Calibrator

1.2.4

Generated by Doxygen 1.8.9.1

Mon Jan 11 2016 05:32:19

Contents

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 objetive 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 <?xml version="1.0"?> <calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm_type" nsimulations="simulations_number" niterations="iterations_number" tolerance="tolerance_value" nbest="best_number" npopulation="population_number" ngenerations="generations_number" mutation="mutation_ratio" reproduction="reproduction_ratio" adaptation="adaptation_ratio" gradient_type="gradient_method_type" nsteps="steps_number" relaxation="relaxation_paramter" nestimates="estimates_number" seed="random_seed" result="result_file" variables="variables_file"> <experiment name="data_file_1" template1="template_1_1" template2="template_1_2" ... weight="weight_1"/> ... <experiment name="data_file_N" template1="template_N_1" template2="template_N_2" ... weight="weight_N"/> <variable name="variable_1" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> ... <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> </calibrate> ‘‘‘

with:

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

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

Implemented algorithms are:

- **sweep**: Sweep brute force algorithm. It requires for each variable:
  - *sweeps*: number of sweeps to generate for each variable in every experiment.
    The total number of simulations to run is:

    > (number of experiments) x (variable 1 number of sweeps) x ... x (variable n number of sweeps) x (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:

    > (number of experiments) x (number of simulations) x (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:

    > x (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:

      > (number of experiments) x (number of iterations) x (number of steps) x 2 x (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:

      > (number of experiments) x (number of iterations) x (number of steps) x (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 <?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@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03008, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03753, "@variable1@" : @, "@variable2@" : @, "@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:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

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

Data Fields

- GMappedFile ∗∗ file [MAX_NINPUTS]

    Matrix of input template files.
- char ∗∗ template [MAX_NINPUTS]

    Matrix of template names of input files.
- char ∗∗ experiment

    Array of experimental data file names.
- char ∗∗ label

    Array of variable names.
- gsl_rng ∗ rng

    GSL random number generator.
- GeneticVariable ∗ genetic_variable

    Array of variables for the genetic algorithm.
- FILE ∗ file_result

    Result file.
- FILE ∗ file_variables

    Variables file.
- char ∗ result

    Name of the result file.
- char ∗ variables

    Name of the variables file.
- char ∗ simulator

    Name of the simulator program.
- char ∗ evaluator

    Name of the program to evaluate the objective function.
- double ∗ value

    Array of variable values.
- double ∗ rangemin

    Array of minimum variable values.
- double ∗ rangemax

Array of maximum variable values.

- double ∗ rangeminabs

  Array of absolute minimum variable values.
- double ∗ rangemaxabs

  Array of absolute maximum variable values.
- double ∗ error_best

  Array of the best minimum errors.
- double ∗ weight

  Array of the experiment weights.
- double ∗ step

  Array of gradient based method step sizes.
- double ∗ gradient

  Vector of gradient estimation.
- double ∗ value_old

  Array of the best variable values on the previous step.
- double ∗ error_old

  Array of the best minimum errors on the previous step.
- unsigned int ∗ precision

  Array of variable precisions.
- unsigned int ∗ nsweeps

  Array of sweeps of the sweep algorithm.
- unsigned int ∗ thread

  Array of simulation numbers to calculate on the thread.
- unsigned int ∗ thread_gradient
- unsigned int ∗ simulation_best

  Array of best simulation numbers.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- double calculation_time

  Calculation time.
- unsigned long int seed

  Seed of the pseudo-random numbers generator.
- unsigned int nvariables

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

  Number of input files to the simulator.
- unsigned int nsimulations

  Simulations number per experiment.
- unsigned int gradient_method

  Method to estimate the gradient.
- unsigned int nsteps

Number of steps for the gradient based method.
- unsigned int nestimates

  Number of simulations to estimate the gradient.
- unsigned int algorithm

  Algorithm type.
- unsigned int nstart

  Beginning simulation number of the task.
- unsigned int nend

  Ending simulation number of the task.
- unsigned int nstart_gradient

  Beginning simulation number of the task for the gradient based method.
- unsigned int nend_gradient

  Ending simulation number of the task for the gradient based method.
- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  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 mpcotool.h.

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

- mpcotool.h

4.4 Options Struct Reference

Struct to define the options dialog.

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

Data Fields

- GtkDialog ∗ dialog

  Main GtkDialog.
- GtkGrid ∗ grid

  Main GtkGrid.
- GtkLabel ∗ label_seed

  Pseudo-random numbers generator seed GtkLabel.
- GtkSpinButton ∗ spin_seed

  Pseudo-random numbers generator seed GtkSpinButton.
- GtkLabel ∗ label_threads

  Threads number GtkLabel.
- GtkSpinButton ∗ spin_threads

  Threads number GtkSpinButton.
- GtkLabel ∗ label_gradient

  Gradient threads number GtkLabel.
- GtkSpinButton ∗ spin_gradient

  Gradient threads number GtkSpinButton.

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:

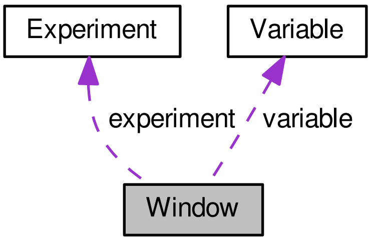
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

- GtkWindow ∗ window

    Main GtkWindow.
- GtkGrid ∗ grid

    Main GtkGrid.
- GtkToolbar ∗ bar_buttons

    GtkToolbar to store the main buttons.
- GtkToolButton ∗ button_open

    Open GtkToolButton.
- GtkToolButton ∗ button_save

    Save GtkToolButton.
- GtkToolButton ∗ button_run

    Run GtkToolButton.
- GtkToolButton ∗ button_options

    Options GtkToolButton.
- GtkToolButton ∗ button_help

    Help GtkToolButton.
- GtkToolButton ∗ button_about

    Help GtkToolButton.
- GtkToolButton ∗ button_exit

Exit GtkToolButton.

- GtkGrid ∗ grid_files

  Files GtkGrid.
- GtkLabel ∗ label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton ∗ button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton ∗ check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton ∗ button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkLabel ∗ label_result

  Result file GtkLabel.
- GtkEntry ∗ entry_result

  Result file GtkEntry.
- GtkLabel ∗ label_variables

  Variables file GtkLabel.
- GtkEntry ∗ entry_variables

  Variables file GtkEntry.
- GtkFrame ∗ frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid ∗ grid_algorithm

  GtkGrid to set the algorithm.
- GtkRadioButton ∗ button_algorithm [NALGORITHMS]

  Array of GtkButtons to set the algorithm.
- GtkLabel ∗ label_simulations

  GtkLabel to set the simulations number.
- GtkSpinButton ∗ spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel ∗ label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton ∗ spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel ∗ label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton ∗ spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel ∗ label_bests

  GtkLabel to set the best number.
- GtkSpinButton ∗ spin_bests

  GtkSpinButton to set the best number.
- GtkLabel ∗ label_population

  GtkLabel to set the population number.
- GtkSpinButton ∗ spin_population

  GtkSpinButton to set the population number.
- GtkLabel ∗ label_generations

  GtkLabel to set the generations number.
- GtkSpinButton ∗ spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel ∗ label_mutation

  GtkLabel to set the mutation ratio.

- GtkSpinButton ∗ spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel ∗ label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton ∗ spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel ∗ label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton ∗ spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton ∗ check_gradient

  GtkCheckButton 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

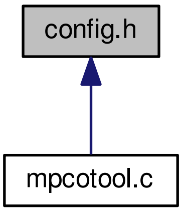
Chapter 5

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"

  algoritm 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

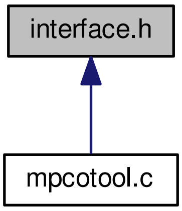
```
/* config.h.  Generated from config.h.in by configure.  */
/*
MPCOTool: a software to make calibrations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2014, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef CONFIG__H
#define CONFIG__H 1

// Array sizes

#define MAX_NINPUTS 8
#define NALGORITHMS 3
#define NGRADIENTS 2
#define NPRECISIONS 15

// Default choices

#define DEFAULT_PRECISION (NPRECISIONS - 1)
#define DEFAULT_RANDOM_SEED 7007
#define DEFAULT_RELAXATION 1.

// Interface labels

#define LOCALE_DIR "locales"
#define PROGRAM_INTERFACE "mpcotool"

// XML labels

#define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
#define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
#define XML_ADAPTATION (const xmlChar*)"adaptation"
#define XML_ALGORITHM (const xmlChar*)"algorithm"
#define XML_CALIBRATE (const xmlChar*)"calibrate"
#define XML_COORDINATES (const xmlChar*)"coordinates"
#define XML_EVALUATOR (const xmlChar*)"evaluator"
#define XML_EXPERIMENT (const xmlChar*)"experiment"
#define XML_GENETIC (const xmlChar*)"genetic"
#define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
#define XML_MINIMUM (const xmlChar*)"minimum"
#define XML_MAXIMUM (const xmlChar*)"maximum"
#define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
#define XML_MUTATION (const xmlChar*)"mutation"
#define XML_NAME (const xmlChar*)"name"
#define XML_NBEST (const xmlChar*)"nbest"
#define XML_NBITS (const xmlChar*)"nbits"
#define XML_NESTIMATES (const xmlChar*)"nestimates"
#define XML_NGENERATIONS (const xmlChar*)"ngenerations"
#define XML_NITERATIONS (const xmlChar*)"niterations"
#define XML_NPOPULATION (const xmlChar*)"npopulation"
#define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
#define XML_NSTEPS (const xmlChar*)"nsteps"
#define XML_NSWEEPS (const xmlChar*)"nsweeps"
#define XML_PRECISION (const xmlChar*)"precision"
#define XML_RANDOM (const xmlChar*)"random"
#define XML_RELAXATION (const xmlChar*)"relaxation"
#define XML_REPRODUCTION (const xmlChar*)"reproduction"
#define XML_RESULT (const xmlChar*)"result"
#define XML_SIMULATOR (const xmlChar*)"simulator"
#define XML_SEED (const xmlChar*)"seed"
```

```
#define XML_STEP (const xmlChar*)"step"
#define XML_SWEEP (const xmlChar*)"sweep"
#define XML_TEMPLATE1 (const xmlChar*)"template1"
#define XML_TEMPLATE2 (const xmlChar*)"template2"
#define XML_TEMPLATE3 (const xmlChar*)"template3"
#define XML_TEMPLATE4 (const xmlChar*)"template4"
#define XML_TEMPLATE5 (const xmlChar*)"template5"
#define XML_TEMPLATE6 (const xmlChar*)"template6"
#define XML_TEMPLATE7 (const xmlChar*)"template7"
#define XML_TEMPLATE8 (const xmlChar*)"template8"
#define XML_TOLERANCE (const xmlChar*)"tolerance"
#define XML_VARIABLE (const xmlChar*)"variable"
#define XML_VARIABLES (const xmlChar*)"variables"
#define XML_WEIGHT (const xmlChar*)"weight"

#endif
```

5.3 interface.h File Reference

Header file of the interface.

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



Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- void input_save (char ∗filename)

  Function to save the input file.
- void options_new ()

  Function to open the options dialog.
- void running_new ()

  Function to open the running dialog.
- int window_get_algorithm ()

  Function to get the stochastic algorithm number.
- int window_get_gradient ()

  Function to get the gradient base method number.
- void window_save_gradient ()

  Function to save the gradient based method data in the input file.
- int window_save ()

  Function to save the input file.
- void window_run ()

  Function to run a calibration.
- void window_help ()

  Function to show a help dialog.
- void window_update_gradient ()

  Function to update gradient based method widgets view in the main window.
- void window_update ()

  Function to update the main window view.
- void window_set_algorithm ()

  Function to avoid memory errors changing the algorithm.
- void window_set_experiment ()

  Function to set the experiment data in the main window.
- void window_remove_experiment ()

  Function to remove an experiment in the main window.
- void window_add_experiment ()

  Function to add an experiment in the main window.
- void window_name_experiment ()

  Function to set the experiment name in the main window.
- void window_weight_experiment ()

  Function to update the experiment weight in the main window.
- void window_inputs_experiment ()

  Function to update the experiment input templates number in the main window.
- void window_template_experiment (void ∗data)

  Function to update the experiment i-th input template in the main window.
- void window_set_variable ()

  Function to set the variable data in the main window.
- void window_remove_variable ()

  Function to remove a variable in the main window.
- void window_add_variable ()

  Function to add a variable in the main window.
- void window_label_variable ()

  Function to set the variable label in the main window.
- void window_precision_variable ()

  Function to update the variable precision in the main window.
- void window_rangemin_variable ()

Function to update the variable rangemin in the main window.

- void window_rangemax_variable ()

  Function to update the variable rangemax in the main window.
- void window_rangeminabs_variable ()

  Function to update the variable rangeminabs in the main window.
- void window_rangemaxabs_variable ()

  Function to update the variable rangemaxabs in the main window.
- void window_update_variable ()

  Function to update the variable data in the main window.
- int window_read (char ∗filename)

  Function to read the input data of a file.
- void window_open ()

  Function to open the input data.
- void window_new ()

  Function to open the main window.
- int cores_number ()

  Function to obtain the cores number.

5.3.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Copyright 2012-2015, all rights reserved.

Definition in file interface.h.

5.3.2 Function Documentation

5.3.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4875 of file mpcotool.c.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

5.3.2.2 void input_save (char ∗ *filename*)

Function to save the input file.

Parameters

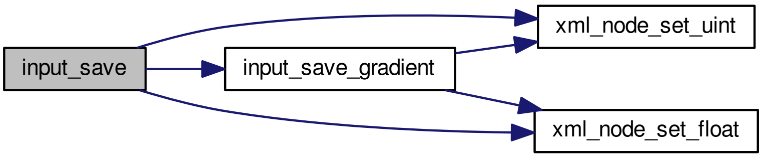
filename	Input file name.

Definition at line 2699 of file mpcotool.c.

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

```
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
      xml_node_set_float (child, XML_MINIMUM, input->rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->rangeminabs[i]);
      xml_node_set_float (child, XML_MAXIMUM, input->rangemax[i]);
      if (input->rangemaxabs[i] != G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->rangemaxabs[i]);
      if (input->precision[i] != DEFAULT_PRECISION)
        xml_node_set_uint (child, XML_PRECISION, input->precision[i]);
      if (input->algorithm == ALGORITHM_SWEEP)
        xml_node_set_uint (child, XML_NSWEEPS, input->nsweeps[i]);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, XML_NBITS, input->nbits[i]);
      if (input->nsteps)
        xml_node_set_float (child, XML_STEP, input->step[i]);
    }

  // Saving the XML file
  xmlSaveFormatFile (filename, doc, 1);

  // Freeing memory
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_save: end\n");
#endif
}
```

Here is the call graph for this function:



5.3.2.3 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2932 of file mpcotool.c.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  for (i = 0; i < NALGORITHMS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}
```

5.3.2.4 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2955 of file mpcotool.c.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_gradient: start\n");
#endif
  for (i = 0; i < NGRADIENTS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_gradient: %u\n", i);
  fprintf (stderr, "window_get_gradient: end\n");
#endif
  return i;
}
```

5.3.2.5 int window_read (char ∗ *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

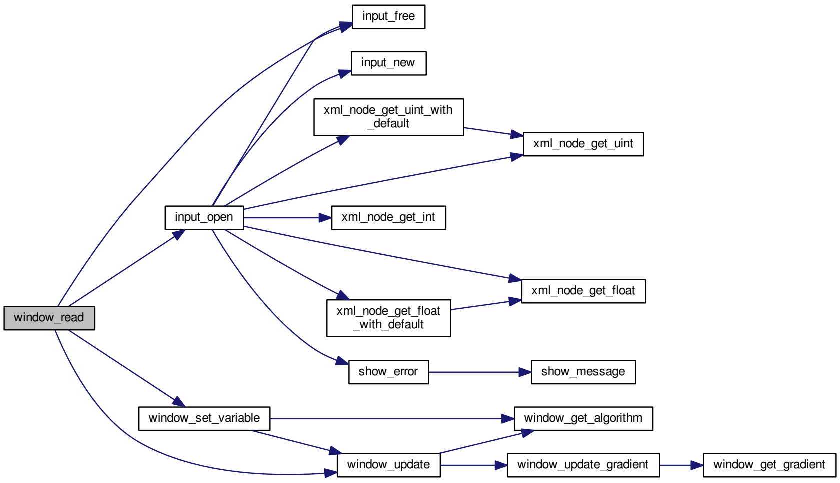
1 on succes, 0 on error.

Definition at line 4052 of file mpcotool.c.

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

  // Reading new input file
  input_free ();
  if (!input_open (filename))
    return 0;

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_gradient
                                [input->gradient_method]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->gradient_method)
            {
            case GRADIENT_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i]);
  g_signal_handler_unblock
```

```
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}
```

Here is the call graph for this function:



5.3.2.6 int window_save ( )

Function to save the input file.

Returns

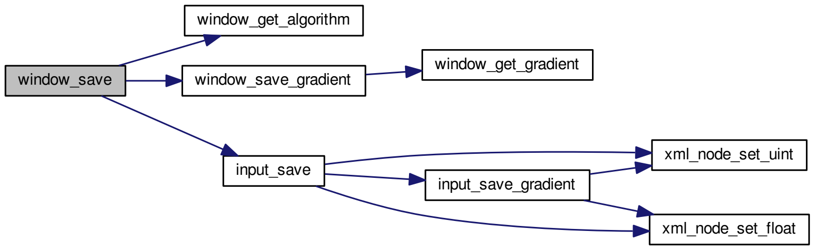
1 on OK, 0 on Cancel.

Definition at line 3010 of file mpcotool.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer;

#if DEBUG
  fprintf (stderr, "window_save: start\n");
#endif

```

```
  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 gettext ("_Cancel"),
                                 GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_gradient ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_gradient ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}
```

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

data	Callback data (i-th input template).

Definition at line 3656 of file mpcotool.c.

```
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}
```

5.4 interface.h

```
/*
MPCOTool: a software to make calibrations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

```

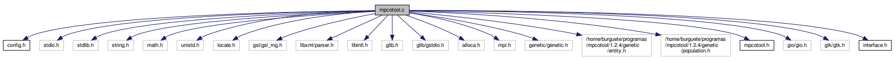
```
Copyright 2012-2015, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef INTERFACE__H
#define INTERFACE__H 1

#define MAX_LENGTH (DEFAULT_PRECISION + 8)

typedef struct
{
  char *template[MAX_NINPUTS];
  char *name;
  double weight;
} Experiment;

typedef struct
{
  char *label;
  double rangemin;
  double rangemax;
  double rangeminabs;
  double rangemaxabs;
  double step;
  unsigned int precision;
  unsigned int nsweeps;
  unsigned int nbits;
} Variable;

typedef struct
{
  GtkDialog *dialog;
  GtkGrid *grid;
  GtkLabel *label_seed;
  GtkSpinButton *spin_seed;
  GtkLabel *label_threads;
  GtkSpinButton *spin_threads;
  GtkLabel *label_gradient;
  GtkSpinButton *spin_gradient;
} Options;

typedef struct
{
  GtkDialog *dialog;
  GtkLabel *label;
} Running;

typedef struct
{
  GtkWindow *window;
  GtkGrid *grid;
  GtkToolbar *bar_buttons;
  GtkToolButton *button_open;
  GtkToolButton *button_save;
  GtkToolButton *button_run;
  GtkToolButton *button_options;
  GtkToolButton *button_help;
  GtkToolButton *button_about;
  GtkToolButton *button_exit;
  GtkGrid *grid_files;
  GtkLabel *label_simulator;
  GtkFileChooserButton *button_simulator;
  GtkCheckButton *check_evaluator;
  GtkFileChooserButton *button_evaluator;
  GtkLabel *label_result;
  GtkEntry *entry_result;
  GtkLabel *label_variables;
  GtkEntry *entry_variables;
  GtkFrame *frame_algorithm;
  GtkGrid *grid_algorithm;
  GtkRadioButton *button_algorithm[NALGORITHMS];
  GtkLabel *label_simulations;
  GtkSpinButton *spin_simulations;
  GtkLabel *label_iterations;
  GtkSpinButton *spin_iterations;
  GtkLabel *label_tolerance;
  GtkSpinButton *spin_tolerance;
  GtkLabel *label_bests;
  GtkSpinButton *spin_bests;
  GtkLabel *label_population;
  GtkSpinButton *spin_population;
  GtkLabel *label_generations;
  GtkSpinButton *spin_generations;
  GtkLabel *label_mutation;
  GtkSpinButton *spin_mutation;
  GtkLabel *label_reproduction;
  GtkSpinButton *spin_reproduction;
  GtkLabel *label_adaptation;
  GtkSpinButton *spin_adaptation;
  GtkCheckButton *check_gradient;
  GtkGrid *grid_gradient;
  GtkRadioButton *button_gradient[NGRADIENTS];
  GtkLabel *label_steps;
  GtkSpinButton *spin_steps;
  GtkLabel *label_estimates;
  GtkSpinButton *spin_estimates;
  GtkLabel *label_relaxation;
  GtkSpinButton *spin_relaxation;
  GtkFrame *frame_variable;
  GtkGrid *grid_variable;
  GtkComboBoxText *combo_variable;
  GtkButton *button_add_variable;
  GtkButton *button_remove_variable;
  GtkLabel *label_variable;
  GtkEntry *entry_variable;
  GtkLabel *label_min;
  GtkSpinButton *spin_min;
  GtkScrolledWindow *scrolled_min;
  GtkLabel *label_max;
  GtkSpinButton *spin_max;
  GtkScrolledWindow *scrolled_max;
  GtkCheckButton *check_minabs;
  GtkSpinButton *spin_minabs;
  GtkScrolledWindow *scrolled_minabs;
  GtkCheckButton *check_maxabs;
  GtkSpinButton *spin_maxabs;
  GtkScrolledWindow *scrolled_maxabs;
  GtkLabel *label_precision;
  GtkSpinButton *spin_precision;
  GtkLabel *label_sweeps;
  GtkSpinButton *spin_sweeps;
  GtkLabel *label_bits;
  GtkSpinButton *spin_bits;
  GtkLabel *label_step;
  GtkSpinButton *spin_step;
  GtkScrolledWindow *scrolled_step;
  GtkFrame *frame_experiment;
  GtkGrid *grid_experiment;
  GtkComboBoxText *combo_experiment;
  GtkButton *button_add_experiment;
  GtkButton *button_remove_experiment;
  GtkLabel *label_experiment;
  GtkFileChooserButton *button_experiment;
  GtkLabel *label_weight;
  GtkSpinButton *spin_weight;
  GtkCheckButton *check_template[MAX_NINPUTS];
  GtkFileChooserButton *button_template[MAX_NINPUTS];
  GdkPixbuf *logo;
  Experiment *experiment;
  Variable *variable;
  char *application_directory;
  gulong id_experiment;
  gulong id_experiment_name;
  gulong id_variable;
  gulong id_variable_label;
  gulong id_template[MAX_NINPUTS];
  gulong id_input[MAX_NINPUTS];
  unsigned int nexperiments;
  unsigned int nvariables;
} Window;

// Public functions
void input_save (char *filename);
void options_new ();
void running_new ();
int window_get_algorithm ();
int window_get_gradient ();
void window_save_gradient ();
int window_save ();
void window_run ();
void window_help ();
void window_update_gradient ();
void window_update ();
void window_set_algorithm ();
void window_set_experiment ();
void window_remove_experiment ();
void window_add_experiment ();
void window_name_experiment ();
void window_weight_experiment ();
void window_inputs_experiment ();
void window_template_experiment (void *data);
void window_set_variable ();
void window_remove_variable ();
void window_add_variable ();
void window_label_variable ();
void window_precision_variable ();
void window_rangemin_variable ();
void window_rangemax_variable ();
void window_rangeminabs_variable ();
void window_rangemaxabs_variable ();
void window_update_variable ();
int window_read (char *filename);
void window_open ();
void window_new ();
int cores_number ();

#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

simulation	Simulation number.
value	Objective function value.

Definition at line 1443 of file mpcotool.c.

```
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_best: start\n");
  fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
           calibrate->nsaveds, calibrate->nbest);
#endif
  if (calibrate->nsaveds < calibrate->nbest
      || value < calibrate->error_best[calibrate->nsaveds - 1])
    {
      if (calibrate->nsaveds < calibrate->nbest)
        ++calibrate->nsaveds;
      calibrate->error_best[calibrate->nsaveds - 1] = value;
      calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
      for (i = calibrate->nsaveds; --i;)
        {
          if (calibrate->error_best[i] < calibrate->error_best[i - 1])
            {
              j = calibrate->simulation_best[i];
              e = calibrate->error_best[i];
              calibrate->simulation_best[i] = calibrate->simulation_best[i - 1];
              calibrate->error_best[i] = calibrate->error_best[i - 1];
              calibrate->simulation_best[i - 1] = j;
              calibrate->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_best: end\n");
#endif
}
```

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

simulation	Simulation number.
value	Objective function value.

Definition at line 1756 of file mpcotool.c.

```
{
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: start\n");
  fprintf (stderr,
           "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, calibrate->error_best[0]);
#endif
  if (value < calibrate->error_best[0])
    {
      calibrate->error_best[0] = value;
      calibrate->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: end\n");
#endif
}
```

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

variable	Variable number.
estimate	Estimate number.

Definition at line 1893 of file mpcotool.c.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
#endif
  x = calibrate->gradient[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += calibrate->step[variable];
      else
        x -= calibrate->step[variable];
    }
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
           variable, x);
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
#endif
  return x;
}
```

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

variable	Variable number.
estimate	Estimate number.

Definition at line 1866 of file mpcotool.c.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
#endif
  x = calibrate->gradient[variable]
    + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
           variable, x);
  fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
#endif
  return x;
}
```

5.5.2.5 double calibrate_genetic_objective (Entity ∗ *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

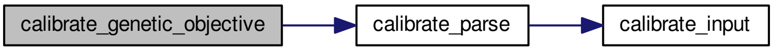
Returns

objective function value.

Definition at line 2059 of file mpcotool.c.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: start\n");
#endif
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      calibrate->value[entity->id * calibrate->nvariables + j]
        = genetic_get_variable (entity, calibrate->genetic_variable + j);
    }
  for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
    objective += calibrate_parse (entity->id, j);
  g_mutex_lock (mutex);
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
      fprintf (calibrate->file_variables, buffer,
               genetic_get_variable (entity, calibrate->genetic_variable + j));
    }
  fprintf (calibrate->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: end\n");
#endif
  return objective;
}
```

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.

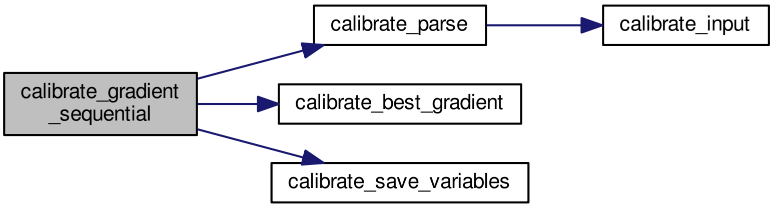
Parameters

simulation	Simulation number.

Definition at line 1786 of file mpcotool.c.

```
{
  unsigned int i, j, k;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_sequential: start\n");
  fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
           "nend_gradient=%u\n",
           calibrate->nstart_gradient, calibrate->
nend_gradient);
#endif
  for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
    {
      k = simulation + i;
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (k, j);
      calibrate_best_gradient (k, e);
      calibrate_save_variables (k, e);
#if DEBUG
      fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient_sequential: end\n");
#endif
}
```

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

data	Function data.

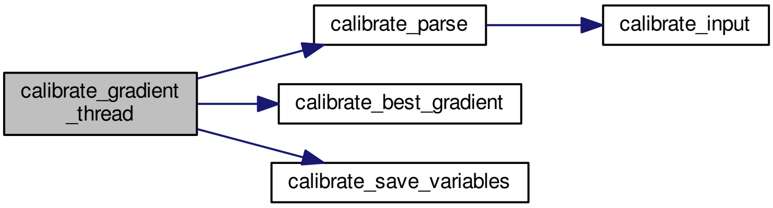
Returns

NULL

Definition at line 1821 of file mpcotool.c.

```
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
           thread,
           calibrate->thread_gradient[thread],
           calibrate->thread_gradient[thread + 1]);
#endif
  for (i = calibrate->thread_gradient[thread];
       i < calibrate->thread_gradient[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best_gradient (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

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

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 1196 of file mpcotool.c.

```
{
  unsigned int i;
  char buffer[32], value[32], *buffer2, *buffer3, *content;
  FILE *file;
  gsize length;
  GRegex *regex;

#if DEBUG
  fprintf (stderr, "calibrate_input: start\n");
#endif

  // Checking the file
  if (!template)
    goto calibrate_input_end;

  // Opening template
  content = g_mapped_file_get_contents (template);
  length = g_mapped_file_get_length (template);
#if DEBUG
  fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
           content);
#endif
  file = g_fopen (input, "w");

  // Parsing template
  for (i = 0; i < calibrate->nvariables; ++i)
    {
#if DEBUG
      fprintf (stderr, "calibrate_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             calibrate->label[i], 0, NULL);
#if DEBUG
          fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             calibrate->label[i], 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      snprintf (value, 32, format[calibrate->precision[i]],
                calibrate->value[simulation * calibrate->
nvariables + i]);

#if DEBUG
      fprintf (stderr, "calibrate_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

calibrate_input_end:
#if DEBUG
  fprintf (stderr, "calibrate_input: end\n");
#endif
  return;
}
```

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

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1561 of file mpcotool.c.

```
{
  unsigned int i, j, k, s[calibrate->nbest];
  double e[calibrate->nbest];
#if DEBUG
  fprintf (stderr, "calibrate_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == calibrate->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
          if (i == calibrate->nsaveds)
            break;
        }
      else if (calibrate->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < calibrate->nbest);
  calibrate->nsaveds = k;
  memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
  memcpy (calibrate->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "calibrate_merge: end\n");
#endif
}
```

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

simulation	Simulation number.
experiment	Experiment number.

Returns

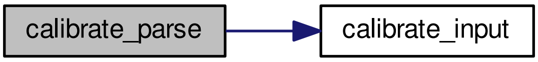
Objective function value.

Definition at line 1283 of file mpcotool.c.

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

```
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "calibrate_parse: end\n");
#endif

  // Returning the objective function
  return e * calibrate->weight[experiment];
}
```

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

simulation	Simulation number.
error	Error value.

Definition at line 1415 of file mpcotool.c.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: start\n");
#endif
  for (i = 0; i < calibrate->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
      fprintf (calibrate->file_variables, buffer,
               calibrate->value[simulation * calibrate->
nvariables + i]);
    }
  fprintf (calibrate->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: end\n");
#endif
}
```

5.5.2.12 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

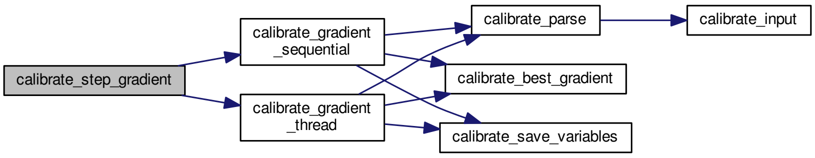
Parameters

simulation	Simulation number.

Definition at line 1923 of file mpcotool.c.

```
{
  GThread *thread[nthreads_gradient];
  ParallelData data[nthreads_gradient];
  unsigned int i, j, k, b;
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: start\n");
#endif
  for (i = 0; i < calibrate->nestimates; ++i)
    {
      k = (simulation + i) * calibrate->nvariables;
      b = calibrate->simulation_best[0] * calibrate->nvariables;
#if DEBUG
      fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
               simulation + i, calibrate->simulation_best[0]);
#endif
      for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
                   i, j, calibrate->value[b]);
#endif
          calibrate->value[k]
            = calibrate->value[b] + calibrate_estimate_gradient (j, i);
          calibrate->value[k] = fmin (fmax (calibrate->value[k],
                                            calibrate->rangeminabs[j]),
                                      calibrate->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
                   i, j, calibrate->value[k]);
#endif
        }
    }
  if (nthreads_gradient == 1)
    calibrate_gradient_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_gradient; ++i)
        {
          calibrate->thread_gradient[i]
            = simulation + calibrate->nstart_gradient
            + i * (calibrate->nend_gradient - calibrate->nstart_gradient)
            / nthreads_gradient;
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: i=%u thread_gradient=%u\n",
                   i, calibrate->thread_gradient[i]);
#endif
        }
      for (i = 0; i < nthreads_gradient; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
        }
      for (i = 0; i < nthreads_gradient; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: end\n");
#endif
}
```

Here is the call graph for this function:



5.5.2.13 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

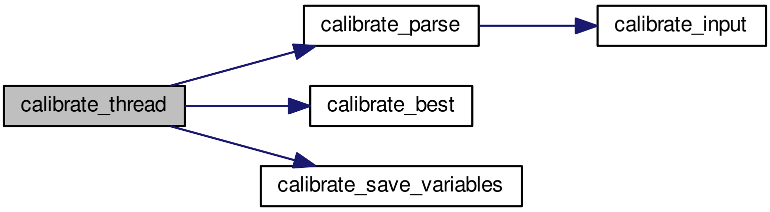
Returns

NULL

Definition at line 1517 of file mpcotool.c.

```
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
           calibrate->thread[thread], calibrate->thread[thread + 1]);
#endif
  for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

Here is the call graph for this function:



5.5.2.14 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4875 of file mpcotool.c.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

5.5.2.15 int input_open (char ∗ *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 548 of file mpcotool.c.

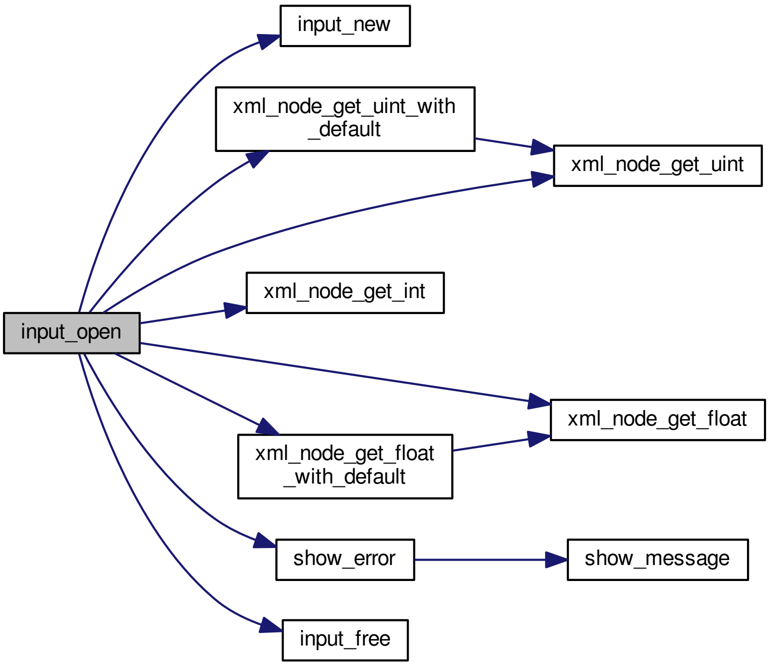
```
{
  char buffer2[64];
  char *buffert[MAX_NINPUTS] =
    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
  xmlDoc *doc;
  xmlNode *node, *child;
  xmlChar *buffer;
  char *msg;
  int error_code;
  unsigned int i;

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif
```

```

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_CALIBRATE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting results file names
  input->result = (char *) xmlGetProp (node, XML_RESULT);
  if (!input->result)
    input->result = (char *) xmlStrdup (result_name);
  input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
  if (!input->variables)
    input->variables = (char *) xmlStrdup (variables_name);

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, XML_NBEST, 1, &error_code);
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node, XML_TOLERANCE, 0.,
                                           &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting gradient method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node, XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->gradient_method = GRADIENT_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->gradient_method = GRADIENT_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the gradient");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child,
XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No calibration experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
```

```
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->
nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables],
input->nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
          input->step[input->nvariables]
            = xml_node_get_float (child, XML_STEP, &error_code);
          if (error_code || input->step[input->nvariables] < 0.)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("bad step size"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      input->label = g_realloc
        (input->label, (1 + input->nvariables) * sizeof (char *));
      input->label[input->nvariables] = (char *) buffer;
      ++input->nvariables;
    }
  if (!input->nvariables)
```

```
    {
      msg = gettext ("No calibration variables");
      goto exit_on_error;
    }
  buffer = NULL;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
  show_error (msg);
  input_free ();
#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



5.5.2.16 void input_save (char * *filename*)

Function to save the input file.

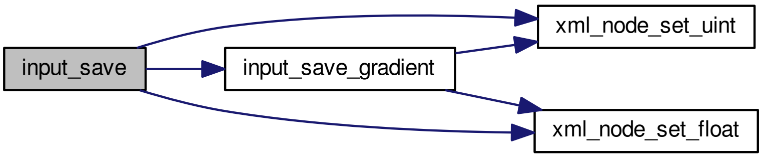
Parameters

filename	Input file name.

Definition at line 2699 of file mpcotool.c.

```
{
  unsigned int i, j;
  char *buffer;
  xmlDoc *doc;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);
  file = g_file_new_for_path (input->directory);

  // Opening the input file
  doc = xmlNewDoc ((const xmlChar *) "1.0");

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp ((const xmlChar *) input->result, result_name))
    xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
  if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
    xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
      xml_node_set_float (child, XML_MINIMUM, input->rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->rangeminabs[i]);
      xml_node_set_float (child, XML_MAXIMUM, input->rangemax[i]);
      if (input->rangemaxabs[i] != G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->rangemaxabs[i]);
      if (input->precision[i] != DEFAULT_PRECISION)
        xml_node_set_uint (child, XML_PRECISION, input->precision[i]);
      if (input->algorithm == ALGORITHM_SWEEP)
        xml_node_set_uint (child, XML_NSWEEPS, input->nsweeps[i]);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, XML_NBITS, input->nbits[i]);
      if (input->nsteps)
        xml_node_set_float (child, XML_STEP, input->step[i]);
    }

  // Saving the XML file
  xmlSaveFormatFile (filename, doc, 1);

  // Freeing memory
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_save: end\n");
#endif
}
```

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.

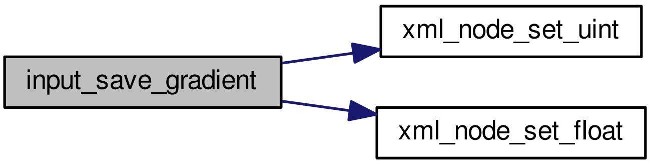
Parameters

node	XML node.

Definition at line 2667 of file mpcotool.c.

```
{
#if DEBUG
  fprintf (stderr, "input_save_gradient: start\n");
#endif
  if (input->nsteps)
    {
      xml_node_set_uint (node, XML_NSTEPS, input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        xml_node_set_float (node, XML_RELAXATION, input->relaxation);
      switch (input->gradient_method)
        {
        case GRADIENT_METHOD_COORDINATES:
          xmlSetProp (node, XML_GRADIENT_METHOD, XML_COORDINATES);
          break;
        default:
          xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
          xml_node_set_uint (node, XML_NESTIMATES, input->nestimates);
        }
    }
#if DEBUG
  fprintf (stderr, "input_save_gradient: end\n");
#endif
}
```

Here is the call graph for this function:



5.5.2.18 int main (int *argn,* char ∗∗ *argc*)

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

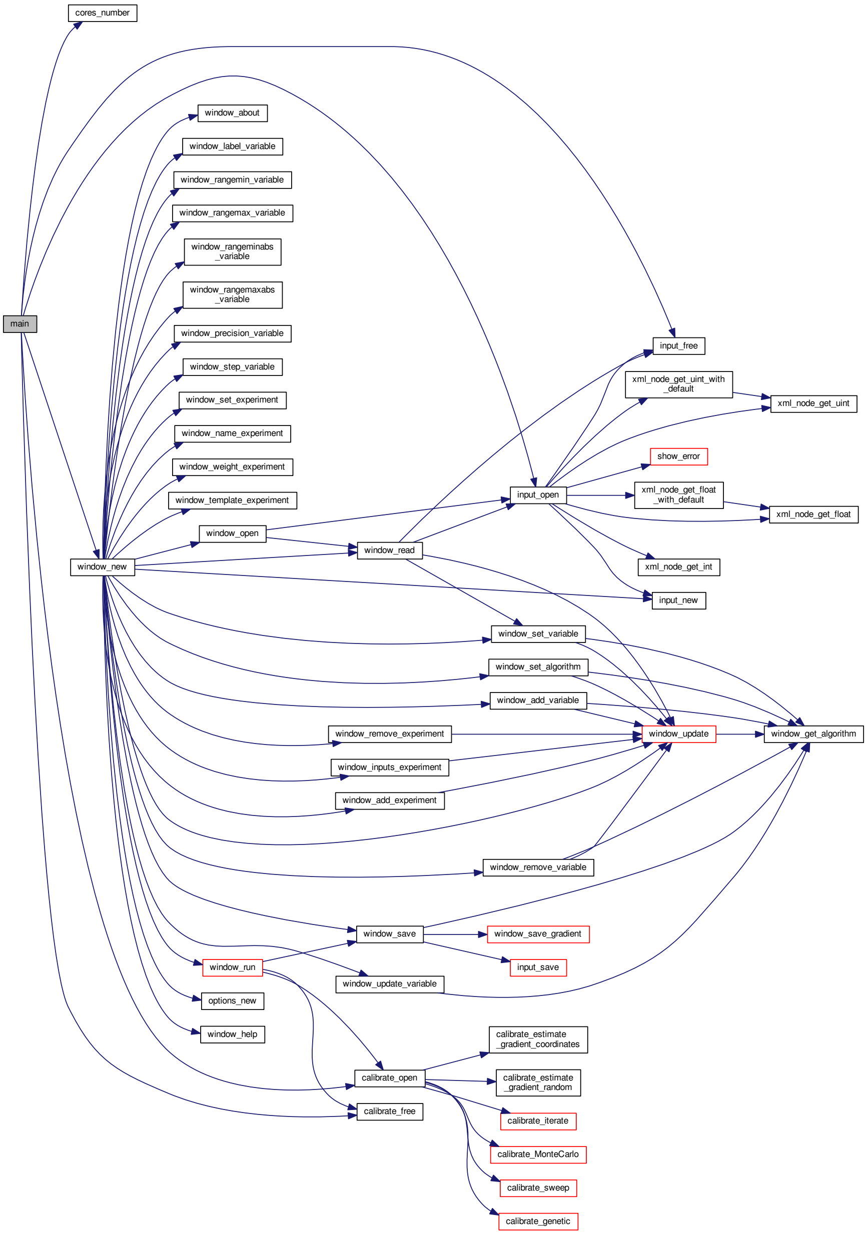
0 on success, >0 on error.

Definition at line 4896 of file mpcotool.c.

```
{
#if HAVE_GTK
  char *buffer;
```

```
#endif

  // Starting pseudo-random numbers generator
  calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
  calibrate->seed = DEFAULT_RANDOM_SEED;

  // Allowing spaces in the XML data file
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
  printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

#if HAVE_GTK

  // Getting threads number
  nthreads_gradient = nthreads = cores_number ();

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  gtk_init (&argn, &argc);

  // Opening the main window
  window_new ();
  gtk_main ();

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_free (window->application_directory);

#else

  // Checking syntax
  if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
    {
      printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
      return 1;
    }

  // Getting threads number
  if (argn == 2)
    nthreads_gradient = nthreads = cores_number ();
  else
    {
      nthreads_gradient = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);

  // Making calibration
  if (input_open (argc[argn - 1]))
    calibrate_open ();

  // Freeing memory
  calibrate_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (calibrate->rng);

  // Closing
  return 0;
}
```

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.

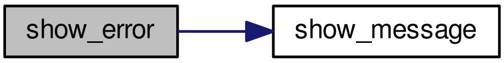
Parameters

msg	Error message.

Definition at line 256 of file mpcotool.c.

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

Here is the call graph for this function:



5.5.2.20 void show_message (char ∗ *title*, char ∗ *msg*, int *type*)

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file mpcotool.c.

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

5.5.2.21 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2932 of file mpcotool.c.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  for (i = 0; i < NALGORITHMS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}
```

5.5.2.22 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2955 of file mpcotool.c.

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_gradient: start\n");
#endif
  for (i = 0; i < NGRADIENTS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_gradient: %u\n", i);
  fprintf (stderr, "window_get_gradient: end\n");
#endif
  return i;
}
```

5.5.2.23 int window_read (char ∗ *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

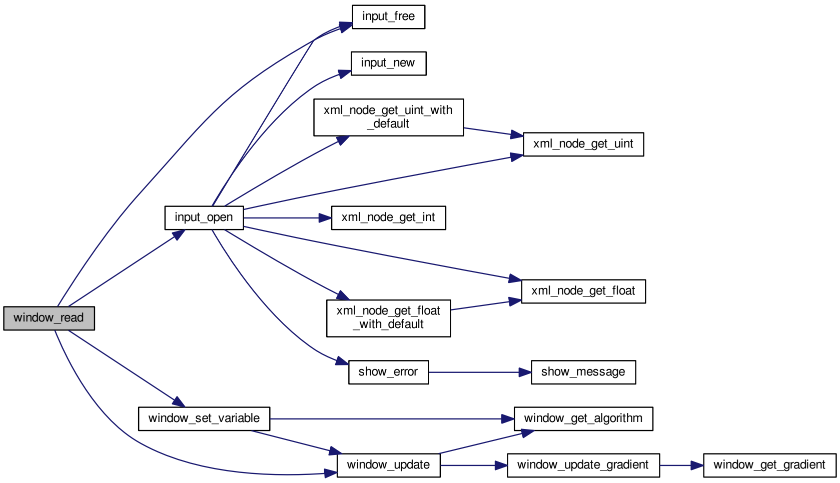
1 on succes, 0 on error.

Definition at line 4052 of file mpcotool.c.

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

```

  // Reading new input file
  input_free ();
  if (!input_open (filename))
    return 0;

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_gradient
                                [input->gradient_method]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->gradient_method)
            {
            case GRADIENT_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i]);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}
```

Here is the call graph for this function:



5.5.2.24 int window_save ( )

Function to save the input file.

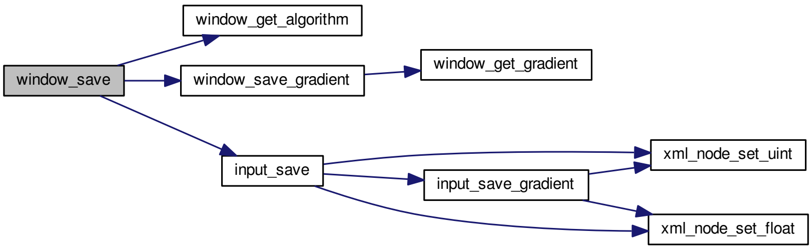
Returns

1 on OK, 0 on Cancel.

Definition at line 3010 of file mpcotool.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer;

#if DEBUG
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 gettext ("_Cancel"),
                                 GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_gradient ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_gradient ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}
```

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

data	Callback data (i-th input template).

Definition at line 3656 of file mpcotool.c.

```
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}
```

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

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 366 of file mpcotool.c.

```
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}
```

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

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

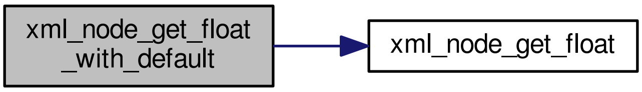
Returns

Floating point number value.

Definition at line 400 of file mpcotool.c.

```
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



5.5.2.28 int xml_node_get_int (xmlNode ∗ *node*, const xmlChar ∗ *prop*, int ∗ *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 274 of file mpcotool.c.

```
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

5.5.2.29 int xml_node_get_uint (xmlNode ∗ *node*, const xmlChar ∗ *prop*, int ∗ *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 305 of file mpcotool.c.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

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

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

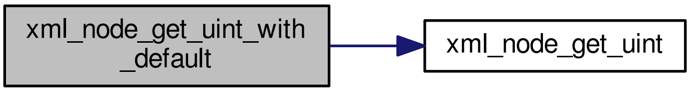
Returns

Unsigned integer number value.

Definition at line 339 of file mpcotool.c.

```
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

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

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 463 of file mpcotool.c.

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

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

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 425 of file mpcotool.c.

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

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

node	XML node.
prop	XML property.
value	Unsigned integer number value.

Definition at line 444 of file mpcotool.c.

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

5.5.3 Variable Documentation

5.5.3.1 const char∗ format[NPRECISIONS]

Initial value:

```
= {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
}
```

Array of C-strings with variable formats.

Definition at line 117 of file mpcotool.c.

5.5.3.2 const double precision[NPRECISIONS]

Initial value:

```
= {
  1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
  1e-13, 1e-14
}
```

Array of variable precisions.

Definition at line 122 of file mpcotool.c.

5.5.3.3 const xmlChar∗ template[MAX_NINPUTS]

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 111 of file mpcotool.c.

5.6 mpcotool.c

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #define _GNU_SOURCE
00037 #include "config.h"
00038 #include <stdio.h>
00039 #include <stdlib.h>
00040 #include <string.h>
00041 #include <math.h>
00042 #include <unistd.h>
00043 #include <locale.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
```

```
#elif (!__BSD_VISIBLE)
#include <alloca.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#include "genetic/genetic.h"
#include "mpcotool.h"
#if HAVE_GTK
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "interface.h"
#endif

#define DEBUG 0


#if HAVE_GTK
#define ERROR_TYPE GTK_MESSAGE_ERROR
#define INFO_TYPE GTK_MESSAGE_INFO
#else
#define ERROR_TYPE 0
#define INFO_TYPE 0
#endif
#ifdef G_OS_WIN32
#define INPUT_FILE "test-ga-win.xml"
#define RM "del"
#else
#define INPUT_FILE "test-ga.xml"
#define RM "rm"
#endif

int ntasks;
unsigned int nthreads;
unsigned int nthreads_gradient;
GMutex mutex[1];
void (*calibrate_algorithm) ();
double (*calibrate_estimate_gradient) (unsigned int variable,
                                       unsigned int estimate);
Input input[1];
Calibrate calibrate[1];

const xmlChar *result_name = (xmlChar *) "result";
const xmlChar *variables_name = (xmlChar *) "variables";

const xmlChar *template[MAX_NINPUTS] = {
  XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3, XML_TEMPLATE4,
  XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7, XML_TEMPLATE8
};

const char *format[NPRECISIONS] = {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
};

const double precision[NPRECISIONS] = {
  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
};

const char *logo[] = {
  "32 32 3 1",
  "     c None",
  ".    c #0000FF",
  "+    c #FF0000",
  "                                ",
  "                                ",
  "                                ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .     +++     .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "    +++     .     +++    +++    ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "    +++     .      .     +++    ",
  "     .      .      .      .     ",
  "     .     +++     .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .     +++     .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "                                ",
  "                                ",
  "                                "
};

/*
const char * logo[] = {
"32 32 3 1",
"     c #FFFFFFFFFFFF",
".    c #00000000FFFF",
"X    c #FFFF00000000",
"                                ",
"                                ",
"                                ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .     XXX     .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"    XXX     .     XXX    XXX    ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"    XXX     .      .     XXX    ",
"     .      .      .      .     ",
"     .     XXX     .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .     XXX     .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"                                ",
"                                ",
"                                "};
*/

#if HAVE_GTK
Options options[1];
Running running[1];
Window window[1];
#endif

void
show_message (char *title, char *msg, int type)
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *) gtk_message_dialog_new
    (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}

void
show_error (char *msg)
{
  show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
}

int
xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
                                unsigned int default_value, int *error_code)
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}

double
xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}

double
xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
                                 double default_value, int *error_code)
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}

void
xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%d", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%u", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%.14lg", value);
  xmlSetProp (node, prop, buffer);
}

void
input_new ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "input_new: start\n");
#endif
  input->nvariables = input->nexperiments = input->ninputs = input->nsteps = 0;
  input->simulator = input->evaluator = input->directory = input->name
    = input->result = input->variables = NULL;
  input->experiment = input->label = NULL;
  input->precision = input->nsweeps = input->nbits = NULL;
  input->rangemin = input->rangemax = input->rangeminabs = input->rangemaxabs
    = input->weight = input->step = NULL;
  for (i = 0; i < MAX_NINPUTS; ++i)
    input->template[i] = NULL;
#if DEBUG
  fprintf (stderr, "input_new: end\n");
#endif
}

void
input_free ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "input_free: start\n");
#endif
  g_free (input->name);
  g_free (input->directory);
  for (i = 0; i < input->nexperiments; ++i)
    {
      xmlFree (input->experiment[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlFree (input->template[j][i]);
      g_free (input->template[j]);
    }
  g_free (input->experiment);
  for (i = 0; i < input->ninputs; ++i)
    g_free (input->template[i]);
  for (i = 0; i < input->nvariables; ++i)
    xmlFree (input->label[i]);
  g_free (input->label);
  g_free (input->precision);
  g_free (input->rangemin);
  g_free (input->rangemax);
  g_free (input->rangeminabs);
  g_free (input->rangemaxabs);
  g_free (input->weight);
  g_free (input->step);
  g_free (input->nsweeps);
  g_free (input->nbits);
  xmlFree (input->evaluator);
  xmlFree (input->simulator);
  xmlFree (input->result);
  xmlFree (input->variables);
  input->nexperiments = input->ninputs = input->nvariables = input->
```

```
nsteps = 0;
#if DEBUG
  fprintf (stderr, "input_free: end\n");
#endif
}

int
input_open (char *filename)
{
  char buffer2[64];
  char *buffert[MAX_NINPUTS] =
    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
  xmlDoc *doc;
  xmlNode *node, *child;
  xmlChar *buffer;
  char *msg;
  int error_code;
  unsigned int i;

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_CALIBRATE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting results file names
  input->result = (char *) xmlGetProp (node, XML_RESULT);
  if (!input->result)
    input->result = (char *) xmlStrdup (result_name);
  input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
  if (!input->variables)
    input->variables = (char *) xmlStrdup (variables_name);

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
```

```
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
```

```
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node,
XML_NBEST, 1, &error_code);
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node,
XML_TOLERANCE, 0.,
                                           &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting gradient method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node,
XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->gradient_method = GRADIENT_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->gradient_method = GRADIENT_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the gradient");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child, XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
```

```
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No calibration experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables], input->
nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
          input->step[input->nvariables]
            = xml_node_get_float (child, XML_STEP, &error_code);
          if (error_code || input->step[input->nvariables] < 0.)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("bad step size"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      input->label = g_realloc
        (input->label, (1 + input->nvariables) * sizeof (char *));
      input->label[input->nvariables] = (char *) buffer;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      msg = gettext ("No calibration variables");
      goto exit_on_error;
    }
  buffer = NULL;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
  show_error (msg);
  input_free ();
#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}

void
calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
{
  unsigned int i;
  char buffer[32], value[32], *buffer2, *buffer3, *content;
  FILE *file;
  gsize length;
  GRegex *regex;

#if DEBUG
  fprintf (stderr, "calibrate_input: start\n");
#endif

  // Checking the file
  if (!template)
    goto calibrate_input_end;

  // Opening template
  content = g_mapped_file_get_contents (template);
  length = g_mapped_file_get_length (template);
#if DEBUG
  fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
           content);
#endif
  file = g_fopen (input, "w");

  // Parsing template
  for (i = 0; i < calibrate->nvariables; ++i)
    {
#if DEBUG
      fprintf (stderr, "calibrate_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             calibrate->label[i], 0, NULL);
#if DEBUG
          fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
#endif
```

```
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             calibrate->label[i], 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      snprintf (value, 32, format[calibrate->precision[i]],
                calibrate->value[simulation * calibrate->nvariables + i]);

#if DEBUG
      fprintf (stderr, "calibrate_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

calibrate_input_end:
#if DEBUG
  fprintf (stderr, "calibrate_input: end\n");
#endif
  return;
}

double
calibrate_parse (unsigned int simulation, unsigned int experiment)
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG
  fprintf (stderr, "calibrate_parse: start\n");
  fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
           experiment);
#endif

  // Opening input files
  for (i = 0; i < calibrate->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG
      fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      calibrate_input (simulation, &input[i][0],
                       calibrate->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG
  fprintf (stderr, "calibrate_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (calibrate->simulator);
  buffer3 = g_path_get_basename (calibrate->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
            input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG
  fprintf (stderr, "calibrate_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (calibrate->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (calibrate->evaluator);
      buffer3 = g_path_get_basename (calibrate->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, calibrate->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG
      fprintf (stderr, "calibrate_parse: %s\n", buffer);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG
  for (i = 0; i < calibrate->ninputs; ++i)
    {
      if (calibrate->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "calibrate_parse: end\n");
#endif

  // Returning the objective function
  return e * calibrate->weight[experiment];
}

void
calibrate_print ()
{
  unsigned int i;
  char buffer[512];
#if HAVE_MPI
  if (calibrate->mpi_rank)
    return;
#endif
  printf ("%s\n", gettext ("Best result"));
  fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
  printf ("error = %.15le\n", calibrate->error_old[0]);
  fprintf (calibrate->file_result, "error = %.15le\n", calibrate->error_old[0]);
  for (i = 0; i < calibrate->nvariables; ++i)
    {
      snprintf (buffer, 512, "%s = %s\n",
                calibrate->label[i], format[calibrate->precision[i]]);
      printf (buffer, calibrate->value_old[i]);
      fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
    }
  fflush (calibrate->file_result);
}

void
calibrate_save_variables (unsigned int simulation, double error)
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: start\n");
#endif
  for (i = 0; i < calibrate->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
      fprintf (calibrate->file_variables, buffer,
               calibrate->value[simulation * calibrate->nvariables + i]);
    }
  fprintf (calibrate->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: end\n");
```

```
#endif
}

void
calibrate_best (unsigned int simulation, double value)
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_best: start\n");
  fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
           calibrate->nsaveds, calibrate->nbest);
#endif
  if (calibrate->nsaveds < calibrate->nbest
      || value < calibrate->error_best[calibrate->nsaveds - 1])
    {
      if (calibrate->nsaveds < calibrate->nbest)
        ++calibrate->nsaveds;
      calibrate->error_best[calibrate->nsaveds - 1] = value;
      calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
      for (i = calibrate->nsaveds; --i;)
        {
          if (calibrate->error_best[i] < calibrate->error_best[i - 1])
            {
              j = calibrate->simulation_best[i];
              e = calibrate->error_best[i];
              calibrate->simulation_best[i] = calibrate->simulation_best[i - 1];
              calibrate->error_best[i] = calibrate->error_best[i - 1];
              calibrate->simulation_best[i - 1] = j;
              calibrate->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_best: end\n");
#endif
}

void
calibrate_sequential ()
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_sequential: start\n");
  fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
           calibrate->nstart, calibrate->nend);
#endif
  for (i = calibrate->nstart; i < calibrate->nend; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      calibrate_best (i, e);
      calibrate_save_variables (i, e);
#if DEBUG
      fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_sequential: end\n");
#endif
}

void *
calibrate_thread (ParallelData * data)
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
           calibrate->thread[thread], calibrate->thread[thread + 1]);
#endif
  for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

void
calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
                 double *error_best)
{
  unsigned int i, j, k, s[calibrate->nbest];
  double e[calibrate->nbest];
#if DEBUG
  fprintf (stderr, "calibrate_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == calibrate->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
          if (i == calibrate->nsaveds)
            break;
        }
      else if (calibrate->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < calibrate->nbest);
  calibrate->nsaveds = k;
  memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
  memcpy (calibrate->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "calibrate_merge: end\n");
#endif
}

#if HAVE_MPI
void
calibrate_synchronise ()
{
  unsigned int i, nsaveds, simulation_best[calibrate->nbest];
  double error_best[calibrate->nbest];
  MPI_Status mpi_stat;
#if DEBUG
  fprintf (stderr, "calibrate_synchronise: start\n");
#endif
  if (calibrate->mpi_rank == 0)
    {
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          calibrate_merge (nsaveds, simulation_best, error_best);
        }
    }
  else
    {
      MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD);
    }
#if DEBUG
  fprintf (stderr, "calibrate_synchronise: end\n");
#endif
}
#endif

void
calibrate_sweep ()
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG
  fprintf (stderr, "calibrate_sweep: start\n");
#endif
  for (i = 0; i < calibrate->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < calibrate->nvariables; ++j)
        {
          l = k % calibrate->nsweeps[j];
          k /= calibrate->nsweeps[j];
          e = calibrate->rangemin[j];
          if (calibrate->nsweeps[j] > 1)
            e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
              / (calibrate->nsweeps[j] - 1);
          calibrate->value[i * calibrate->nvariables + j] = e;
        }
    }
  calibrate->nsaveds = 0;
  if (nthreads <= 1)
    calibrate_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  calibrate_synchronise ();
#endif
#if DEBUG
  fprintf (stderr, "calibrate_sweep: end\n");
#endif
}

void
calibrate_MonteCarlo ()
{
  unsigned int i, j;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG
  fprintf (stderr, "calibrate_MonteCarlo: start\n");
#endif
  for (i = 0; i < calibrate->nsimulations; ++i)
    for (j = 0; j < calibrate->nvariables; ++j)
      calibrate->value[i * calibrate->nvariables + j]
        = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
        * (calibrate->rangemax[j] - calibrate->rangemin[j]);
  calibrate->nsaveds = 0;
  if (nthreads <= 1)
    calibrate_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  calibrate_synchronise ();
#endif
#if DEBUG
  fprintf (stderr, "calibrate_MonteCarlo: end\n");
#endif
}

void
calibrate_best_gradient (unsigned int simulation, double value)
{
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: start\n");
  fprintf (stderr,
           "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, calibrate->error_best[0]);
#endif
  if (value < calibrate->error_best[0])
    {
      calibrate->error_best[0] = value;
      calibrate->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: end\n");
#endif
}

void
calibrate_gradient_sequential (unsigned int simulation)
{
  unsigned int i, j, k;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_sequential: start\n");
  fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
           "nend_gradient=%u\n",
           calibrate->nstart_gradient, calibrate->nend_gradient);
#endif
  for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
    {
      k = simulation + i;
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (k, j);
      calibrate_best_gradient (k, e);
      calibrate_save_variables (k, e);
#if DEBUG
      fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient_sequential: end\n");
#endif
}

void *
calibrate_gradient_thread (ParallelData * data)
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
           thread,
           calibrate->thread_gradient[thread],
           calibrate->thread_gradient[thread + 1]);
#endif
  for (i = calibrate->thread_gradient[thread];
       i < calibrate->thread_gradient[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best_gradient (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

double
calibrate_estimate_gradient_random (unsigned int variable,
                                    unsigned int estimate)
{
  double x;
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
#endif
  x = calibrate->gradient[variable]
    + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
           variable, x);
  fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
#endif
  return x;
}

double
calibrate_estimate_gradient_coordinates (unsigned int variable,
                                         unsigned int estimate)
{
  double x;
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
#endif
  x = calibrate->gradient[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += calibrate->step[variable];
      else
        x -= calibrate->step[variable];
    }
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
           variable, x);
  fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
#endif
  return x;
}

void
calibrate_step_gradient (unsigned int simulation)
{
  GThread *thread[nthreads_gradient];
  ParallelData data[nthreads_gradient];
  unsigned int i, j, k, b;
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: start\n");
#endif
  for (i = 0; i < calibrate->nestimates; ++i)
    {
      k = (simulation + i) * calibrate->nvariables;
      b = calibrate->simulation_best[0] * calibrate->nvariables;
#if DEBUG
      fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
               simulation + i, calibrate->simulation_best[0]);
#endif
      for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
                   i, j, calibrate->value[b]);
#endif
          calibrate->value[k]
            = calibrate->value[b] + calibrate_estimate_gradient (j, i);
          calibrate->value[k] = fmin (fmax (calibrate->value[k],
                                            calibrate->rangeminabs[j]),
                                      calibrate->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
                   i, j, calibrate->value[k]);
#endif
        }
    }
  if (nthreads_gradient == 1)
    calibrate_gradient_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_gradient; ++i)
        {
          calibrate->thread_gradient[i]
            = simulation + calibrate->nstart_gradient
            + i * (calibrate->nend_gradient - calibrate->
nstart_gradient)
            / nthreads_gradient;
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: i=%u thread_gradient=%u\n",
                   i, calibrate->thread_gradient[i]);
#endif
        }
      for (i = 0; i < nthreads_gradient; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
        }
      for (i = 0; i < nthreads_gradient; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: end\n");
#endif
}

void
calibrate_gradient ()
{
  unsigned int i, j, k, b, s, adjust;
#if DEBUG
  fprintf (stderr, "calibrate_gradient: start\n");
#endif
  for (i = 0; i < calibrate->nvariables; ++i)
    calibrate->gradient[i] = 0.;
  b = calibrate->simulation_best[0] * calibrate->nvariables;
  s = calibrate->nsimulations;
  adjust = 1;
  for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
    {
#if DEBUG
      fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
               i, calibrate->simulation_best[0]);
#endif
      calibrate_step_gradient (s);
      k = calibrate->simulation_best[0] * calibrate->nvariables;
#if DEBUG
      fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
               i, calibrate->simulation_best[0]);
#endif
      if (k == b)
        {
          if (adjust)
            for (j = 0; j < calibrate->nvariables; ++j)
              calibrate->step[j] *= 0.5;
          for (j = 0; j < calibrate->nvariables; ++j)
            calibrate->gradient[j] = 0.;
          adjust = 1;
        }
      else
        {
          for (j = 0; j < calibrate->nvariables; ++j)
            {
#if DEBUG
              fprintf (stderr,
                       "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
                       j, calibrate->value[k + j], j, calibrate->value[b + j]);
#endif
              calibrate->gradient[j]
                = (1. - calibrate->relaxation) * calibrate->gradient[j]
                + calibrate->relaxation
                * (calibrate->value[k + j] - calibrate->value[b + j]);
#if DEBUG
              fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
                       j, calibrate->gradient[j]);
#endif
            }
          adjust = 0;
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient: end\n");
#endif
}

double
calibrate_genetic_objective (Entity * entity)
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: start\n");
#endif
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      calibrate->value[entity->id * calibrate->nvariables + j]
        = genetic_get_variable (entity, calibrate->genetic_variable + j);
    }
  for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
    objective += calibrate_parse (entity->id, j);
  g_mutex_lock (mutex);
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
      fprintf (calibrate->file_variables, buffer,
               genetic_get_variable (entity, calibrate->genetic_variable + j));
    }
  fprintf (calibrate->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: end\n");
#endif
  return objective;
}

void
calibrate_genetic ()
{
  char *best_genome;
  double best_objective, *best_variable;
#if DEBUG
  fprintf (stderr, "calibrate_genetic: start\n");
  fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
           nthreads);
  fprintf (stderr,
           "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
           calibrate->nvariables, calibrate->nsimulations,
           calibrate->niterations);
  fprintf (stderr,
           "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
           calibrate->mutation_ratio, calibrate->reproduction_ratio,
           calibrate->adaptation_ratio);
#endif
  genetic_algorithm_default (calibrate->nvariables,
                             calibrate->genetic_variable,
                             calibrate->nsimulations,
                             calibrate->niterations,
                             calibrate->mutation_ratio,
                             calibrate->reproduction_ratio,
                             calibrate->adaptation_ratio,
                             &calibrate_genetic_objective,
                             &best_genome, &best_variable, &best_objective);
#if DEBUG
  fprintf (stderr, "calibrate_genetic: the best\n");
#endif
  calibrate->error_old = (double *) g_malloc (sizeof (double));
  calibrate->value_old
    = (double *) g_malloc (calibrate->nvariables * sizeof (double));
  calibrate->error_old[0] = best_objective;
  memcpy (calibrate->value_old, best_variable,
          calibrate->nvariables * sizeof (double));
  g_free (best_genome);
  g_free (best_variable);
  calibrate_print ();
#if DEBUG
  fprintf (stderr, "calibrate_genetic: end\n");
#endif
}

void
calibrate_save_old ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "calibrate_save_old: start\n");
  fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
#endif
  memcpy (calibrate->error_old, calibrate->error_best,
          calibrate->nbest * sizeof (double));
  for (i = 0; i < calibrate->nbest; ++i)
    {
      j = calibrate->simulation_best[i];
#if DEBUG
      fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
#endif
      memcpy (calibrate->value_old + i * calibrate->nvariables,
              calibrate->value + j * calibrate->nvariables,
              calibrate->nvariables * sizeof (double));
    }
#if DEBUG
  for (i = 0; i < calibrate->nvariables; ++i)
    fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
             i, calibrate->value_old[i]);
  fprintf (stderr, "calibrate_save_old: end\n");
#endif
}

void
calibrate_merge_old ()
{
  unsigned int i, j, k;
  double v[calibrate->nbest * calibrate->nvariables], e[calibrate->nbest],
    *enew, *eold;
#if DEBUG
  fprintf (stderr, "calibrate_merge_old: start\n");
#endif
  enew = calibrate->error_best;
  eold = calibrate->error_old;
  i = j = k = 0;
  do
    {
      if (*enew < *eold)
        {
          memcpy (v + k * calibrate->nvariables,
                  calibrate->value
                  + calibrate->simulation_best[i] * calibrate->nvariables,
                  calibrate->nvariables * sizeof (double));
          e[k] = *enew;
          ++k;
          ++enew;
          ++i;
        }
      else
        {
          memcpy (v + k * calibrate->nvariables,
                  calibrate->value_old + j * calibrate->nvariables,
                  calibrate->nvariables * sizeof (double));
          e[k] = *eold;
          ++k;
          ++eold;
          ++j;
        }
    }
  while (k < calibrate->nbest);
  memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
  memcpy (calibrate->error_old, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "calibrate_merge_old: end\n");
#endif
}

void
calibrate_refine ()
{
  unsigned int i, j;
  double d;
#if HAVE_MPI
  MPI_Status mpi_stat;
#endif
#if DEBUG
  fprintf (stderr, "calibrate_refine: start\n");
#endif
#if HAVE_MPI
  if (!calibrate->mpi_rank)
    {
#endif
      for (j = 0; j < calibrate->nvariables; ++j)
    {
     calibrate->rangemin[j] = calibrate->rangemax[j]
      = calibrate->value_old[j];
    }
   for (i = 0; ++i < calibrate->nbest;)
    {
     for (j = 0; j < calibrate->nvariables; ++j)
      {
       calibrate->rangemin[j]
        = fmin (calibrate->rangemin[j],
            calibrate->value_old[i * calibrate->nvariables + j]);
       calibrate->rangemax[j]
        = fmax (calibrate->rangemax[j],
            calibrate->value_old[i * calibrate->nvariables + j]);
      }
    }
   for (j = 0; j < calibrate->nvariables; ++j)
    {
     d = calibrate->tolerance
      * (calibrate->rangemax[j] - calibrate->rangemin[j]);
     switch (calibrate->algorithm)
      {
      case ALGORITHM_MONTE_CARLO:
       d *= 0.5;
       break;
      default:
       if (calibrate->nsweeps[j] > 1)
        d /= calibrate->nsweeps[j] - 1;
       else
        d = 0.;
      }
     calibrate->rangemin[j] -= d;
     calibrate->rangemin[j]
      = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
     calibrate->rangemax[j] += d;
     calibrate->rangemax[j]
      = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
     printf ("%s min=%lg max=%lg\n", calibrate->label[j],
         calibrate->rangemin[j], calibrate->rangemax[j]);
     fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
          calibrate->label[j], calibrate->rangemin[j],
          calibrate->rangemax[j]);
    }
#if HAVE_MPI
   for (i = 1; i < ntasks; ++i)
    {
     MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
          1, MPI_COMM_WORLD);
     MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
          1, MPI_COMM_WORLD);
    }
  }
 else
  {
   MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
        MPI_COMM_WORLD, &mpi_stat);
   MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
        MPI_COMM_WORLD, &mpi_stat);
  }
#endif
#if DEBUG
 fprintf (stderr, "calibrate_refine: end\n");
#endif
}

void
calibrate_step ()
{
#if DEBUG
 fprintf (stderr, "calibrate_step: start\n");
#endif
 calibrate_algorithm ();
 if (calibrate->nsteps)
  calibrate_gradient ();
#if DEBUG
 fprintf (stderr, "calibrate_step: end\n");
#endif
}

void
calibrate_iterate ()
{
 unsigned int i;
#if DEBUG
 fprintf (stderr, "calibrate_iterate: start\n");
#endif
 calibrate->error_old
```

```
    = (double *) g_malloc (calibrate->nbest * sizeof (double));
  calibrate->value_old = (double *)
    g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
  calibrate_step ();
  calibrate_save_old ();
  calibrate_refine ();
  calibrate_print ();
  for (i = 1; i < calibrate->niterations; ++i)
    {
      calibrate_step ();
      calibrate_merge_old ();
      calibrate_refine ();
      calibrate_print ();
    }
#if DEBUG
  fprintf (stderr, "calibrate_iterate: end\n");
#endif
}

void
calibrate_free ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "calibrate_free: start\n");
#endif
  for (j = 0; j < calibrate->ninputs; ++j)
    {
      for (i = 0; i < calibrate->nexperiments; ++i)
        g_mapped_file_unref (calibrate->file[j][i]);
      g_free (calibrate->file[j]);
    }
  g_free (calibrate->error_old);
  g_free (calibrate->value_old);
  g_free (calibrate->value);
  g_free (calibrate->genetic_variable);
  g_free (calibrate->rangemax);
  g_free (calibrate->rangemin);
#if DEBUG
  fprintf (stderr, "calibrate_free: end\n");
#endif
}

void
calibrate_open ()
{
  GTimeZone *tz;
  GDateTime *t0, *t;
  unsigned int i, j, *nbits;

#if DEBUG
  char *buffer;
  fprintf (stderr, "calibrate_open: start\n");
#endif

  // Getting initial time
#if DEBUG
  fprintf (stderr, "calibrate_open: getting initial time\n");
#endif
  tz = g_time_zone_new_utc ();
  t0 = g_date_time_new_now (tz);

  // Obtaining and initing the pseudo-random numbers generator seed
#if DEBUG
  fprintf (stderr, "calibrate_open: getting initial seed\n");
#endif
  calibrate->seed = input->seed;
  gsl_rng_set (calibrate->rng, calibrate->seed);

  // Replacing the working directory
#if DEBUG
  fprintf (stderr, "calibrate_open: replacing the working directory\n");
#endif
  g_chdir (input->directory);

  // Getting results file names
  calibrate->result = input->result;
  calibrate->variables = input->variables;

  // Obtaining the simulator file
  calibrate->simulator = input->simulator;

  // Obtaining the evaluator file
  calibrate->evaluator = input->evaluator;

  // Reading the algorithm
  calibrate->algorithm = input->algorithm;
  switch (calibrate->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      calibrate_algorithm = calibrate_MonteCarlo;
      break;
    case ALGORITHM_SWEEP:
      calibrate_algorithm = calibrate_sweep;
      break;
    default:
      calibrate_algorithm = calibrate_genetic;
      calibrate->mutation_ratio = input->mutation_ratio;
      calibrate->reproduction_ratio = input->reproduction_ratio;
      calibrate->adaptation_ratio = input->adaptation_ratio;
    }
  calibrate->nvariables = input->nvariables;
  calibrate->nsimulations = input->nsimulations;
  calibrate->niterations = input->niterations;
  calibrate->nbest = input->nbest;
  calibrate->tolerance = input->tolerance;
  calibrate->nsteps = input->nsteps;
  calibrate->nestimates = 0;
  if (input->nsteps)
    {
      calibrate->gradient_method = input->gradient_method;
      calibrate->relaxation = input->relaxation;
      switch (input->gradient_method)
        {
        case GRADIENT_METHOD_COORDINATES:
          calibrate->nestimates = 2 * calibrate->nvariables;
          calibrate_estimate_gradient = calibrate_estimate_gradient_coordinates;
          break;
        default:
          calibrate->nestimates = input->nestimates;
          calibrate_estimate_gradient = calibrate_estimate_gradient_random;
        }
    }

#if DEBUG
  fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
#endif
  calibrate->simulation_best
    = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
  calibrate->error_best
    = (double *) alloca (calibrate->nbest * sizeof (double));

  // Reading the experimental data
#if DEBUG
  buffer = g_get_current_dir ();
  fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
  g_free (buffer);
#endif
  calibrate->nexperiments = input->nexperiments;
  calibrate->ninputs = input->ninputs;
  calibrate->experiment = input->experiment;
  calibrate->weight = input->weight;
  for (i = 0; i < input->ninputs; ++i)
    {
      calibrate->template[i] = input->template[i];
      calibrate->file[i]
        = g_malloc (input->nexperiments * sizeof (GMappedFile *));
    }
  for (i = 0; i < input->nexperiments; ++i)
    {
#if DEBUG
      fprintf (stderr, "calibrate_open: i=%u\n", i);
      fprintf (stderr, "calibrate_open: experiment=%s\n",
               calibrate->experiment[i]);
      fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
#endif
      for (j = 0; j < input->ninputs; ++j)
        {
#if DEBUG
          fprintf (stderr, "calibrate_open: template%u\n", j + 1);
          fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
                   i, j + 1, calibrate->template[j][i]);
#endif
          calibrate->file[j][i]
            = g_mapped_file_new (input->template[j][i], 0, NULL);
        }
    }

  // Reading the variables data
#if DEBUG
  fprintf (stderr, "calibrate_open: reading variables\n");
```

```
#endif
  calibrate->label = input->label;
  j = input->nvariables * sizeof (double);
  calibrate->rangemin = (double *) g_malloc (j);
  calibrate->rangemax = (double *) g_malloc (j);
  memcpy (calibrate->rangemin, input->rangemin, j);
  memcpy (calibrate->rangemax, input->rangemax, j);
  calibrate->rangeminabs = input->rangeminabs;
  calibrate->rangemaxabs = input->rangemaxabs;
  calibrate->precision = input->precision;
  calibrate->nsweeps = input->nsweeps;
  calibrate->step = input->step;
  nbits = input->nbits;
  if (input->algorithm == ALGORITHM_SWEEP)
    {
      calibrate->nsimulations = 1;
      for (i = 0; i < input->nvariables; ++i)
        {
          if (input->algorithm == ALGORITHM_SWEEP)
            {
              calibrate->nsimulations *= input->nsweeps[i];
#if DEBUG
              fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
                       calibrate->nsweeps[i], calibrate->nsimulations);
#endif
            }
        }
    }
  if (calibrate->nsteps)
    calibrate->gradient
      = (double *) alloca (calibrate->nvariables * sizeof (double));

  // Allocating values
#if DEBUG
  fprintf (stderr, "calibrate_open: allocating variables\n");
  fprintf (stderr, "calibrate_open: nvariables=%u\n", calibrate->nvariables);
#endif
  calibrate->genetic_variable = NULL;
  if (calibrate->algorithm == ALGORITHM_GENETIC)
    {
      calibrate->genetic_variable = (GeneticVariable *)
        g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
      for (i = 0; i < calibrate->nvariables; ++i)
        {
#if DEBUG
          fprintf (stderr, "calibrate_open: i=%u min=%lg max=%lg nbits=%u\n",
                   i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
#endif
          calibrate->genetic_variable[i].minimum = calibrate->
rangemin[i];
          calibrate->genetic_variable[i].maximum = calibrate->
rangemax[i];
          calibrate->genetic_variable[i].nbits = nbits[i];
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_open: nvariables=%u nsimulations=%u\n",
           calibrate->nvariables, calibrate->nsimulations);
#endif
  calibrate->value = (double *)
    g_malloc ((calibrate->nsimulations
               + calibrate->nestimates * calibrate->nsteps)
              * calibrate->nvariables * sizeof (double));

  // Calculating simulations to perform on each task
#if HAVE_MPI
#if DEBUG
  fprintf (stderr, "calibrate_open: rank=%u ntasks=%u\n",
           calibrate->mpi_rank, ntasks);
#endif
  calibrate->nstart = calibrate->mpi_rank * calibrate->
nsimulations / ntasks;
  calibrate->nend
    = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
ntasks;
  if (calibrate->nsteps)
    {
      calibrate->nstart_gradient
        = calibrate->mpi_rank * calibrate->nestimates / ntasks;
      calibrate->nend_gradient
        = (1 + calibrate->mpi_rank) * calibrate->nestimates /
ntasks;
    }
#else
  calibrate->nstart = 0;
  calibrate->nend = calibrate->nsimulations;
  if (calibrate->nsteps)
```

```
    {
      calibrate->nstart_gradient = 0;
      calibrate->nend_gradient = calibrate->nestimates;
    }
#endif
#if DEBUG
  fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
           calibrate->nend);
#endif

  // Calculating simulations to perform for each thread
  calibrate->thread
    = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
  for (i = 0; i <= nthreads; ++i)
    {
      calibrate->thread[i] = calibrate->nstart
        + i * (calibrate->nend - calibrate->nstart) / nthreads;
#if DEBUG
      fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
               calibrate->thread[i]);
#endif
    }
  if (calibrate->nsteps)
    calibrate->thread_gradient = (unsigned int *)
      alloca ((1 + nthreads_gradient) * sizeof (unsigned int));

  // Opening result files
  calibrate->file_result = g_fopen (calibrate->result, "w");
  calibrate->file_variables = g_fopen (calibrate->variables, "w");

  // Performing the algorithm
  switch (calibrate->algorithm)
    {
      // Genetic algorithm
    case ALGORITHM_GENETIC:
      calibrate_genetic ();
      break;

      // Iterative algorithm
    default:
      calibrate_iterate ();
    }

  // Getting calculation time
  t = g_date_time_new_now (tz);
  calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
  g_date_time_unref (t);
  g_date_time_unref (t0);
  g_time_zone_unref (tz);
  printf ("%s = %.6lg s\n",
          gettext ("Calculation time"), calibrate->calculation_time);
  fprintf (calibrate->file_result, "%s = %.6lg s\n",
           gettext ("Calculation time"), calibrate->calculation_time);

  // Closing result files
  fclose (calibrate->file_variables);
  fclose (calibrate->file_result);

#if DEBUG
  fprintf (stderr, "calibrate_open: end\n");
#endif
}

#if HAVE_GTK

void
input_save_gradient (xmlNode * node)
{
#if DEBUG
  fprintf (stderr, "input_save_gradient: start\n");
#endif
  if (input->nsteps)
    {
      xml_node_set_uint (node, XML_NSTEPS, input->
nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        xml_node_set_float (node, XML_RELAXATION, input->
relaxation);
      switch (input->gradient_method)
        {
        case GRADIENT_METHOD_COORDINATES:
          xmlSetProp (node, XML_GRADIENT_METHOD,
XML_COORDINATES);
          break;
        default:
          xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
          xml_node_set_uint (node, XML_NESTIMATES, input->
nestimates);
        }
    }
#if DEBUG
  fprintf (stderr, "input_save_gradient: end\n");
#endif
}

void
input_save (char *filename)
{
  unsigned int i, j;
  char *buffer;
  xmlDoc *doc;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);
  file = g_file_new_for_path (input->directory);

  // Opening the input file
  doc = xmlNewDoc ((const xmlChar *) "1.0");

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp ((const xmlChar *) input->result, result_name))
    xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
  if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
    xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
      if (input->weight[i] != 1.)
        xml_node_set_float (child, XML_WEIGHT, input->
weight[i]);
      for (j = 0; j < input->ninputs; ++j)
        xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
      xml_node_set_float (child, XML_MINIMUM, input->
rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
rangeminabs[i]);
      xml_node_set_float (child, XML_MAXIMUM, input->
rangemax[i]);
      if (input->rangemaxabs[i] != G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
rangemaxabs[i]);
      if (input->precision[i] != DEFAULT_PRECISION)
        xml_node_set_uint (child, XML_PRECISION, input->
precision[i]);
      if (input->algorithm == ALGORITHM_SWEEP)
        xml_node_set_uint (child, XML_NSWEEPS, input->
nsweeps[i]);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, XML_NBITS, input->
nbits[i]);
      if (input->nsteps)
        xml_node_set_float (child, XML_STEP, input->
step[i]);
    }

  // Saving the XML file
  xmlSaveFormatFile (filename, doc, 1);

  // Freeing memory
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_save: end\n");
#endif
}

void
options_new ()
{
#if DEBUG
  fprintf (stderr, "options_new: start\n");
#endif
  options->label_seed = (GtkLabel *)
    gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
  options->spin_seed = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_seed),
     gettext ("Seed to init the pseudo-random numbers generator"));
  gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
  options->label_threads = (GtkLabel *)
    gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
  options->spin_threads
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_threads),
     gettext ("Number of threads to perform the calibration/optimization for "
              "the stochastic algorithm"));
  gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
  options->label_gradient = (GtkLabel *)
```

```
    gtk_label_new (gettext ("Threads number for the gradient based method"));
  options->spin_gradient
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_gradient),
     gettext ("Number of threads to perform the calibration/optimization for "
              "the gradient based method"));
  gtk_spin_button_set_value (options->spin_gradient,
                             (gdouble) nthreads_gradient);
  options->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
                   0, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
                   1, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
                   0, 2, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
                   1, 2, 1, 1);
  gtk_widget_show_all (GTK_WIDGET (options->grid));
  options->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (gettext ("Options"),
                                 window->window,
                                 GTK_DIALOG_MODAL,
                                 gettext ("_OK"), GTK_RESPONSE_OK,
                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
                                 NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
     GTK_WIDGET (options->grid));
  if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
    {
      input->seed
        = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
      nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
      nthreads_gradient
        = gtk_spin_button_get_value_as_int (options->spin_gradient);
    }
  gtk_widget_destroy (GTK_WIDGET (options->dialog));
#if DEBUG
  fprintf (stderr, "options_new: end\n");
#endif
}

void
running_new ()
{
#if DEBUG
  fprintf (stderr, "running_new: start\n");
#endif
  running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
  running->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (gettext ("Calculating"),
                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
     GTK_WIDGET (running->label));
  gtk_widget_show_all (GTK_WIDGET (running->dialog));
#if DEBUG
  fprintf (stderr, "running_new: end\n");
#endif
}

int
window_get_algorithm ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  for (i = 0; i < NALGORITHMS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}

int
window_get_gradient ()
{
  unsigned int i;
#if DEBUG
```

```
  fprintf (stderr, "window_get_gradient: start\n");
#endif
  for (i = 0; i < NGRADIENTS; ++i)
    if (gtk_toggle_button_get_active
        (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
      break;
#if DEBUG
  fprintf (stderr, "window_get_gradient: %u\n", i);
  fprintf (stderr, "window_get_gradient: end\n");
#endif
  return i;
}

void
window_save_gradient ()
{
#if DEBUG
  fprintf (stderr, "window_save_gradient: start\n");
#endif
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
    {
      input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
      input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
      switch (window_get_gradient ())
        {
        case GRADIENT_METHOD_COORDINATES:
          input->gradient_method = GRADIENT_METHOD_COORDINATES;
          break;
        default:
          input->gradient_method = GRADIENT_METHOD_RANDOM;
          input->nestimates
            = gtk_spin_button_get_value_as_int (window->spin_estimates);
        }
    }
  else
    input->nsteps = 0;
#if DEBUG
  fprintf (stderr, "window_save_gradient: end\n");
#endif
}

int
window_save ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer;

#if DEBUG
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 gettext ("_Cancel"),
                                 GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      input->result
```

```
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_result));
      input->variables
        = (char *) xmlStrdup ((const xmlChar *)
                              gtk_entry_get_text (window->entry_variables));

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
          window_save_gradient ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
          window_save_gradient ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}

void
window_run ()
{
  unsigned int i;
  char *msg, *msg2, buffer[64], buffer2[64];
#if DEBUG
  fprintf (stderr, "window_run: start\n");
#endif
  if (!window_save ())
    {
#if DEBUG
      fprintf (stderr, "window_run: end\n");
#endif
      return;
    }
  running_new ();
  while (gtk_events_pending ())
    gtk_main_iteration ();
  calibrate_open ();
  gtk_widget_destroy (GTK_WIDGET (running->dialog));
  snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
  msg2 = g_strdup (buffer);
  for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
    {
      snprintf (buffer, 64, "%s = %s\n",
                calibrate->label[i], format[calibrate->precision[i]]);
      snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
      msg = g_strconcat (msg2, buffer2, NULL);
      g_free (msg2);
    }
  snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
            calibrate->calculation_time);
  msg = g_strconcat (msg2, buffer, NULL);
  g_free (msg2);
  show_message (gettext ("Best result"), msg, INFO_TYPE);
  g_free (msg);
  calibrate_free ();
#if DEBUG
  fprintf (stderr, "window_run: end\n");
#endif
}

void
window_help ()
{
  char *buffer, *buffer2;
#if DEBUG
  fprintf (stderr, "window_help: start\n");
#endif
  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
                              gettext ("user-manual.pdf"), NULL);
  buffer = g_filename_to_uri (buffer2, NULL, NULL);
  g_free (buffer2);
  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
#if DEBUG
  fprintf (stderr, "window_help: uri=%s\n", buffer);
#endif
  g_free (buffer);
#if DEBUG
  fprintf (stderr, "window_help: end\n");
#endif
}

void
window_about ()
{
  static const gchar *authors[] = {
    "Javier Burguete Tolosa <jburguete@eead.csic.es>",
    "Borja Latorre Garcés <borja.latorre@csic.es>",
    NULL
  };
#if DEBUG
  fprintf (stderr, "window_about: start\n");
#endif
  gtk_show_about_dialog
    (window->window,
     "program_name", "MPCOTool",
     "comments",
     gettext ("A software to perform calibrations/optimizations of empirical "
              "parameters"),
     "authors", authors,
     "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
     "version", "1.2.4",
     "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
     "logo", window->logo,
     "website", "https://github.com/jburguete/mpcotool",
     "license-type", GTK_LICENSE_BSD, NULL);
#if DEBUG
  fprintf (stderr, "window_about: end\n");
#endif
}

void
window_update_gradient ()
{
#if DEBUG
  fprintf (stderr, "window_update_gradient: start\n");
#endif
  gtk_widget_show (GTK_WIDGET (window->check_gradient));
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
    {
      gtk_widget_show (GTK_WIDGET (window->grid_gradient));
      gtk_widget_show (GTK_WIDGET (window->label_step));
      gtk_widget_show (GTK_WIDGET (window->spin_step));
    }
  switch (window_get_gradient ())
    {
```

```
    case GRADIENT_METHOD_COORDINATES:
      gtk_widget_hide (GTK_WIDGET (window->label_estimates));
      gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_estimates));
      gtk_widget_show (GTK_WIDGET (window->spin_estimates));
    }
#if DEBUG
  fprintf (stderr, "window_update_gradient: end\n");
#endif
}

void
window_update ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_update: start\n");
#endif
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_evaluator),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                   (window->check_evaluator)));
  gtk_widget_hide (GTK_WIDGET (window->label_simulations));
  gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
  gtk_widget_hide (GTK_WIDGET (window->label_iterations));
  gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
  gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->label_bests));
  gtk_widget_hide (GTK_WIDGET (window->spin_bests));
  gtk_widget_hide (GTK_WIDGET (window->label_population));
  gtk_widget_hide (GTK_WIDGET (window->spin_population));
  gtk_widget_hide (GTK_WIDGET (window->label_generations));
  gtk_widget_hide (GTK_WIDGET (window->spin_generations));
  gtk_widget_hide (GTK_WIDGET (window->label_mutation));
  gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
  gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->label_bits));
  gtk_widget_hide (GTK_WIDGET (window->spin_bits));
  gtk_widget_hide (GTK_WIDGET (window->check_gradient));
  gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
  gtk_widget_hide (GTK_WIDGET (window->label_step));
  gtk_widget_hide (GTK_WIDGET (window->spin_step));
  i = gtk_spin_button_get_value_as_int (window->spin_iterations);
  switch (window_get_algorithm ())
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_widget_show (GTK_WIDGET (window->label_simulations));
      gtk_widget_show (GTK_WIDGET (window->spin_simulations));
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      window_update_gradient ();
      break;
    case ALGORITHM_SWEEP:
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      gtk_widget_show (GTK_WIDGET (window->label_sweeps));
      gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
      gtk_widget_show (GTK_WIDGET (window->check_gradient));
      window_update_gradient ();
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_population));
      gtk_widget_show (GTK_WIDGET (window->spin_population));
      gtk_widget_show (GTK_WIDGET (window->label_generations));
      gtk_widget_show (GTK_WIDGET (window->spin_generations));
      gtk_widget_show (GTK_WIDGET (window->label_mutation));
      gtk_widget_show (GTK_WIDGET (window->spin_mutation));
      gtk_widget_show (GTK_WIDGET (window->label_reproduction));
      gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
      gtk_widget_show (GTK_WIDGET (window->label_adaptation));
      gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
      gtk_widget_show (GTK_WIDGET (window->label_bits));
      gtk_widget_show (GTK_WIDGET (window->spin_bits));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
  for (i = 0; i < input->ninputs; ++i)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->id_input[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
      g_signal_handler_unblock
        (window->button_template[i], window->id_input[i]);
      g_signal_handler_unblock
        (window->check_template[i], window->id_template[i]);
    }
  if (i > 0)
    {
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i - 1]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
    }
  if (i < MAX_NINPUTS)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i]));
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->id_input[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
      g_signal_handler_unblock
        (window->button_template[i], window->id_input[i]);
      g_signal_handler_unblock
        (window->check_template[i], window->id_template[i]);
    }
  while (++i < MAX_NINPUTS)
    {
      gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
      gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_minabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_maxabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
#if DEBUG
  fprintf (stderr, "window_update: end\n");
#endif
}

void
window_set_algorithm ()
{
  int i;
#if DEBUG
  fprintf (stderr, "window_set_algorithm: start\n");
#endif
  i = window_get_algorithm ();
  switch (i)
    {
    case ALGORITHM_SWEEP:
      input->nsweeps = (unsigned int *) g_realloc
        (input->nsweeps, input->nvariables * sizeof (unsigned int));
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->nsweeps[i]);
      break;
    case ALGORITHM_GENETIC:
      input->nbits = (unsigned int *) g_realloc
        (input->nbits, input->nvariables * sizeof (unsigned int));
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_bits, (gdouble) input->nbits[i]);
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_set_algorithm: end\n");
#endif
}

void
window_set_experiment ()
{
  unsigned int i, j;
  char *buffer1, *buffer2;
#if DEBUG
  fprintf (stderr, "window_set_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
  buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
  buffer2 = g_build_filename (input->directory, buffer1, NULL);
  g_free (buffer1);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_file_chooser_set_filename
    (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_free (buffer2);
  for (j = 0; j < input->ninputs; ++j)
    {
      g_signal_handler_block (window->button_template[j], window->id_input[j]);
      buffer2
        = g_build_filename (input->directory, input->template[j][i], NULL);
      gtk_file_chooser_set_filename
        (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
      g_free (buffer2);
      g_signal_handler_unblock
        (window->button_template[j], window->id_input[j]);
    }
#if DEBUG
  fprintf (stderr, "window_set_experiment: end\n");
#endif
}

void
window_remove_experiment ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_remove_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  xmlFree (input->experiment[i]);
  --input->nexperiments;
  for (j = i; j < input->nexperiments; ++j)
    {
      input->experiment[j] = input->experiment[j + 1];
      input->weight[j] = input->weight[j + 1];
    }
  j = input->nexperiments - 1;
  if (i > j)
    i = j;
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
```

```
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_remove_experiment: end\n");
#endif
}

void
window_add_experiment ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_add_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_insert_text
    (window->combo_experiment, i, input->experiment[i]);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  input->experiment = (char **) g_realloc
    (input->experiment, (input->nexperiments + 1) * sizeof (char *));
  input->weight = (double *) g_realloc
    (input->weight, (input->nexperiments + 1) * sizeof (double));
  for (j = input->nexperiments - 1; j > i; --j)
    {
      input->experiment[j + 1] = input->experiment[j];
      input->weight[j + 1] = input->weight[j];
    }
  input->experiment[j + 1]
    = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
  input->weight[j + 1] = input->weight[j];
  ++input->nexperiments;
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_add_experiment: end\n");
#endif
}

void
window_name_experiment ()
{
  unsigned int i;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_name_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_name_experiment: end\n");
#endif
}

void
```

```
window_weight_experiment ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_weight_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
#if DEBUG
  fprintf (stderr, "window_weight_experiment: end\n");
#endif
}

void
window_inputs_experiment ()
{
  unsigned int j;
#if DEBUG
  fprintf (stderr, "window_inputs_experiment: start\n");
#endif
  j = input->ninputs - 1;
  if (j
      && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                        (window->check_template[j])))
    --input->ninputs;
  if (input->ninputs < MAX_NINPUTS
      && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                       (window->check_template[j])))
    {
      ++input->ninputs;
      for (j = 0; j < input->ninputs; ++j)
        {
          input->template[j] = (char **)
            g_realloc (input->template[j], input->nvariables * sizeof (char *));
        }
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_inputs_experiment: end\n");
#endif
}

void
window_template_experiment (void *data)
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}

void
window_set_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_set_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_entry_set_text (window->entry_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
  gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
  if (input->rangeminabs[i] != -G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_minabs, input->rangeminabs[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
    }
  if (input->rangemaxabs[i] != G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_maxabs, input->rangemaxabs[i]);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
    }
  gtk_spin_button_set_value (window->spin_precision, input->precision[i]);
  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
  if (input->nsteps)
    gtk_spin_button_set_value (window->spin_step, input->step[i]);
#if DEBUG
  fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
           input->precision[i]);
#endif
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->nsweeps[i]);
#if DEBUG
      fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
               input->nsweeps[i]);
#endif
      break;
    case ALGORITHM_GENETIC:
      gtk_spin_button_set_value (window->spin_bits, (gdouble) input->nbits[i]);
#if DEBUG
      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
               input->nbits[i]);
#endif
      break;
    }
  window_update ();
#if DEBUG
  fprintf (stderr, "window_set_variable: end\n");
#endif
}

void
window_remove_variable ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_remove_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  xmlFree (input->label[i]);
  --input->nvariables;
  for (j = i; j < input->nvariables; ++j)
    {
      input->label[j] = input->label[j + 1];
      input->rangemin[j] = input->rangemin[j + 1];
      input->rangemax[j] = input->rangemax[j + 1];
      input->rangeminabs[j] = input->rangeminabs[j + 1];
      input->rangemaxabs[j] = input->rangemaxabs[j + 1];
      input->precision[j] = input->precision[j + 1];
      input->step[j] = input->step[j + 1];
      switch (window_get_algorithm ())
        {
        case ALGORITHM_SWEEP:
          input->nsweeps[j] = input->nsweeps[j + 1];
          break;
        case ALGORITHM_GENETIC:
          input->nbits[j] = input->nbits[j + 1];
        }
```

```c
    }
  j = input->nvariables - 1;
  if (i > j)
    i = j;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_remove_variable: end\n");
#endif
}

void
window_add_variable ()
{
  unsigned int i, j;
#if DEBUG
  fprintf (stderr, "window_add_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_insert_text (window->combo_variable, i, input->label[i]);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  input->label = (char **) g_realloc
    (input->label, (input->nvariables + 1) * sizeof (char *));
  input->rangemin = (double *) g_realloc
    (input->rangemin, (input->nvariables + 1) * sizeof (double));
  input->rangemax = (double *) g_realloc
    (input->rangemax, (input->nvariables + 1) * sizeof (double));
  input->rangeminabs = (double *) g_realloc
    (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
  input->rangemaxabs = (double *) g_realloc
    (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
  input->precision = (unsigned int *) g_realloc
    (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
  input->step = (double *) g_realloc
    (input->step, (input->nvariables + 1) * sizeof (double));
  for (j = input->nvariables - 1; j > i; --j)
    {
      input->label[j + 1] = input->label[j];
      input->rangemin[j + 1] = input->rangemin[j];
      input->rangemax[j + 1] = input->rangemax[j];
      input->rangeminabs[j + 1] = input->rangeminabs[j];
      input->rangemaxabs[j + 1] = input->rangemaxabs[j];
      input->precision[j + 1] = input->precision[j];
      input->step[j + 1] = input->step[j];
    }
  input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
  input->rangemin[j + 1] = input->rangemin[j];
  input->rangemax[j + 1] = input->rangemax[j];
  input->rangeminabs[j + 1] = input->rangeminabs[j];
  input->rangemaxabs[j + 1] = input->rangemaxabs[j];
  input->precision[j + 1] = input->precision[j];
  input->step[j + 1] = input->step[j];
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      input->nsweeps = (unsigned int *) g_realloc
        (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
      for (j = input->nvariables - 1; j > i; --j)
        input->nsweeps[j + 1] = input->nsweeps[j];
      input->nsweeps[j + 1] = input->nsweeps[j];
      break;
    case ALGORITHM_GENETIC:
      input->nbits = (unsigned int *) g_realloc
        (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
      for (j = input->nvariables - 1; j > i; --j)
        input->nbits[j + 1] = input->nbits[j];
      input->nbits[j + 1] = input->nbits[j];
    }
  ++input->nvariables;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG
  fprintf (stderr, "window_add_variable: end\n");
#endif
}
```

```

void
window_label_variable ()
{
  unsigned int i;
  const char *buffer;
#if DEBUG
  fprintf (stderr, "window_label_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  buffer = gtk_entry_get_text (window->entry_variable);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
#if DEBUG
  fprintf (stderr, "window_label_variable: end\n");
#endif
}

void
window_precision_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_precision_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->precision[i]
    = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
  gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
  gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
#if DEBUG
  fprintf (stderr, "window_precision_variable: end\n");
#endif
}

void
window_rangemin_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemin_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
#if DEBUG
  fprintf (stderr, "window_rangemin_variable: end\n");
#endif
}

void
window_rangemax_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemax_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
#if DEBUG
  fprintf (stderr, "window_rangemax_variable: end\n");
#endif
}

void
window_rangeminabs_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangeminabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangeminabs[i] = gtk_spin_button_get_value (window->spin_minabs);
#if DEBUG
  fprintf (stderr, "window_rangeminabs_variable: end\n");
#endif
}

void
window_rangemaxabs_variable ()
```

```
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_rangemaxabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->rangemaxabs[i] = gtk_spin_button_get_value (window->spin_maxabs);
#if DEBUG
  fprintf (stderr, "window_rangemaxabs_variable: end\n");
#endif
}

void
window_step_variable ()
{
  unsigned int i;
#if DEBUG
  fprintf (stderr, "window_step_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->step[i] = gtk_spin_button_get_value (window->spin_step);
#if DEBUG
  fprintf (stderr, "window_step_variable: end\n");
#endif
}

void
window_update_variable ()
{
  int i;
#if DEBUG
  fprintf (stderr, "window_update_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  if (i < 0)
    i = 0;
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
      input->nsweeps[i]
        = gtk_spin_button_get_value_as_int (window->spin_sweeps);
#if DEBUG
      fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
               input->nsweeps[i]);
#endif
      break;
    case ALGORITHM_GENETIC:
      input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
#if DEBUG
      fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
               input->nbits[i]);
#endif
    }
#if DEBUG
  fprintf (stderr, "window_update_variable: end\n");
#endif
}

int
window_read (char *filename)
{
  unsigned int i;
  char *buffer;
#if DEBUG
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  if (!input_open (filename))
    return 0;

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_gradient
                                [input->gradient_method]), TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->gradient_method)
            {
            case GRADIENT_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i]);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable, input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}

void
window_open ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer, *directory, *name;

#if DEBUG
  fprintf (stderr, "window_open: start\n");
#endif

  // Saving a backup of the current input file
  directory = g_strdup (input->directory);
  name = g_strdup (input->name);

  // Opening dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (gettext ("Open input file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_OPEN,
                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK saving
  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Traying to open the input file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      if (!window_read (buffer))
        {
#if DEBUG
          fprintf (stderr, "window_open: error reading input file\n");
#endif
          g_free (buffer);

          // Reading backup file on error
          buffer = g_build_filename (directory, name, NULL);
          if (!input_open (buffer))
            {

              // Closing on backup file reading error
#if DEBUG
              fprintf (stderr, "window_read: error reading backup file\n");
#endif
              g_free (buffer);
              break;
            }
          g_free (buffer);
        }
      else
        {
          g_free (buffer);
          break;
        }
    }

  // Freeing and closing
  g_free (name);
  g_free (directory);
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
  fprintf (stderr, "window_open: end\n");
#endif
}

void
window_new ()
{
  unsigned int i;
  char *buffer, *buffer2, buffer3[64];
  char *label_algorithm[NALGORITHMS] = {
    "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
  };
  char *tip_algorithm[NALGORITHMS] = {
    gettext ("Monte-Carlo brute force algorithm"),
    gettext ("Sweep brute force algorithm"),
    gettext ("Genetic algorithm")
  };
  char *label_gradient[NGRADIENTS] = {
    gettext ("_Coordinates descent"), gettext ("_Random")
  };
  char *tip_gradient[NGRADIENTS] = {
    gettext ("Coordinates descent gradient estimate method"),
    gettext ("Random gradient estimate method")
  };

#if DEBUG
  fprintf (stderr, "window_new: start\n");
#endif

  // Creating the window
  window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);

  // Finish when closing the window
  g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);

  // Setting the window title
  gtk_window_set_title (window->window, "MPCOTool");

  // Creating the open button
  window->button_open = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-open",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Open"));
  g_signal_connect (window->button_open, "clicked", window_open, NULL);

  // Creating the save button
  window->button_save = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-save",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Save"));
  g_signal_connect (window->button_save, "clicked", (void (*)) window_save,
                    NULL);

  // Creating the run button
  window->button_run = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("system-run",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Run"));
  g_signal_connect (window->button_run, "clicked", window_run, NULL);

  // Creating the options button
  window->button_options = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("preferences-system",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Options"));
  g_signal_connect (window->button_options, "clicked", options_new, NULL);

  // Creating the help button
  window->button_help = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-browser",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Help"));
  g_signal_connect (window->button_help, "clicked", window_help, NULL);

  // Creating the about button
  window->button_about = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-about",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("About"));
  g_signal_connect (window->button_about, "clicked", window_about, NULL);

  // Creating the exit button
  window->button_exit = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("application-exit",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
     gettext ("Exit"));
  g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);

  // Creating the buttons bar
  window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
  gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);

  // Creating the simulator program label and entry
  window->label_simulator
```

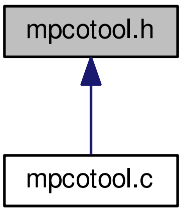
```
    = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
  window->button_simulator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Simulator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
                               gettext ("Simulator program executable file"));

  // Creating the evaluator program label and entry
  window->check_evaluator = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
  g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
  window->button_evaluator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Evaluator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_evaluator),
     gettext ("Optional evaluator program executable file"));

  // Creating the results files labels and entries
  window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
  window->entry_result = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
  window->label_variables
    = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
  window->entry_variables = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variables),
     gettext ("All simulated results file"));

  // Creating the files grid and attaching widgets
  window->grid_files = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
                   0, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
                   1, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
                   2, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
                   3, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
                   0, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
                   1, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
                   2, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
                   3, 1, 1, 1);

  // Creating the algorithm properties
  window->label_simulations = (GtkLabel *) gtk_label_new
    (gettext ("Simulations number"));
  window->spin_simulations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_simulations),
     gettext ("Number of simulations to perform for each iteration"));
  window->label_iterations = (GtkLabel *)
    gtk_label_new (gettext ("Iterations number"));
  window->spin_iterations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
  g_signal_connect
    (window->spin_iterations, "value-changed", window_update, NULL);
  window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
  window->spin_tolerance
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_tolerance),
     gettext ("Tolerance to set the variable interval on the next iteration"));
  window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
  window->spin_bests
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bests),
     gettext ("Number of best simulations used to set the variable interval "
              "on the next iteration"));
  window->label_population
    = (GtkLabel *) gtk_label_new (gettext ("Population number"));
  window->spin_population
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_population),
     gettext ("Number of population for the genetic algorithm"));
  window->label_generations
    = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
  window->spin_generations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_generations),
     gettext ("Number of generations for the genetic algorithm"));
  window->label_mutation
    = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
  window->spin_mutation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_mutation),
     gettext ("Ratio of mutation for the genetic algorithm"));
  window->label_reproduction
    = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
  window->spin_reproduction
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_reproduction),
     gettext ("Ratio of reproduction for the genetic algorithm"));
  window->label_adaptation
    = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
  window->spin_adaptation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_adaptation),
     gettext ("Ratio of adaptation for the genetic algorithm"));

  // Creating the gradient based method properties
  window->check_gradient = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Gradient based method"));
  g_signal_connect (window->check_gradient, "clicked",
window_update, NULL);
  window->grid_gradient = (GtkGrid *) gtk_grid_new ();
  window->button_gradient[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
  gtk_grid_attach (window->grid_gradient,
                   GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_gradient[0], "clicked",
window_update, NULL);
  for (i = 0; ++i < NGRADIENTS;)
    {
      window->button_gradient[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_gradient[0]),
         label_gradient[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
                                   tip_gradient[i]);
      gtk_grid_attach (window->grid_gradient,
                       GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
      g_signal_connect (window->button_gradient[i], "clicked",
                        window_update, NULL);
    }
  window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
  window->spin_steps = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e12, 1.);
  window->label_estimates
    = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
  window->spin_estimates = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e3, 1.);
  window->label_relaxation
    = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
  window->spin_relaxation = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., 2., 0.001);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_steps),
                   0, NGRADIENTS, 1, 1);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_steps),
                   1, NGRADIENTS, 1, 1);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_estimates),
                   0, NGRADIENTS + 1, 1, 1);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
spin_estimates),
                   1, NGRADIENTS + 1, 1, 1);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
label_relaxation),
                   0, NGRADIENTS + 2, 1, 1);
  gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->spin_relaxation),
                   1, NGRADIENTS + 2, 1, 1);

  // Creating the array of algorithms
  window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
  window->button_algorithm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
                               tip_algorithm[0]);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_algorithm[0], "clicked",
                    window_set_algorithm, NULL);
  for (i = 0; ++i < NALGORITHMS;)
    {
      window->button_algorithm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_algorithm[0]),
         label_algorithm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
                                   tip_algorithm[i]);
      gtk_grid_attach (window->grid_algorithm,
                       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_algorithm[i], "clicked",
                        window_set_algorithm, NULL);
    }
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_simulations), 0,
                   NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_iterations), 0,
                   NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_iterations), 1,
                   NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_tolerance), 0,
                   NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_tolerance), 1,
                   NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_population), 0,
                   NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_population), 1,
                   NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_generations), 0,
                   NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_generations), 1,
                   NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_mutation), 0,
                   NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_mutation), 1,
                   NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_reproduction), 0,
                   NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_reproduction), 1,
                   NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_adaptation), 0,
                   NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_adaptation), 1,
                   NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->check_gradient), 0,
                   NALGORITHMS + 9, 2, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->grid_gradient), 0,
                   NALGORITHMS + 10, 2, 1);
  window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
  gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
                     GTK_WIDGET (window->grid_algorithm));

  // Creating the variable widgets
  window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
  window->id_variable = g_signal_connect
    (window->combo_variable, "changed", window_set_variable, NULL);
  window->button_add_variable
    = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_variable, "clicked",
window_add_variable, NULL);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
  window->button_remove_variable
    = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
  window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
  window->entry_variable = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
  window->id_variable_label = g_signal_connect
    (window->entry_variable, "changed", window_label_variable, NULL);
  window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
  window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_min),
     gettext ("Minimum initial value of the variable"));
  window->scrolled_min
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_min),
                     GTK_WIDGET (window->spin_min));
  g_signal_connect (window->spin_min, "value-changed",
                    window_rangemin_variable, NULL);
  window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
  window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_max),
     gettext ("Maximum initial value of the variable"));
  window->scrolled_max
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_max),
                     GTK_WIDGET (window->spin_max));
  g_signal_connect (window->spin_max, "value-changed",
                    window_rangemax_variable, NULL);
  window->check_minabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
  g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
  window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_minabs),
     gettext ("Minimum allowed value of the variable"));
  window->scrolled_minabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
                     GTK_WIDGET (window->spin_minabs));
  g_signal_connect (window->spin_minabs, "value-changed",
                    window_rangeminabs_variable, NULL);
  window->check_maxabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
  g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
  window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_maxabs),
     gettext ("Maximum allowed value of the variable"));
  window->scrolled_maxabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
                     GTK_WIDGET (window->spin_maxabs));
  g_signal_connect (window->spin_maxabs, "value-changed",
                    window_rangemaxabs_variable, NULL);
  window->label_precision
    = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
  window->spin_precision = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_precision),
     gettext ("Number of precision floating point digits\n"
              "0 is for integer numbers"));
  g_signal_connect (window->spin_precision, "value-changed",
                    window_precision_variable, NULL);
  window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
  window->spin_sweeps
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_sweeps),
     gettext ("Number of steps sweeping the variable"));
  g_signal_connect
    (window->spin_sweeps, "value-changed", window_update_variable, NULL);
  window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
  window->spin_bits
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bits),
     gettext ("Number of bits to encode the variable"));
  g_signal_connect
    (window->spin_bits, "value-changed", window_update_variable, NULL);
  window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
  window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_step),
     gettext ("Initial step size for the gradient based method"));
  window->scrolled_step
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_step),
                     GTK_WIDGET (window->spin_step));
  g_signal_connect
    (window->spin_step, "value-changed", window_step_variable, NULL);
  window->grid_variable = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
  window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
  gtk_container_add (GTK_CONTAINER (window->frame_variable),
                     GTK_WIDGET (window->grid_variable));

  // Creating the experiment widgets
  window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
                               gettext ("Experiment selector"));
  window->id_experiment = g_signal_connect
    (window->combo_experiment, "changed", window_set_experiment, NULL)
;
  window->button_add_experiment
```

```
    = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
                               gettext ("Add experiment"));
  window->button_remove_experiment
    = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
                                                   GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_experiment, "clicked",
                    window_remove_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
                               gettext ("Remove experiment"));
  window->label_experiment
    = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
  window->button_experiment = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (gettext ("Experimental data file"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
                               gettext ("Experimental data file"));
  window->id_experiment_name
    = g_signal_connect (window->button_experiment, "selection-changed",
                        window_name_experiment, NULL);
  window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
  window->spin_weight
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_weight),
     gettext ("Weight factor to build the objective function"));
  g_signal_connect
    (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
  window->grid_experiment = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
  for (i = 0; i < MAX_NINPUTS; ++i)
    {
      snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
      window->check_template[i] = (GtkCheckButton *)
        gtk_check_button_new_with_label (buffer3);
      window->id_template[i]
        = g_signal_connect (window->check_template[i], "toggled",
                            window_inputs_experiment, NULL);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
      window->button_template[i] = (GtkFileChooserButton *)
        gtk_file_chooser_button_new (gettext ("Input template"),
                                     GTK_FILE_CHOOSER_ACTION_OPEN);
      gtk_widget_set_tooltip_text
        (GTK_WIDGET (window->button_template[i]),
         gettext ("Experimental input template file"));
      window->id_input[i]
        = g_signal_connect_swapped (window->button_template[i],
                                    "selection-changed",
                                    (void (*)) window_template_experiment,
                                    (void *) (size_t) i);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
    }
  window->frame_experiment
    = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
  gtk_container_add (GTK_CONTAINER (window->frame_experiment),
                     GTK_WIDGET (window->grid_experiment));

  // Creating the grid and attaching the widgets to the grid
  window->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->grid));

  // Setting the window logo
  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
  gtk_window_set_icon (window->window, window->logo);

  // Showing the window
  gtk_widget_show_all (GTK_WIDGET (window->window));

  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
#if GTK_MINOR_VERSION >= 16
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
#endif

  // Reading initial example
  input_new ();
  buffer2 = g_get_current_dir ();
  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
  g_free (buffer2);
  window_read (buffer);
  g_free (buffer);

#if DEBUG
  fprintf (stderr, "window_new: start\n");
#endif
}

#endif

int
cores_number ()
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}

int
main (int argn, char **argc)
{
#if HAVE_GTK
  char *buffer;
#endif

  // Starting pseudo-random numbers generator
  calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
  calibrate->seed = DEFAULT_RANDOM_SEED;

  // Allowing spaces in the XML data file
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
  printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

#if HAVE_GTK

  // Getting threads number
  nthreads_gradient = nthreads = cores_number ();

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory, LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  gtk_init (&argn, &argc);

  // Opening the main window
  window_new ();
  gtk_main ();

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_free (window->application_directory);

#else

  // Checking syntax
  if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
    {
      printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
      return 1;
    }

  // Getting threads number
  if (argn == 2)
    nthreads_gradient = nthreads = cores_number ();
  else
    {
      nthreads_gradient = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);

  // Making calibration
  if (input_open (argc[argn - 1]))
    calibrate_open ();

  // Freeing memory
  calibrate_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (calibrate->rng);

  // Closing
  return 0;
}
```

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** ()
- void ∗ calibrate_gradient_thread (ParallelData ∗data)

    Function to estimate the gradient on a thread.
- double **calibrate_variable_step_gradient** (unsigned int variable)
- 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_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

simulation	Simulation number.
value	Objective function value.

Definition at line 1443 of file mpcotool.c.

```
{
  unsigned int i, j;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_best: start\n");
  fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
           calibrate->nsaveds, calibrate->nbest);
#endif
  if (calibrate->nsaveds < calibrate->nbest
      || value < calibrate->error_best[calibrate->nsaveds - 1])
    {
      if (calibrate->nsaveds < calibrate->nbest)
        ++calibrate->nsaveds;
      calibrate->error_best[calibrate->nsaveds - 1] = value;
      calibrate->simulation_best[calibrate->
nsaveds - 1] = simulation;
      for (i = calibrate->nsaveds; --i;)
        {
          if (calibrate->error_best[i] < calibrate->
error_best[i - 1])
            {
              j = calibrate->simulation_best[i];
              e = calibrate->error_best[i];
              calibrate->simulation_best[i] = calibrate->
simulation_best[i - 1];
              calibrate->error_best[i] = calibrate->
error_best[i - 1];
              calibrate->simulation_best[i - 1] = j;
              calibrate->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_best: end\n");
#endif
}
```

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

simulation	Simulation number.
value	Objective function value.

Definition at line 1756 of file mpcotool.c.

```
{
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: start\n");
  fprintf (stderr,
           "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, calibrate->error_best[0]);
#endif
  if (value < calibrate->error_best[0])
    {
      calibrate->error_best[0] = value;
      calibrate->simulation_best[0] = simulation;
#if DEBUG
      fprintf (stderr,
               "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_best_gradient: end\n");
#endif
}
```

5.7.3.3 double calibrate_genetic_objective (Entity ∗ *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

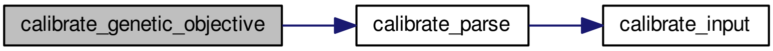
Returns

objective function value.

Definition at line 2059 of file mpcotool.c.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: start\n");
#endif
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      calibrate->value[entity->id * calibrate->nvariables + j]
        = genetic_get_variable (entity, calibrate->genetic_variable + j);
    }
  for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
    objective += calibrate_parse (entity->id, j);
  g_mutex_lock (mutex);
  for (j = 0; j < calibrate->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
      fprintf (calibrate->file_variables, buffer,
               genetic_get_variable (entity, calibrate->
genetic_variable + j));
    }
  fprintf (calibrate->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: end\n");
#endif
  return objective;
}
```

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

data	Function data.

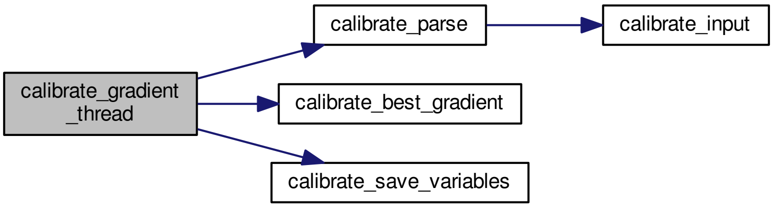
Returns

NULL

Definition at line 1821 of file mpcotool.c.

```
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
           thread,
           calibrate->thread_gradient[thread],
           calibrate->thread_gradient[thread + 1]);
#endif
  for (i = calibrate->thread_gradient[thread];
       i < calibrate->thread_gradient[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best_gradient (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_gradient_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

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

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 1196 of file mpcotool.c.

```
{
  unsigned int i;
  char buffer[32], value[32], *buffer2, *buffer3, *content;
  FILE *file;
  gsize length;
  GRegex *regex;

#if DEBUG
  fprintf (stderr, "calibrate_input: start\n");
#endif

  // Checking the file
  if (!template)
    goto calibrate_input_end;

  // Opening template
  content = g_mapped_file_get_contents (template);
  length = g_mapped_file_get_length (template);
#if DEBUG
  fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
           content);
#endif
  file = g_fopen (input, "w");

  // Parsing template
  for (i = 0; i < calibrate->nvariables; ++i)
    {
#if DEBUG
      fprintf (stderr, "calibrate_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             calibrate->label[i], 0, NULL);
#if DEBUG
          fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             calibrate->label[i], 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, 0, 0, NULL);
      snprintf (value, 32, format[calibrate->precision[i]],
                calibrate->value[simulation * calibrate->nvariables + i]);

#if DEBUG
      fprintf (stderr, "calibrate_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

calibrate_input_end:
#if DEBUG
  fprintf (stderr, "calibrate_input: end\n");
#endif
  return;
}
```

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

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1561 of file mpcotool.c.

```
{
  unsigned int i, j, k, s[calibrate->nbest];
  double e[calibrate->nbest];
#if DEBUG
  fprintf (stderr, "calibrate_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == calibrate->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
          if (i == calibrate->nsaveds)
            break;
        }
      else if (calibrate->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = calibrate->simulation_best[i];
          e[k] = calibrate->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < calibrate->nbest);
  calibrate->nsaveds = k;
  memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
  memcpy (calibrate->error_best, e, k * sizeof (double));
#if DEBUG
  fprintf (stderr, "calibrate_merge: end\n");
#endif
}
```

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

simulation	Simulation number.
experiment	Experiment number.

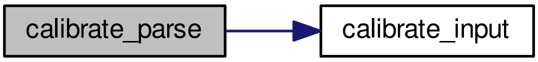
Returns

Objective function value.

Definition at line 1283 of file mpcotool.c.

```
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG
  fprintf (stderr, "calibrate_parse: start\n");
  fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
           experiment);
#endif

  // Opening input files
  for (i = 0; i < calibrate->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG
      fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      calibrate_input (simulation, &input[i][0],
                       calibrate->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG
  fprintf (stderr, "calibrate_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (calibrate->simulator);
  buffer3 = g_path_get_basename (calibrate->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
            input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG
  fprintf (stderr, "calibrate_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (calibrate->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (calibrate->evaluator);
      buffer3 = g_path_get_basename (calibrate->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, calibrate->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG
      fprintf (stderr, "calibrate_parse: %s\n", buffer);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG
  for (i = 0; i < calibrate->ninputs; ++i)
    {
      if (calibrate->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

#if DEBUG
  fprintf (stderr, "calibrate_parse: end\n");
#endif

  // Returning the objective function
  return e * calibrate->weight[experiment];
}
```

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

simulation	Simulation number.
error	Error value.

Definition at line 1415 of file mpcotool.c.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: start\n");
#endif
  for (i = 0; i < calibrate->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
      fprintf (calibrate->file_variables, buffer,
               calibrate->value[simulation * calibrate->
nvariables + i]);
    }
  fprintf (calibrate->file_variables, "%.14le\n", error);
#if DEBUG
  fprintf (stderr, "calibrate_save_variables: end\n");
#endif
}
```

5.7.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

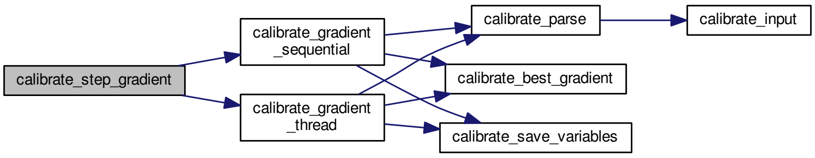
Parameters

simulation	Simulation number.

Definition at line 1923 of file mpcotool.c.

```
{
  GThread *thread[nthreads_gradient];
  ParallelData data[nthreads_gradient];
  unsigned int i, j, k, b;
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: start\n");
#endif
  for (i = 0; i < calibrate->nestimates; ++i)
    {
      k = (simulation + i) * calibrate->nvariables;
      b = calibrate->simulation_best[0] * calibrate->nvariables;
#if DEBUG
      fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
               simulation + i, calibrate->simulation_best[0]);
#endif
      for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
        {
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
                   i, j, calibrate->value[b]);
#endif
          calibrate->value[k]
            = calibrate->value[b] + calibrate_estimate_gradient (j, i);
          calibrate->value[k] = fmin (fmax (calibrate->value[k],
                                            calibrate->rangeminabs[j]),
                                      calibrate->rangemaxabs[j]);
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
                   i, j, calibrate->value[k]);
#endif
        }
    }
  if (nthreads_gradient == 1)
    calibrate_gradient_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_gradient; ++i)
        {
          calibrate->thread_gradient[i]
            = simulation + calibrate->nstart_gradient
            + i * (calibrate->nend_gradient - calibrate->nstart_gradient)
            / nthreads_gradient;
#if DEBUG
          fprintf (stderr,
                   "calibrate_step_gradient: i=%u thread_gradient=%u\n",
                   i, calibrate->thread_gradient[i]);
#endif
        }
      for (i = 0; i < nthreads_gradient; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
        }
      for (i = 0; i < nthreads_gradient; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG
  fprintf (stderr, "calibrate_step_gradient: end\n");
#endif
}
```

Here is the call graph for this function:



5.7.3.10 void∗ calibrate_thread (ParallelData ∗ *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

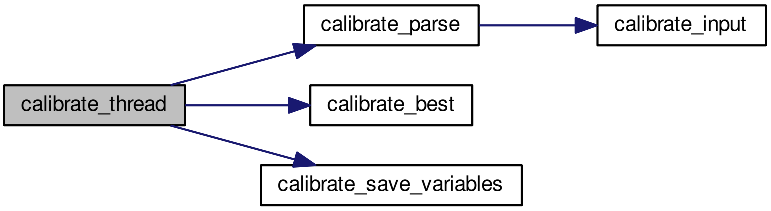
Returns

NULL

Definition at line 1517 of file mpcotool.c.

```
{
  unsigned int i, j, thread;
  double e;
#if DEBUG
  fprintf (stderr, "calibrate_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG
  fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
           calibrate->thread[thread], calibrate->thread[thread + 1]);
#endif
  for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
    {
      e = 0.;
      for (j = 0; j < calibrate->nexperiments; ++j)
        e += calibrate_parse (i, j);
      g_mutex_lock (mutex);
      calibrate_best (i, e);
      calibrate_save_variables (i, e);
      g_mutex_unlock (mutex);
#if DEBUG
      fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG
  fprintf (stderr, "calibrate_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

Here is the call graph for this function:



5.7.3.11 int input_open (char ∗ *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 548 of file mpcotool.c.

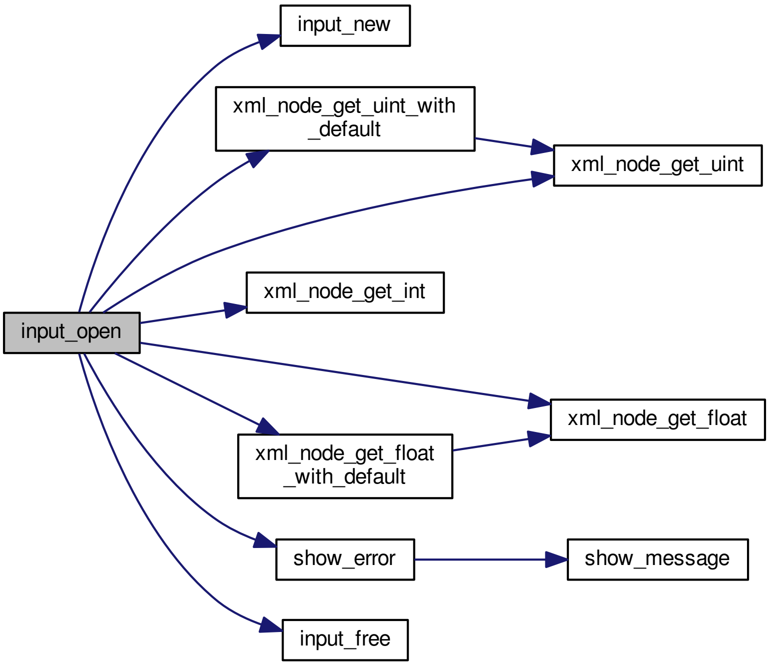
```
{
  char buffer2[64];
  char *buffert[MAX_NINPUTS] =
    { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
  xmlDoc *doc;
  xmlNode *node, *child;
  xmlChar *buffer;
  char *msg;
  int error_code;
  unsigned int i;

#if DEBUG
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input_new ();

  // Parsing the input file
#if DEBUG
  fprintf (stderr, "input_open: parsing the input file %s\n", filename);
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
      msg = gettext ("Unable to parse the input file");
      goto exit_on_error;
    }

  // Getting the root node
#if DEBUG
  fprintf (stderr, "input_open: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, XML_CALIBRATE))
    {
      msg = gettext ("Bad root XML node");
      goto exit_on_error;
    }

  // Getting results file names
  input->result = (char *) xmlGetProp (node, XML_RESULT);
  if (!input->result)
    input->result = (char *) xmlStrdup (result_name);
  input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
  if (!input->variables)
    input->variables = (char *) xmlStrdup (variables_name);

  // Opening simulator program name
  input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
  if (!input->simulator)
    {
      msg = gettext ("Bad simulator program");
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node,
XML_SEED, DEFAULT_RANDOM_SEED,
                                      &error_code);
  if (error_code)
    {
      msg = gettext ("Bad pseudo-random numbers generator seed");
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, XML_ALGORITHM);
  if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
      if (error_code)
        {
          msg = gettext ("Bad simulations number");
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, XML_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, XML_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, XML_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              msg = gettext ("Invalid population number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No population number");
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, XML_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              msg = gettext ("Invalid generations number");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No generations number");
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, XML_MUTATION))
```

```
        {
          input->mutation_ratio
            = xml_node_get_float (node, XML_MUTATION, &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              msg = gettext ("Invalid mutation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No mutation probability");
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, XML_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              msg = gettext ("Invalid reproduction probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No reproduction probability");
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, XML_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, XML_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->
nsimulations;
      i += input->adaptation_ratio * input->
nsimulations;
      if (i > input->nsimulations - 2)
        {
          msg = gettext
            ("No enough survival entities to reproduce the population");
          goto exit_on_error;
        }
    }
  else
    {
      msg = gettext ("Unknown algorithm");
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          msg = gettext ("Bad iterations number");
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, XML_NBEST, 1, &error_code);
      if (error_code || !input->nbest)
        {
          msg = gettext ("Invalid best number");
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node, XML_TOLERANCE, 0.,
                                            &error_code);
      if (error_code || input->tolerance < 0.)
        {
          msg = gettext ("Invalid tolerance");
          goto exit_on_error;
        }

      // Getting gradient method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node, XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->gradient_method = GRADIENT_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->gradient_method = GRADIENT_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the gradient");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node, XML_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) * sizeof (double));
      input->weight[input->nexperiments]
        = xml_node_get_float_with_default (child, XML_WEIGHT, 1., &error_code);
      if (error_code)
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("bad weight"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: weight=%lg\n",
               input->weight[input->nexperiments]);
#endif
      if (!input->nexperiments)
        input->ninputs = 0;
#if DEBUG
      fprintf (stderr, "input_open: template[0]\n");
#endif
      if (xmlHasProp (child, XML_TEMPLATE1))
        {
          input->template[0]
            = (char **) g_realloc (input->template[0],
                                   (1 + input->nexperiments) * sizeof (char *));
          buffert[0] = (char *) xmlGetProp (child, template[0]);
#if DEBUG
          fprintf (stderr, "input_open: experiment=%u template1=%s\n",
                   input->nexperiments, buffert[0]);
#endif
          if (!input->nexperiments)
            ++input->ninputs;
#if DEBUG
          fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Experiment"), buffer, gettext ("no template"));
          msg = buffer2;
          goto exit_on_error;
        }
      for (i = 1; i < MAX_NINPUTS; ++i)
        {
#if DEBUG
          fprintf (stderr, "input_open: template%u\n", i + 1);
#endif
          if (xmlHasProp (child, template[i]))
            {
              if (input->nexperiments && input->ninputs <= i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Experiment"),
                            buffer, gettext ("bad templates number"));
                  msg = buffer2;
                  while (i-- > 0)
                    xmlFree (buffert[i]);
                  goto exit_on_error;
                }
              input->template[i] = (char **)
                g_realloc (input->template[i],
                           (1 + input->nexperiments) * sizeof (char *));
              buffert[i] = (char *) xmlGetProp (child, template[i]);
#if DEBUG
              fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
                       input->nexperiments, i + 1,
                       input->template[i][input->nexperiments]);
#endif
              if (!input->nexperiments)
                ++input->ninputs;
#if DEBUG
              fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
#endif
            }
          else if (input->nexperiments && input->ninputs > i)
            {
              snprintf (buffer2, 64, "%s %s: %s%u",
                        gettext ("Experiment"),
                        buffer, gettext ("no template"), i + 1);
              msg = buffer2;
              while (i-- > 0)
```

```
                xmlFree (buffert[i]);
              goto exit_on_error;
            }
          else
            break;
        }
      input->experiment
        = g_realloc (input->experiment,
                     (1 + input->nexperiments) * sizeof (char *));
      input->experiment[input->nexperiments] = (char *) buffer;
      for (i = 0; i < input->ninputs; ++i)
        input->template[i][input->nexperiments] = buffert[i];
      ++input->nexperiments;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      msg = gettext ("No calibration experiments");
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  for (; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("bad XML node"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Variable"),
                    input->nvariables + 1, gettext ("no name"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MINIMUM))
        {
          input->rangemin = g_realloc
            (input->rangemin, (1 + input->nvariables) * sizeof (double));
          input->rangeminabs = g_realloc
            (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
          input->rangemin[input->nvariables]
            = xml_node_get_float (child, XML_MINIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangeminabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MINIMUM,
                                               -G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute minimum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemin[input->nvariables]
              < input->rangeminabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("minimum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no minimum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (xmlHasProp (child, XML_MAXIMUM))
        {
          input->rangemax = g_realloc
            (input->rangemax, (1 + input->nvariables) * sizeof (double));
          input->rangemaxabs = g_realloc
            (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
          input->rangemax[input->nvariables]
            = xml_node_get_float (child, XML_MAXIMUM, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"), buffer, gettext ("bad maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (child,
XML_ABSOLUTE_MAXIMUM,
                                               G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
      if (error_code || input->precision[input->nvariables] >=
NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
              if (error_code || !input->nsweeps[input->
nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables],
input->nsimulations);
#endif
        }
      if (input->algorithm == ALGORITHM_GENETIC)
        {
          // Obtaining bits representing each variable
          if (xmlHasProp (child, XML_NBITS))
            {
              input->nbits = (unsigned int *)
                g_realloc (input->nbits,
                           (1 + input->nvariables) * sizeof (unsigned int));
              i = xml_node_get_uint (child, XML_NBITS, &error_code);
              if (error_code || !i)
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("invalid bits number"));
                  msg = buffer2;
                  goto exit_on_error;
                }
              input->nbits[input->nvariables] = i;
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("no bits number"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else if (input->nsteps)
        {
          input->step = (double *)
            g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
          input->step[input->nvariables]
            = xml_node_get_float (child, XML_STEP, &error_code);
          if (error_code || input->step[input->nvariables] < 0.)
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("bad step size"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      input->label = g_realloc
        (input->label, (1 + input->nvariables) * sizeof (char *));
      input->label[input->nvariables] = (char *) buffer;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      msg = gettext ("No calibration variables");
      goto exit_on_error;
    }
  buffer = NULL;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
  show_error (msg);
  input_free ();
#if DEBUG
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
```

```
01182 }
```

Here is the call graph for this function:



5.7.3.12 void show_error (char ∗ *msg*)

Function to show a dialog with an error message.

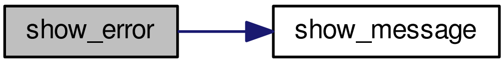
Parameters

msg	Error message.

Definition at line 256 of file mpcotool.c.

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

Here is the call graph for this function:



5.7.3.13 void show_message (char ∗ *title,* char ∗ *msg,* int *type*)

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file mpcotool.c.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *) gtk_message_dialog_new
    (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}
```

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

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 366 of file mpcotool.c.

```
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}
```

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

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

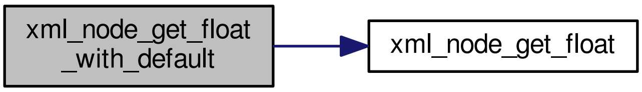
Returns

Floating point number value.

Definition at line 400 of file mpcotool.c.

```
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



5.7.3.16 int xml_node_get_int (xmlNode ∗ *node*, const xmlChar ∗ *prop*, int ∗ *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 274 of file mpcotool.c.

```
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
```

```
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

5.7.3.17 unsigned int xml_node_get_uint (xmlNode ∗ *node*, const xmlChar ∗ *prop*, int ∗ *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 305 of file mpcotool.c.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

5.7.3.18 unsigned int xml_node_get_uint_with_default (xmlNode ∗ *node*, const xmlChar ∗ *prop*, unsigned int *default_value*, int ∗ *error_code*)

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

Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

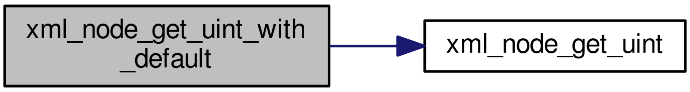
Returns

Unsigned integer number value.

Definition at line 339 of file mpcotool.c.

```
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
00348       *error_code = 0;
00349     }
00350   return i;
00351 }
```

Here is the call graph for this function:



5.7.3.19 void xml_node_set_float (xmlNode ∗ *node,* const xmlChar ∗ *prop,* double *value*)

Function to set a floating point number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 463 of file mpcotool.c.

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

5.7.3.20 void xml_node_set_int (xmlNode ∗ *node,* const xmlChar ∗ *prop,* int *value*)

Function to set an integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 425 of file mpcotool.c.

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

5.7.3.21 void xml_node_set_uint (xmlNode ∗ *node,* const xmlChar ∗ *prop,* unsigned int *value*)

Function to set an unsigned integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Unsigned integer number value.

Definition at line 444 of file mpcotool.c.

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

5.8 mpcotool.h

```
/*
MPCOTool: a software to make calibrations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2015, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef CALIBRATOR__H
#define CALIBRATOR__H 1

enum Algorithm
{
  ALGORITHM_MONTE_CARLO = 0,
  ALGORITHM_SWEEP = 1,
  ALGORITHM_GENETIC = 2
};

enum GradientMethod
{
  GRADIENT_METHOD_COORDINATES = 0,
  GRADIENT_METHOD_RANDOM = 1,
};

typedef struct
{
  char **template[MAX_NINPUTS];
  char **experiment;
  char **label;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  char *directory;
  char *name;
  double *rangemin;
  double *rangemax;
  double *rangeminabs;
  double *rangemaxabs;
  double *weight;
  double *step;
  unsigned int *precision;
  unsigned int *nsweeps;
  unsigned int *nbits;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int ninputs;
  unsigned int nsimulations;
  unsigned int algorithm;
  unsigned int nsteps;
  unsigned int gradient_method;
  unsigned int nestimates;
  unsigned int niterations;
  unsigned int nbest;
} Input;

typedef struct
{
  GMappedFile **file[MAX_NINPUTS];
  char **template[MAX_NINPUTS];
  char **experiment;
  char **label;
  gsl_rng *rng;
  GeneticVariable *genetic_variable;
  FILE *file_result;
  FILE *file_variables;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  double *value;
  double *rangemin;
  double *rangemax;
  double *rangeminabs;
  double *rangemaxabs;
  double *error_best;
  double *weight;
  double *step;
  double *gradient;
  double *value_old;
  double *error_old;
  unsigned int *precision;
  unsigned int *nsweeps;
  unsigned int *thread;
  unsigned int *thread_gradient;
  unsigned int *simulation_best;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  double calculation_time;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int ninputs;
  unsigned int nsimulations;
  unsigned int gradient_method;
  unsigned int nsteps;
  unsigned int nestimates;
  unsigned int algorithm;
  unsigned int nstart;
  unsigned int nend;
  unsigned int nstart_gradient;
  unsigned int nend_gradient;
  unsigned int niterations;
  unsigned int nbest;
  unsigned int nsaveds;
#if HAVE_MPI
  int mpi_rank;
#endif
} Calibrate;

typedef struct
{
  unsigned int thread;
} ParallelData;

// Public functions
void show_message (char *title, char *msg, int type);
void show_error (char *msg);
int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
                                int *error_code);
```

```
unsigned int xml_node_get_uint_with_default (xmlNode * node,
                                             const xmlChar * prop,
                                             unsigned int default_value,
                                             int *error_code);
double xml_node_get_float (xmlNode * node, const xmlChar * prop,
                           int *error_code);
double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
,
                                        double default_value, int *error_code);
void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
                        unsigned int value);
void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
void input_new ();
void input_free ();
int input_open (char *filename);
void calibrate_input (unsigned int simulation, char *input,
                      GMappedFile * template);
double calibrate_parse (unsigned int simulation, unsigned int experiment);
void calibrate_print ();
void calibrate_save_variables (unsigned int simulation, double error);
void calibrate_best (unsigned int simulation, double value);
void calibrate_sequential ();
void *calibrate_thread (ParallelData * data);
void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
                      double *error_best);
#if HAVE_MPI
void calibrate_synchronise ();
#endif
void calibrate_sweep ();
void calibrate_MonteCarlo ();
void calibrate_best_gradient (unsigned int simulation, double value);
void calibrate_gradient_sequential ();
void *calibrate_gradient_thread (ParallelData * data);
double calibrate_variable_step_gradient (unsigned int variable);
void calibrate_step_gradient (unsigned int simulation);
void calibrate_gradient ();
double calibrate_genetic_objective (Entity * entity);
void calibrate_genetic ();
void calibrate_save_old ();
void calibrate_merge_old ();
void calibrate_refine ();
void calibrate_step ();
void calibrate_iterate ();
void calibrate_open ();

#endif
```

Index