_iterations
04397 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04398 gtk_widget_set_tooltip_text
04399   (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04400 g_signal_connect
04401   (window->spin_iterations, "value-changed", window_update, NULL);
04402 window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04403 window->spin_tolerance
04404 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04405 gtk_widget_set_tooltip_text
04406   (GTK_WIDGET (window->spin_tolerance),
04407   gettext ("Tolerance to set the variable interval on the next iteration"));
04408 window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04409 window->spin_bests
04410 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04411 gtk_widget_set_tooltip_text
04412   (GTK_WIDGET (window->spin_bests),
04413   gettext ("Number of best simulations used to set the variable interval "

```

```

04414         "on the next iteration"));
04415     window->label_population
04416     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04417     window->spin_population
04418     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04419     gtk_widget_set_tooltip_text
04420     (GTK_WIDGET (window->spin_population),
04421      gettext ("Number of population for the genetic algorithm"));
04422     window->label_generations
04423     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04424     window->spin_generations
04425     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04426     gtk_widget_set_tooltip_text
04427     (GTK_WIDGET (window->spin_generations),
04428      gettext ("Number of generations for the genetic algorithm"));
04429     window->label_mutation
04430     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04431     window->spin_mutation
04432     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04433     gtk_widget_set_tooltip_text
04434     (GTK_WIDGET (window->spin_mutation),
04435      gettext ("Ratio of mutation for the genetic algorithm"));
04436     window->label_reproduction
04437     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04438     window->spin_reproduction
04439     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04440     gtk_widget_set_tooltip_text
04441     (GTK_WIDGET (window->spin_reproduction),
04442      gettext ("Ratio of reproduction for the genetic algorithm"));
04443     window->label_adaptation
04444     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04445     window->spin_adaptation
04446     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04447     gtk_widget_set_tooltip_text
04448     (GTK_WIDGET (window->spin_adaptation),
04449      gettext ("Ratio of adaptation for the genetic algorithm"));
04450
04451     // Creating the gradient based method properties
04452     window->check_gradient = (GtkCheckButton *)
04453     gtk_check_button_new_with_mnemonic (gettext ("Gradient based method"));
04454     g_signal_connect (window->check_gradient, "clicked",
04455     window_update, NULL);
04456     window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04457     window->button_gradient[0] = (GtkRadioButton *)
04458     gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04459     gtk_grid_attach (window->grid_gradient,
04460     GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04461     g_signal_connect (window->button_gradient[0], "clicked",
04462     window_update, NULL);
04463     for (i = 0; ++i < NGRADIENTS;)
04464     {
04465         window->button_gradient[i] = (GtkRadioButton *)
04466         gtk_radio_button_new_with_mnemonic
04467         (gtk_radio_button_get_group (window->button_gradient[0]),
04468          label_gradient[i]);
04469         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04470          tip_gradient[i]);
04471         gtk_grid_attach (window->grid_gradient,
04472          GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04473         g_signal_connect (window->button_gradient[i], "clicked",
04474          window_update, NULL);
04475     }
04476     window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04477     window->spin_steps = (GtkSpinButton *)
04478     gtk_spin_button_new_with_range (1., 1.e12, 1.);
04479     window->label_estimates
04480     = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04481     window->spin_estimates = (GtkSpinButton *)
04482     gtk_spin_button_new_with_range (1., 1.e3, 1.);
04483     window->label_relaxation
04484     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04485     window->spin_relaxation = (GtkSpinButton *)
04486     gtk_spin_button_new_with_range (0., 2., 0.001);
04487     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04488     label_steps),
04489     0, NGRADIENTS, 1, 1);
04490     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04491     spin_steps),
04492     1, NGRADIENTS, 1, 1);
04493     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04494     label_estimates),
04495     0, NGRADIENTS + 1, 1, 1);
04496     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04497     spin_estimates),
04498     1, NGRADIENTS + 1, 1, 1);
04499     gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
04500     label_relaxation),

```



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

```

04580         GTK_WIDGET (window->grid_algorithm));
04581
04582     // Creating the variable widgets
04583     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04584     gtk_widget_set_tooltip_text
04585         (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04586     window->id_variable = g_signal_connect
04587         (window->combo_variable, "changed", window_set_variable, NULL);
04588     window->button_add_variable
04589         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04590             GTK_ICON_SIZE_BUTTON);
04591     g_signal_connect
04592         (window->button_add_variable, "clicked",
04593         window_add_variable, NULL);
04594     gtk_widget_set_tooltip_text
04595         (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
04596     window->button_remove_variable
04597         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04598             GTK_ICON_SIZE_BUTTON);
04599     g_signal_connect
04600         (window->button_remove_variable, "clicked",
04601         window_remove_variable, NULL);
04602     gtk_widget_set_tooltip_text
04603         (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04604     window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04605     window->entry_variable = (GtkEntry *) gtk_entry_new ();
04606     gtk_widget_set_tooltip_text
04607         (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04608     window->id_variable_label = g_signal_connect
04609         (window->entry_variable, "changed", window_label_variable, NULL);
04610     window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04611     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04612         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04613     gtk_widget_set_tooltip_text
04614         (GTK_WIDGET (window->spin_min),
04615         gettext ("Minimum initial value of the variable"));
04616     window->scrolled_min
04617         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04618     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04619         GTK_WIDGET (window->spin_min));
04620     g_signal_connect (window->spin_min, "value-changed",
04621         window_rangemin_variable, NULL);
04622     window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04623     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04624         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04625     gtk_widget_set_tooltip_text
04626         (GTK_WIDGET (window->spin_max),
04627         gettext ("Maximum initial value of the variable"));
04628     window->scrolled_max
04629         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04630     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04631         GTK_WIDGET (window->spin_max));
04632     g_signal_connect (window->spin_max, "value-changed",
04633         window_rangemax_variable, NULL);
04634     window->check_minabs = (GtkCheckButton *)
04635         gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
04636     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04637     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04638         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04639     gtk_widget_set_tooltip_text
04640         (GTK_WIDGET (window->spin_minabs),
04641         gettext ("Minimum allowed value of the variable"));
04642     window->scrolled_minabs
04643         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04644     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04645         GTK_WIDGET (window->spin_minabs));
04646     g_signal_connect (window->spin_minabs, "value-changed",
04647         window_rangeminabs_variable, NULL);
04648     window->check_maxabs = (GtkCheckButton *)
04649         gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
04650     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04651     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04652         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04653     gtk_widget_set_tooltip_text
04654         (GTK_WIDGET (window->spin_maxabs),
04655         gettext ("Maximum allowed value of the variable"));
04656     window->scrolled_maxabs
04657         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04658     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04659         GTK_WIDGET (window->spin_maxabs));
04660     g_signal_connect (window->spin_maxabs, "value-changed",
04661         window_rangemaxabs_variable, NULL);
04662     window->label_precision
04663         = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04664     window->spin_precision = (GtkSpinButton *)
04665         gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04666     gtk_widget_set_tooltip_text

```

```

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

```

```

04751 window->button_add_experiment
04752     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04753     GTK_ICON_SIZE_BUTTON);
04754 g_signal_connect
04755     (window->button_add_experiment, "clicked",
04756 window_add_experiment, NULL);
04757 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04758     gettext ("Add experiment"));
04759 window->button_remove_experiment
04760     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04761     GTK_ICON_SIZE_BUTTON);
04762 g_signal_connect (window->button_remove_experiment, "clicked",
04763     window_remove_experiment, NULL);
04764 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04765     gettext ("Remove experiment"));
04766 window->label_experiment
04767     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04768 window->button_experiment = (GtkFileChooserButton *)
04769     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04770     GTK_FILE_CHOOSER_ACTION_OPEN);
04771 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04772     gettext ("Experimental data file"));
04773 window->id_experiment_name
04774     = g_signal_connect (window->button_experiment, "selection-changed",
04775     window_name_experiment, NULL);
04776 window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
04777 window->spin_weight
04778     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04779 gtk_widget_set_tooltip_text
04780     (GTK_WIDGET (window->spin_weight),
04781     gettext ("Weight factor to build the objective function"));
04782 g_signal_connect
04783     (window->spin_weight, "value-changed", window_weight_experiment,
04784     NULL);
04785 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04786 gtk_grid_attach (window->grid_experiment,
04787     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
04788 gtk_grid_attach (window->grid_experiment,
04789     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
04790 gtk_grid_attach (window->grid_experiment,
04791     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
04792 gtk_grid_attach (window->grid_experiment,
04793     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
04794 gtk_grid_attach (window->grid_experiment,
04795     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
04796 gtk_grid_attach (window->grid_experiment,
04797     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04798 gtk_grid_attach (window->grid_experiment,
04799     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04800 for (i = 0; i < MAX_NINPUTS; ++i)
04801 {
04802     snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04803     window->check_template[i] = (GtkCheckButton *)
04804     gtk_check_button_new_with_label (buffer3);
04805     window->id_template[i]
04806     = g_signal_connect (window->check_template[i], "toggled",
04807     window_inputs_experiment, NULL);
04808     gtk_grid_attach (window->grid_experiment,
04809     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
04810     window->button_template[i] = (GtkFileChooserButton *)
04811     gtk_file_chooser_button_new (gettext ("Input template"),
04812     GTK_FILE_CHOOSER_ACTION_OPEN);
04813     gtk_widget_set_tooltip_text
04814     (GTK_WIDGET (window->button_template[i]),
04815     gettext ("Experimental input template file"));
04816     window->id_input[i]
04817     = g_signal_connect_swapped (window->button_template[i],
04818     "selection-changed",
04819     (void *) window_template_experiment,
04820     (void *) (size_t) i);
04821     gtk_grid_attach (window->grid_experiment,
04822     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
04823 }
04824 window->frame_experiment
04825     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04826 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04827     GTK_WIDGET (window->grid_experiment));
04828 // Creating the grid and attaching the widgets to the grid
04829 window->grid = (GtkGrid *) gtk_grid_new ();
04830 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04831 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04832 gtk_grid_attach (window->grid,
04833     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04834 gtk_grid_attach (window->grid,
04835     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04836 gtk_grid_attach (window->grid,

```

```

04836         GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04837     gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));
04838
04839     // Setting the window logo
04840     window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04841     gtk_window_set_icon (window->window, window->logo);
04842
04843     // Showing the window
04844     gtk_widget_show_all (GTK_WIDGET (window->window));
04845
04846     // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04847     #if GTK_MINOR_VERSION >= 16
04848     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04849     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04850     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04851     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04852     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
04853     #endif
04854
04855     // Reading initial example
04856     input_new ();
04857     buffer2 = g_get_current_dir ();
04858     buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04859     g_free (buffer2);
04860     window_read (buffer);
04861     g_free (buffer);
04862
04863     #if DEBUG
04864     fprintf (stderr, "window_new: start\n");
04865     #endif
04866 }
04867
04868 #endif
04869
04870 int
04871 cores_number ()
04872 {
04873     #ifdef G_OS_WIN32
04874     SYSTEM_INFO sysinfo;
04875     GetSystemInfo (&sysinfo);
04876     return sysinfo.dwNumberOfProcessors;
04877     #else
04878     return (int) sysconf (_SC_NPROCESSORS_ONLN);
04879     #endif
04880 }
04881
04882 int
04883 main (int argn, char **argc)
04884 {
04885     #if HAVE_GTK
04886     char *buffer;
04887     #endif
04888
04889     // Starting pseudo-random numbers generator
04890     calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04891     calibrate->seed = DEFAULT_RANDOM_SEED;
04892
04893     // Allowing spaces in the XML data file
04894     xmlKeepBlanksDefault (0);
04895
04896     // Starting MPI
04897     #if HAVE_MPI
04898     MPI_Init (&argn, &argc);
04899     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04900     MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04901     printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04902     #else
04903     ntasks = 1;
04904     #endif
04905
04906     #if HAVE_GTK
04907     // Getting threads number
04908     nthreads_gradient = nthreads = cores_number ();
04909
04910     // Setting local language and international floating point numbers notation
04911     setlocale (LC_ALL, "");
04912     setlocale (LC_NUMERIC, "C");
04913     window->application_directory = g_get_current_dir ();
04914     buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
04915     bindtextdomain (PROGRAM_INTERFACE, buffer);
04916     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04917     textdomain (PROGRAM_INTERFACE);
04918
04919     // Initing GTK+

```

```

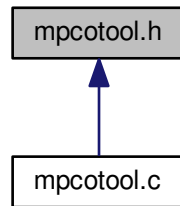
04935  gtk_disable_setlocale ();
04936  gtk_init (&argn, &argc);
04937
04938  // Opening the main window
04939  window_new ();
04940  gtk_main ();
04941
04942  // Freeing memory
04943  input_free ();
04944  g_free (buffer);
04945  gtk_widget_destroy (GTK_WIDGET (window->window));
04946  g_free (window->application_directory);
04947
04948  #else
04949
04950  // Checking syntax
04951  if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04952  {
04953      printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04954      return 1;
04955  }
04956
04957  // Getting threads number
04958  if (argn == 2)
04959      nthreads_gradient = nthreads = cores_number ();
04960  else
04961  {
04962      nthreads_gradient = nthreads = atoi (argc[2]);
04963      if (!nthreads)
04964      {
04965          printf ("Bad threads number\n");
04966          return 2;
04967      }
04968  }
04969  printf ("nthreads=%u\n", nthreads);
04970
04971  // Making calibration
04972  if (input_open (argc[argn - 1]))
04973      calibrate_open ();
04974
04975  // Freeing memory
04976  calibrate_free ();
04977
04978  #endif
04979
04980  // Closing MPI
04981  #if HAVE_MPI
04982      MPI_Finalize ();
04983  #endif
04984
04985  // Freeing memory
04986  gsl_rng_free (calibrate->rng);
04987
04988  // Closing
04989  return 0;
04990 }

```

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 1444 of file [mpcotool.c](#).

```

01445 {
01446     unsigned int i, j;
01447     double e;
01448     #if DEBUG
01449         fprintf (stderr, "calibrate_best: start\n");
01450         fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01451                 calibrate->nsaveds, calibrate->nbest);
01452     #endif
01453     if (calibrate->nsaveds < calibrate->nbest
01454         || value < calibrate->error_best[calibrate->nsaveds - 1])
01455     {
01456         if (calibrate->nsaveds < calibrate->nbest)
01457             ++calibrate->nsaveds;
01458         calibrate->error_best[calibrate->nsaveds - 1] = value;
01459         calibrate->simulation_best[calibrate->
01460 nsaveds - 1] = simulation;
01461         for (i = calibrate->nsaveds; --i;)
01462         {
01463             if (calibrate->error_best[i] < calibrate->
01464 error_best[i - 1])
01465             {
01466                 j = calibrate->simulation_best[i];
01467                 e = calibrate->error_best[i];
01468                 calibrate->simulation_best[i] = calibrate->
01469 simulation_best[i - 1];
01470                 calibrate->error_best[i] = calibrate->
01471 error_best[i - 1];
01472                 calibrate->simulation_best[i - 1] = j;
01473                 calibrate->error_best[i - 1] = e;
01474             }
01475             else
01476                 break;
01477         }
01478     }
01479     #if DEBUG
01480         fprintf (stderr, "calibrate_best: end\n");
01481     #endif
01482 }

```

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 1757 of file [mpcotool.c](#).

```

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

```

```

01771             "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01772             simulation, value);
01773 #endif
01774     }
01775 #if DEBUG
01776     fprintf (stderr, "calibrate_best_gradient: end\n");
01777 #endif
01778 }

```

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.

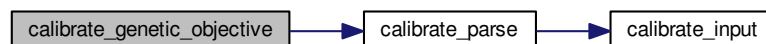
Definition at line 2060 of file [mpcotool.c](#).

```

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

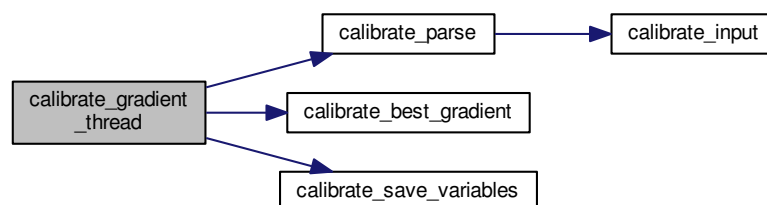
Definition at line 1822 of file [mpcotool.c](#).

```

01823 {
01824     unsigned int i, j, thread;
01825     double e;
01826     #if DEBUG
01827         fprintf (stderr, "calibrate_gradient_thread: start\n");
01828     #endif
01829     thread = data->thread;
01830     #if DEBUG
01831         fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01832                 thread,
01833                 calibrate->thread_gradient[thread],
01834                 calibrate->thread_gradient[thread + 1]);
01835     #endif
01836     for (i = calibrate->thread_gradient[thread];
01837          i < calibrate->thread_gradient[thread + 1]; ++i)
01838     {
01839         e = 0.;
01840         for (j = 0; j < calibrate->nexperiments; ++j)
01841             e += calibrate_parse (i, j);
01842         g_mutex_lock (mutex);
01843         calibrate_best_gradient (i, e);
01844         calibrate_save_variables (i, e);
01845         g_mutex_unlock (mutex);
01846     #if DEBUG
01847         fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01848     #endif
01849     }
01850     #if DEBUG
01851         fprintf (stderr, "calibrate_gradient_thread: end\n");
01852     #endif
01853     g_thread_exit (NULL);
01854     return NULL;
01855 }

```

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 1197 of file [mpcotool.c](#).

```

01198 {
01199     unsigned int i;
01200     char buffer[32], value[32], *buffer2, *buffer3, *content;
01201     FILE *file;
01202     gsize length;
01203     GRegex *regex;
01204
01205     #if DEBUG
01206         fprintf (stderr, "calibrate_input: start\n");
01207     #endif
01208
01209     // Checking the file
01210     if (!template)
01211         goto calibrate_input_end;
01212
01213     // Opening template
01214     content = g_mapped_file_get_contents (template);
01215     length = g_mapped_file_get_length (template);
01216     #if DEBUG
01217         fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01218                 content);
01219     #endif
01220     file = g_fopen (input, "w");
01221
01222     // Parsing template
01223     for (i = 0; i < calibrate->nvariables; ++i)
01224     {
01225         #if DEBUG
01226             fprintf (stderr, "calibrate_input: variable=%u\n", i);
01227         #endif
01228         snprintf (buffer, 32, "@variable%u@", i + 1);
01229         regex = g_regex_new (buffer, 0, 0, NULL);
01230         if (i == 0)
01231         {
01232             buffer2 = g_regex_replace_literal (regex, content, length, 0,
01233                                                calibrate->label[i], 0, NULL);
01234             #if DEBUG
01235                 fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01236             #endif
01237         }
01238         else
01239         {
01240             length = strlen (buffer3);
01241             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01242                                                calibrate->label[i], 0, NULL);
01243             g_free (buffer3);
01244         }
01245         g_regex_unref (regex);
01246         length = strlen (buffer2);
01247         snprintf (buffer, 32, "@value%u@", i + 1);
01248         regex = g_regex_new (buffer, 0, 0, NULL);
01249         snprintf (value, 32, format[calibrate->precision[i]],
01250                  calibrate->value[simulation * calibrate->
01251                                nvariables + i]);
01252         #if DEBUG
01253             fprintf (stderr, "calibrate_input: value=%s\n", value);
01254         #endif
01255         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01256                                           0, NULL);
01257         g_free (buffer2);
01258         g_regex_unref (regex);
01259     }
01260
01261     // Saving input file
01262     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01263     g_free (buffer3);
01264     fclose (file);
01265
01266     calibrate_input_end:
01267     #if DEBUG
01268         fprintf (stderr, "calibrate_input: end\n");
01269     #endif
01270     return;
01271 }

```

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 1562 of file `mpcotool.c`.

```

01564 {
01565     unsigned int i, j, k, s[calibrate->nbest];
01566     double e[calibrate->nbest];
01567     #if DEBUG
01568     fprintf (stderr, "calibrate_merge: start\n");
01569     #endif
01570     i = j = k = 0;
01571     do
01572     {
01573         if (i == calibrate->nsaveds)
01574         {
01575             s[k] = simulation_best[j];
01576             e[k] = error_best[j];
01577             ++j;
01578             ++k;
01579             if (j == nsaveds)
01580                 break;
01581         }
01582         else if (j == nsaveds)
01583         {
01584             s[k] = calibrate->simulation_best[i];
01585             e[k] = calibrate->error_best[i];
01586             ++i;
01587             ++k;
01588             if (i == calibrate->nsaveds)
01589                 break;
01590         }
01591         else if (calibrate->error_best[i] > error_best[j])
01592         {
01593             s[k] = simulation_best[j];
01594             e[k] = error_best[j];
01595             ++j;
01596             ++k;
01597         }
01598         else
01599         {
01600             s[k] = calibrate->simulation_best[i];
01601             e[k] = calibrate->error_best[i];
01602             ++i;
01603             ++k;
01604         }
01605     }
01606     while (k < calibrate->nbest);
01607     calibrate->nsaveds = k;
01608     memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01609     memcpy (calibrate->error_best, e, k * sizeof (double));
01610     #if DEBUG
01611     fprintf (stderr, "calibrate_merge: end\n");
01612     #endif
01613 }

```

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

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

Parameters

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

Returns

Objective function value.

Definition at line 1284 of file [mpcotool.c](#).

```

01285 {
01286     unsigned int i;
01287     double e;
01288     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01289         *buffer3, *buffer4;
01290     FILE *file_result;
01291
01292     #if DEBUG
01293         fprintf (stderr, "calibrate_parse: start\n");
01294         fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01295             experiment);
01296     #endif
01297
01298     // Opening input files
01299     for (i = 0; i < calibrate->ninputs; ++i)
01300     {
01301         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01302     #if DEBUG
01303         fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01304     #endif
01305         calibrate_input (simulation, &input[i][0],
01306             calibrate->file[i][experiment]);
01307     }
01308     for (; i < MAX_NINPUTS; ++i)
01309         strcpy (&input[i][0], "");
01310     #if DEBUG
01311         fprintf (stderr, "calibrate_parse: parsing end\n");
01312     #endif
01313
01314     // Performing the simulation
01315     snprintf (output, 32, "output-%u-%u", simulation, experiment);
01316     buffer2 = g_path_get_dirname (calibrate->simulator);
01317     buffer3 = g_path_get_basename (calibrate->simulator);
01318     buffer4 = g_build_filename (buffer2, buffer3, NULL);
01319     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
01320         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01321         input[6], input[7], output);
01322     g_free (buffer4);
01323     g_free (buffer3);
01324     g_free (buffer2);
01325     #if DEBUG
01326         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01327     #endif
01328     system (buffer);
01329
01330     // Checking the objective value function
01331     if (calibrate->evaluator)
01332     {
01333         snprintf (result, 32, "result-%u-%u", simulation, experiment);
01334         buffer2 = g_path_get_dirname (calibrate->evaluator);
01335         buffer3 = g_path_get_basename (calibrate->evaluator);
01336         buffer4 = g_build_filename (buffer2, buffer3, NULL);
01337         snprintf (buffer, 512, "%s\ " %s %s %s",
01338             buffer4, output, calibrate->experiment[experiment], result);
01339         g_free (buffer4);
01340         g_free (buffer3);
01341         g_free (buffer2);
01342     #if DEBUG
01343         fprintf (stderr, "calibrate_parse: %s\n", buffer);
01344     #endif
01345         system (buffer);
01346         file_result = g_fopen (result, "r");
01347         e = atof (fgets (buffer, 512, file_result));
01348         fclose (file_result);
01349     }
01350     else
01351     {
01352         strcpy (result, "");
01353         file_result = g_fopen (output, "r");
01354         e = atof (fgets (buffer, 512, file_result));
01355         fclose (file_result);
01356     }
01357
01358     // Removing files
01359     #if !DEBUG
01360     for (i = 0; i < calibrate->ninputs; ++i)
01361     {
01362         if (calibrate->file[i][0])
01363         {

```

```

01364         snprintf (buffer, 512, RM " %s", &input[i][0]);
01365         system (buffer);
01366     }
01367 }
01368     snprintf (buffer, 512, RM " %s %s", output, result);
01369     system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373     fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376 // Returning the objective function
01377 return e * calibrate->weight[experiment];
01378 }

```

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 1416 of file [mpcotool.c](#).

```

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

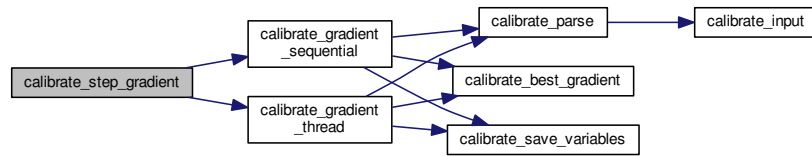
Definition at line 1924 of file `mpcotool.c`.

```

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

```

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

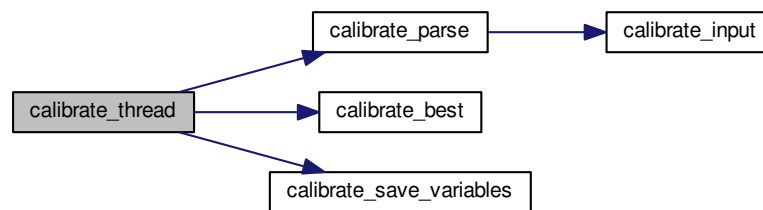
Definition at line 1518 of file [mpcotool.c](#).

```

01519 {
01520     unsigned int i, j, thread;
01521     double e;
01522     #if DEBUG
01523         fprintf (stderr, "calibrate_thread: start\n");
01524     #endif
01525     thread = data->thread;
01526     #if DEBUG
01527         fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01528                 calibrate->thread[thread], calibrate->thread[thread + 1]);
01529     #endif
01530     for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01531     {
01532         e = 0.;
01533         for (j = 0; j < calibrate->nexperiments; ++j)
01534             e += calibrate_parse (i, j);
01535         g_mutex_lock (mutex);
01536         calibrate_best (i, e);
01537         calibrate_save_variables (i, e);
01538         g_mutex_unlock (mutex);
01539     #if DEBUG
01540         fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01541     #endif
01542     }
01543     #if DEBUG
01544         fprintf (stderr, "calibrate_thread: end\n");
01545     #endif
01546     g_thread_exit (NULL);
01547     return NULL;
01548 }

```

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 549 of file `mpcotool.c`.

```

00550 {
00551     char buffer2[64];
00552     char *buffert[MAX_NINPUTS] =
00553         { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00554     xmlDoc *doc;
00555     xmlNode *node, *child;
00556     xmlChar *buffer;
00557     char *msg;
00558     int error_code;
00559     unsigned int i;
00560
00561     #if DEBUG
00562     fprintf (stderr, "input_open: start\n");
00563     #endif
00564
00565     // Resetting input data
00566     buffer = NULL;
00567     input_new ();
00568
00569     // Parsing the input file
00570     #if DEBUG
00571     fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00572     #endif
00573     doc = xmlParseFile (filename);
00574     if (!doc)
00575     {
00576         msg = gettext ("Unable to parse the input file");
00577         goto exit_on_error;
00578     }
00579
00580     // Getting the root node
00581     #if DEBUG
00582     fprintf (stderr, "input_open: getting the root node\n");
00583     #endif
00584     node = xmlDocGetRootElement (doc);
00585     if (xmlStrcmp (node->name, XML_CALIBRATE))
00586     {
00587         msg = gettext ("Bad root XML node");
00588         goto exit_on_error;
00589     }
00590

```

```

00591 // Getting results file names
00592 input->result = (char *) xmlGetProp (node, XML_RESULT);
00593 if (!input->result)
00594     input->result = (char *) xmlStrdup (result_name);
00595 input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00596 if (!input->variables)
00597     input->variables = (char *) xmlStrdup (variables_name);
00598
00599 // Opening simulator program name
00600 input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00601 if (!input->simulator)
00602 {
00603     msg = gettext ("Bad simulator program");
00604     goto exit_on_error;
00605 }
00606
00607 // Opening evaluator program name
00608 input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00609
00610 // Obtaining pseudo-random numbers generator seed
00611 input->seed
00612     = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
00613                                     &error_code);
00614 if (error_code)
00615 {
00616     msg = gettext ("Bad pseudo-random numbers generator seed");
00617     goto exit_on_error;
00618 }
00619
00620 // Opening algorithm
00621 buffer = xmlGetProp (node, XML_ALGORITHM);
00622 if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00623 {
00624     input->algorithm = ALGORITHM_MONTE_CARLO;
00625
00626     // Obtaining simulations number
00627     input->nsimulations
00628         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00629     if (error_code)
00630     {
00631         msg = gettext ("Bad simulations number");
00632         goto exit_on_error;
00633     }
00634 }
00635 else if (!xmlStrcmp (buffer, XML_SWEEP))
00636     input->algorithm = ALGORITHM_SWEEP;
00637 else if (!xmlStrcmp (buffer, XML_GENETIC))
00638 {
00639     input->algorithm = ALGORITHM_GENETIC;
00640
00641     // Obtaining population
00642     if (xmlHasProp (node, XML_NPOPULATION))
00643     {
00644         input->nsimulations
00645             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00646         if (error_code || input->nsimulations < 3)
00647         {
00648             msg = gettext ("Invalid population number");
00649             goto exit_on_error;
00650         }
00651     }
00652     else
00653     {
00654         msg = gettext ("No population number");
00655         goto exit_on_error;
00656     }
00657
00658     // Obtaining generations
00659     if (xmlHasProp (node, XML_NGENERATIONS))
00660     {
00661         input->niterations
00662             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00663         if (error_code || !input->niterations)
00664         {
00665             msg = gettext ("Invalid generations number");
00666             goto exit_on_error;
00667         }
00668     }
00669     else
00670     {
00671         msg = gettext ("No generations number");
00672         goto exit_on_error;
00673     }
00674
00675     // Obtaining mutation probability
00676     if (xmlHasProp (node, XML_MUTATION))

```

```

00677     {
00678         input->mutation_ratio
00679         = xml_node_get_float (node, XML_MUTATION, &error_code);
00680         if (error_code || input->mutation_ratio < 0.
00681             || input->mutation_ratio >= 1.)
00682         {
00683             msg = gettext ("Invalid mutation probability");
00684             goto exit_on_error;
00685         }
00686     }
00687     else
00688     {
00689         msg = gettext ("No mutation probability");
00690         goto exit_on_error;
00691     }
00692
00693     // Obtaining reproduction probability
00694     if (xmlHasProp (node, XML_REPRODUCTION))
00695     {
00696         input->reproduction_ratio
00697         = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00698         if (error_code || input->reproduction_ratio < 0.
00699             || input->reproduction_ratio >= 1.0)
00700         {
00701             msg = gettext ("Invalid reproduction probability");
00702             goto exit_on_error;
00703         }
00704     }
00705     else
00706     {
00707         msg = gettext ("No reproduction probability");
00708         goto exit_on_error;
00709     }
00710
00711     // Obtaining adaptation probability
00712     if (xmlHasProp (node, XML_ADAPTATION))
00713     {
00714         input->adaptation_ratio
00715         = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00716         if (error_code || input->adaptation_ratio < 0.
00717             || input->adaptation_ratio >= 1.)
00718         {
00719             msg = gettext ("Invalid adaptation probability");
00720             goto exit_on_error;
00721         }
00722     }
00723     else
00724     {
00725         msg = gettext ("No adaptation probability");
00726         goto exit_on_error;
00727     }
00728
00729     // Checking survivals
00730     i = input->mutation_ratio * input->nsimulations;
00731     i += input->reproduction_ratio * input->
00732 nsimulations;
00733     i += input->adaptation_ratio * input->
00734 nsimulations;
00735     if (i > input->nsimulations - 2)
00736     {
00737         msg = gettext
00738             ("No enough survival entities to reproduce the population");
00739         goto exit_on_error;
00740     }
00741     else
00742     {
00743         msg = gettext ("Unknown algorithm");
00744         goto exit_on_error;
00745     }
00746     xmlFree (buffer);
00747     buffer = NULL;
00748
00749     if (input->algorithm == ALGORITHM_MONTE_CARLO
00750         || input->algorithm == ALGORITHM_SWEEP)
00751     {
00752         // Obtaining iterations number
00753         input->niterations
00754         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00755         if (error_code == 1)
00756             input->niterations = 1;
00757         else if (error_code)
00758         {
00759             msg = gettext ("Bad iterations number");
00760             goto exit_on_error;
00761         }
00762     }

```

```

00762
00763     // Obtaining best number
00764     input->nbest
00765     = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
00766     if (error_code || !input->nbest)
00767     {
00768         msg = gettext ("Invalid best number");
00769         goto exit_on_error;
00770     }
00771
00772     // Obtaining tolerance
00773     input->tolerance
00774     = xml_node_get_float_with_default (node,
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,
XML_NSTEPS, &error_code);
00786         if (error_code || !input->nsteps)
00787         {
00788             msg = gettext ("Invalid steps number");
00789             goto exit_on_error;
00790         }
00791         buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00792         if (!xmlStrcmp (buffer, XML_COORDINATES))
00793             input->gradient_method =
GRADIENT_METHOD_COORDINATES;
00794         else if (!xmlStrcmp (buffer, XML_RANDOM))
00795         {
00796             input->gradient_method =
GRADIENT_METHOD_RANDOM;
00797             input->nestimates
00798             = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00799             if (error_code || !input->nestimates)
00800             {
00801                 msg = gettext ("Invalid estimates number");
00802                 goto exit_on_error;
00803             }
00804             }
00805         else
00806         {
00807             msg = gettext ("Unknown method to estimate the gradient");
00808             goto exit_on_error;
00809         }
00810         xmlFree (buffer);
00811         buffer = NULL;
00812         input->relaxation
00813         = xml_node_get_float_with_default (node,
XML_RELAXATION,
00814                                         DEFAULT_RELAXATION, &error_code);
00815         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00816         {
00817             msg = gettext ("Invalid relaxation parameter");
00818             goto exit_on_error;
00819         }
00820     }
00821     else
00822         input->nsteps = 0;
00823 }
00824
00825 // Reading the experimental data
00826 for (child = node->children; child; child = child->next)
00827 {
00828     if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831     fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833     if (xmlHasProp (child, XML_NAME))
00834         buffer = xmlGetProp (child, XML_NAME);
00835     else
00836     {
00837         snprintf (buffer2, 64, "%s %u: %s",
00838                 gettext ("Experiment"),
00839                 input->nexperiments + 1, gettext ("no data file name"));
00840         msg = buffer2;
00841         goto exit_on_error;

```

```

00842     }
00843     #if DEBUG
00844     fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845     #endif
00846     input->weight = g_realloc (input->weight,
00847                               (1 + input->nexperiments) * sizeof (double));
00848     input->weight[input->nexperiments]
00849     = xml_node_get_float_with_default (child,
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 templatel=%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"));
00901             msg = buffer2;
00902             while (i-- > 0)
00903                 xmlFree (buffert[i]);
00904             goto exit_on_error;
00905         }
00906         input->template[i] = (char **)
00907             g_realloc (input->template[i],
00908                       (1 + input->nexperiments) * sizeof (char *));
00909         buffert[i] = (char *) xmlGetProp (child, template[i]);
00910     #if DEBUG
00911     fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00912             input->nexperiments, i + 1,
00913             input->template[i][input->nexperiments]);
00914     #endif
00915     if (!input->nexperiments)
00916         ++input->ninputs;
00917     #if DEBUG
00918     fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00919     #endif
00920     }
00921     else if (input->nexperiments && input->ninputs >= i)
00922     {
00923         snprintf (buffer2, 64, "%s %s: %s%u",
00924                 gettext ("Experiment"),
00925                 buffer, gettext ("no template"), i + 1);
00926         msg = buffer2;
00927         while (i-- > 0)

```

```

00928         xmlFree (buffert[i]);
00929         goto exit_on_error;
00930     }
00931     else
00932         break;
00933 }
00934 input->experiment
00935 = g_realloc (input->experiment,
00936             (1 + input->nexperiments) * sizeof (char *));
00937 input->experiment[input->nexperiments] = (char *) buffer;
00938 for (i = 0; i < input->ninputs; ++i)
00939     input->template[i][input->nexperiments] = buffert[i];
00940 ++input->nexperiments;
00941 #if DEBUG
00942 fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00943 #endif
00944 }
00945 if (!input->nexperiments)
00946 {
00947     msg = gettext ("No calibration experiments");
00948     goto exit_on_error;
00949 }
00950 buffer = NULL;
00951
00952 // Reading the variables data
00953 for (; child; child = child->next)
00954 {
00955     if (xmlStrcmp (child->name, XML_VARIABLE))
00956     {
00957         snprintf (buffer2, 64, "%s %u: %s",
00958                 gettext ("Variable"),
00959                 input->nvariables + 1, gettext ("bad XML node"));
00960         msg = buffer2;
00961         goto exit_on_error;
00962     }
00963     if (xmlHasProp (child, XML_NAME))
00964         buffer = xmlGetProp (child, XML_NAME);
00965     else
00966     {
00967         snprintf (buffer2, 64, "%s %u: %s",
00968                 gettext ("Variable"),
00969                 input->nvariables + 1, gettext ("no name"));
00970         msg = buffer2;
00971         goto exit_on_error;
00972     }
00973     if (xmlHasProp (child, XML_MINIMUM))
00974     {
00975         input->rangemin = g_realloc
00976             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00977         input->rangeminabs = g_realloc
00978             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00979         input->rangemin[input->nvariables]
00980             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00981         if (error_code)
00982         {
00983             snprintf (buffer2, 64, "%s %s: %s",
00984                     gettext ("Variable"), buffer, gettext ("bad minimum"));
00985             msg = buffer2;
00986             goto exit_on_error;
00987         }
00988         input->rangeminabs[input->nvariables]
00989             = xml_node_get_float_with_default (child,
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;

```



```

01014     }
01015     if (xmlHasProp (child, XML_MAXIMUM))
01016     {
01017         input->rangemax = g_realloc
01018             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01019         input->rangemaxabs = g_realloc
01020             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01021         input->rangemax[input->nvariables]
01022             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01023         if (error_code)
01024         {
01025             snprintf (buffer2, 64, "%s %s: %s",
01026                 gettext ("Variable"), buffer, gettext ("bad maximum"));
01027             msg = buffer2;
01028             goto exit_on_error;
01029         }
01030         input->rangemaxabs[input->nvariables]
01031             = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
01032                                     G_MAXDOUBLE, &error_code);
01033         if (error_code)
01034         {
01035             snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01036                 gettext ("bad absolute maximum"));
01037             msg = buffer2;
01038             goto exit_on_error;
01039         }
01040         if (input->rangemax[input->nvariables]
01041             > input->rangemaxabs[input->nvariables])
01042         {
01043             snprintf (buffer2, 64, "%s %s: %s",
01044                 gettext ("Variable"),
01045                 buffer, gettext ("maximum range not allowed"));
01046             msg = buffer2;
01047             goto exit_on_error;
01048         }
01049     }
01050     else
01051     {
01052         snprintf (buffer2, 64, "%s %s: %s",
01053             gettext ("Variable"), buffer, gettext ("no maximum range"));
01054         msg = buffer2;
01055         goto exit_on_error;
01056     }
01057     if (input->rangemax[input->nvariables]
01058         < input->rangemin[input->nvariables])
01059     {
01060         snprintf (buffer2, 64, "%s %s: %s",
01061             gettext ("Variable"), buffer, gettext ("bad range"));
01062         msg = buffer2;
01063         goto exit_on_error;
01064     }
01065     input->precision = g_realloc
01066         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01067     input->precision[input->nvariables]
01068         = xml_node_get_uint_with_default (child,
XML_PRECISION,
01069                                     DEFAULT_PRECISION, &error_code);
01070     if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
01071     {
01072         snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01073             gettext ("bad precision"));
01074         msg = buffer2;
01075         goto exit_on_error;
01076     }
01077     if (input->algorithm == ALGORITHM_SWEEP)
01078     {
01079         if (xmlHasProp (child, XML_NSWEEPS))
01080         {
01081             input->nsweeps = (unsigned int *)
01082                 g_realloc (input->nsweeps,
01083                     (1 + input->nvariables) * sizeof (unsigned int));
01084             input->nsweeps[input->nvariables]
01085                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01086             if (error_code || !input->nsweeps[input->
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     }
01096     {

```

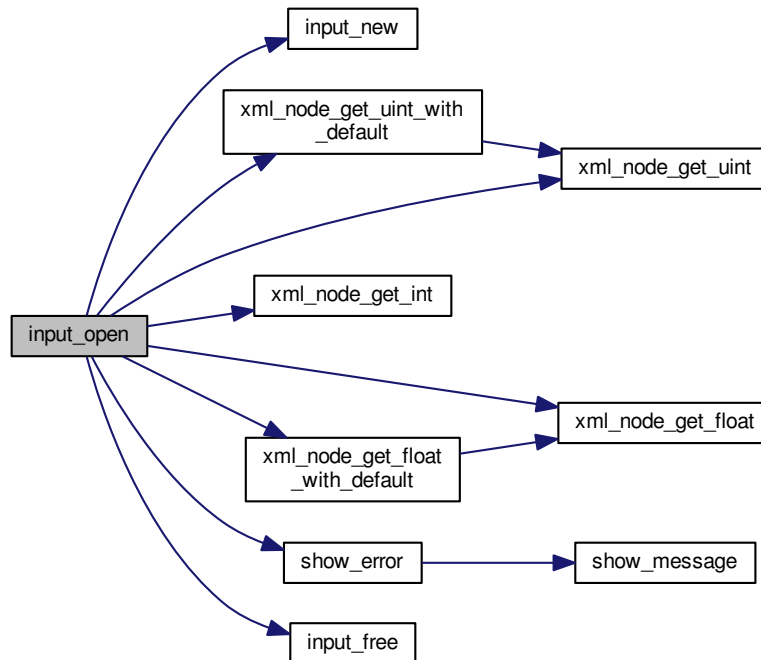
```

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 #if DEBUG
01103     fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01104             input->nsweeps[input->nvariables],
01105             input->nsimulations);
01106 #endif
01107     if (input->algorithm == ALGORITHM_GENETIC)
01108     {
01109         // Obtaining bits representing each variable
01110         if (xmlHasProp (child, XML_NBITS))
01111         {
01112             input->nbits = (unsigned int *)
01113                 g_realloc (input->nbits,
01114                             (1 + input->nvariables) * sizeof (unsigned int));
01115             i = xml_node_get_uint (child, XML_NBITS, &error_code);
01116             if (error_code || !i)
01117             {
01118                 snprintf (buffer2, 64, "%s %s: %s",
01119                             gettext ("Variable"),
01120                             buffer, gettext ("invalid bits number"));
01121                 msg = buffer2;
01122                 goto exit_on_error;
01123             }
01124             input->nbits[input->nvariables] = i;
01125         }
01126         else
01127         {
01128             snprintf (buffer2, 64, "%s %s: %s",
01129                         gettext ("Variable"),
01130                         buffer, gettext ("no bits number"));
01131             msg = buffer2;
01132             goto exit_on_error;
01133         }
01134     }
01135     else if (input->nsteps)
01136     {
01137         input->step = (double *)
01138             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01139         input->step[input->nvariables]
01140             = xml_node_get_float (child, XML_STEP, &error_code);
01141         if (error_code || input->step[input->nvariables] < 0.)
01142         {
01143             snprintf (buffer2, 64, "%s %s: %s",
01144                         gettext ("Variable"),
01145                         buffer, gettext ("bad step size"));
01146             msg = buffer2;
01147             goto exit_on_error;
01148         }
01149     }
01150     input->label = g_realloc
01151         (input->label, (1 + input->nvariables) * sizeof (char *));
01152     input->label[input->nvariables] = (char *) buffer;
01153     ++input->nvariables;
01154 }
01155 if (!input->nvariables)
01156 {
01157     msg = gettext ("No calibration variables");
01158     goto exit_on_error;
01159 }
01160 buffer = NULL;
01161
01162 // Getting the working directory
01163 input->directory = g_path_get_dirname (filename);
01164 input->name = g_path_get_basename (filename);
01165
01166 // Closing the XML document
01167 xmlFreeDoc (doc);
01168
01169 #if DEBUG
01170     fprintf (stderr, "input_open: end\n");
01171 #endif
01172     return 1;
01173
01174 exit_on_error:
01175     xmlFree (buffer);
01176     xmlFreeDoc (doc);
01177     show_error (msg);
01178     input_free ();
01179 #if DEBUG
01180     fprintf (stderr, "input_open: end\n");
01181 #endif
01182     return 0;

```

```
01183 }
```

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

```

00258 {
00259     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00260 }
```

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

```

00228 {
00229     #if HAVE_GTK
00230         GtkMessageDialog *dlg;
00231
00232         // Creating the dialog
00233         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00234             (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00235
00236         // Setting the dialog title
00237         gtk_window_set_title (GTK_WINDOW (dlg), title);
00238
00239         // Showing the dialog and waiting response
00240         gtk_dialog_run (GTK_DIALOG (dlg));
00241
00242         // Closing and freeing memory
00243         gtk_widget_destroy (GTK_WIDGET (dlg));
00244
00245     #else
00246         printf ("%s: %s\n", title, msg);
00247     #endif
00248 }
```

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

```

00368 {
00369     double x = 0.;
00370     xmlChar *buffer;
00371     buffer = xmlGetProp (node, prop);
00372     if (!buffer)
00373         *error_code = 1;
00374     else
00375     {
00376         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00377             *error_code = 2;
00378         else
00379             *error_code = 0;
00380         xmlFree (buffer);
00381     }
00382     return x;
00383 }
```

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

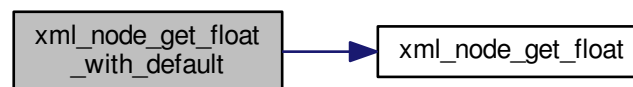
Floating point number value.

Definition at line 401 of file [mpcotool.c](#).

```

00403 {
00404     double x;
00405     if (xmlHasProp (node, prop))
00406         x = xml_node_get_float (node, prop, error_code);
00407     else
00408     {
00409         x = default_value;
00410         *error_code = 0;
00411     }
00412     return x;
00413 }
```

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

```

00276 {
00277     int i = 0;
00278     xmlChar *buffer;
00279     buffer = xmlGetProp (node, prop);
00280     if (!buffer)
00281         *error_code = 1;
00282     else
00283     {
00284         if (sscanf ((char *) buffer, "%d", &i) != 1)
```

```

00285         *error_code = 2;
00286     else
00287         *error_code = 0;
00288     xmlFree (buffer);
00289 }
00290 return i;
00291 }

```

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

```

00307 {
00308     unsigned int i = 0;
00309     xmlChar *buffer;
00310     buffer = xmlGetProp (node, prop);
00311     if (!buffer)
00312         *error_code = 1;
00313     else
00314     {
00315         if (sscanf ((char *) buffer, "%u", &i) != 1)
00316             *error_code = 2;
00317         else
00318             *error_code = 0;
00319         xmlFree (buffer);
00320     }
00321     return i;
00322 }

```

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

```

00342 {
00343     unsigned int i;
00344     if (xmlHasProp (node, prop))
00345         i = xml_node_get_uint (node, prop, error_code);
00346     else
00347     {
00348         i = default_value;

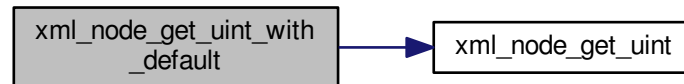
```

```

00349     *error_code = 0;
00350 }
00351 return i;
00352 }

```

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

```

00465 {
00466     xmlChar buffer[64];
00467     snprintf ((char *) buffer, 64, "%.14lg", value);
00468     xmlSetProp (node, prop, buffer);
00469 }

```

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

```

00427 {
00428     xmlChar buffer[64];
00429     snprintf ((char *) buffer, 64, "%d", value);
00430     xmlSetProp (node, prop, buffer);
00431 }

```

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 445 of file `mpcotool.c`.

```

00446 {
00447     xmlChar buffer[64];
00448     snprintf ((char *) buffer, 64, "%u", value);
00449     xmlSetProp (node, prop, buffer);
00450 }
```

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
 - mpcotool.h, [139](#)
- ALGORITHM_MONTE_CARLO
 - mpcotool.h, [139](#)
- ALGORITHM_SWEEP
 - mpcotool.h, [139](#)
- Algorithm
 - mpcotool.h, [139](#)
- Calibrate, [13](#)
 - thread_gradient, [15](#)
- calibrate_best
 - mpcotool.c, [46](#)
 - mpcotool.h, [140](#)
- calibrate_best_gradient
 - mpcotool.c, [46](#)
 - mpcotool.h, [140](#)
- calibrate_estimate_gradient_coordinates
 - mpcotool.c, [47](#)
- calibrate_estimate_gradient_random
 - mpcotool.c, [47](#)
- calibrate_genetic_objective
 - mpcotool.c, [48](#)
 - mpcotool.h, [141](#)
- calibrate_gradient_sequential
 - mpcotool.c, [49](#)
- calibrate_gradient_thread
 - mpcotool.c, [49](#)
 - mpcotool.h, [141](#)
- calibrate_input
 - mpcotool.c, [51](#)
 - mpcotool.h, [142](#)
- calibrate_merge
 - mpcotool.c, [52](#)
 - mpcotool.h, [143](#)
- calibrate_parse
 - mpcotool.c, [53](#)
 - mpcotool.h, [144](#)
- calibrate_save_variables
 - mpcotool.c, [55](#)
 - mpcotool.h, [146](#)
- calibrate_step_gradient
 - mpcotool.c, [55](#)
 - mpcotool.h, [146](#)
- calibrate_thread
 - mpcotool.c, [57](#)
 - mpcotool.h, [148](#)
- config.h, [25](#)
- cores_number
 - interface.h, [31](#)
- mpcotool.c, [58](#)
- Experiment, [15](#)
- format
 - mpcotool.c, [82](#)
- GRADIENT_METHOD_COORDINATES
 - mpcotool.h, [139](#)
- GRADIENT_METHOD_RANDOM
 - mpcotool.h, [139](#)
- GradientMethod
 - mpcotool.h, [139](#)
- Input, [16](#)
- input_open
 - mpcotool.c, [58](#)
 - mpcotool.h, [149](#)
- input_save
 - interface.h, [31](#)
 - mpcotool.c, [66](#)
- input_save_gradient
 - mpcotool.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
 - mpcotool.c, [70](#)
- mpcotool.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
- mpcotoool.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
 - mpcotoool.c, 82
- Running, 19
- show_error
 - mpcotoool.c, 72
 - mpcotoool.h, 157
- show_message
 - mpcotoool.c, 73
 - mpcotoool.h, 157
- template
 - mpcotoool.c, 83
- thread_gradient
 - Calibrate, 15
- Variable, 19
- Window, 20
- window_get_algorithm
 - interface.h, 33
 - mpcotoool.c, 73
- window_get_gradient
 - interface.h, 34
 - mpcotoool.c, 74
- window_read
 - interface.h, 34
 - mpcotoool.c, 74
- window_save
 - interface.h, 36
 - mpcotoool.c, 76
- window_template_experiment
 - interface.h, 38
 - mpcotoool.c, 78
- xml_node_get_float
 - mpcotoool.c, 78
 - mpcotoool.h, 159
- xml_node_get_float_with_default
 - mpcotoool.c, 79
 - mpcotoool.h, 159
- xml_node_get_int
 - mpcotoool.c, 80
 - mpcotoool.h, 160
- xml_node_get_uint
 - mpcotoool.c, 80
 - mpcotoool.h, 161
- xml_node_get_uint_with_default
 - mpcotoool.c, 81
 - mpcotoool.h, 161
- xml_node_set_float
 - mpcotoool.c, 81
 - mpcotoool.h, 162
- xml_node_set_int
 - mpcotoool.c, 82
 - mpcotoool.h, 162
- xml_node_set_uint
 - mpcotoool.c, 82
 - mpcotoool.h, 162