MPCOTool

1.5.3

Generated by Doxygen 1.8.9.1

Tue Jan 12 2016 12:14:01

Contents

Chapter 1

MPCOTool

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

VERSIONS

- 1.2.5: Stable and recommended version.
- 1.5.3: 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:

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

BUILDING INSTRUCTIONS

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

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

FreeBSD 10.2

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

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

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

2. Download this repository:

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

3. Link the latest genetic version to genetic:

   $ cd mpcotool/1.2.5
   $ ln -s ../../genetic/0.6.1 genetic

4. Build doing on a terminal:

   $ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

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

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

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

   $ make windist

Fedora Linux 23

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

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

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

MAKING MANUALS INSTRUCTIONS

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

$ make manuals

MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory mpcotool/1.2.5):

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

2. Build all tests doing in the same terminal:

```
$ cd ../../1.2.5
$ make tests
```

USER INSTRUCTIONS

- Command line in sequential mode:

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

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

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

- The syntax of the simulator has to be:

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

- The syntax of the program to evaluate the 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.
- double p

  Exponent of the P error norm.
- 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 129 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 162 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.

- double p

  Exponent of the P error norm.

- 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.

- unsigned int norm

  Error norm type.

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 80 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 203 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:

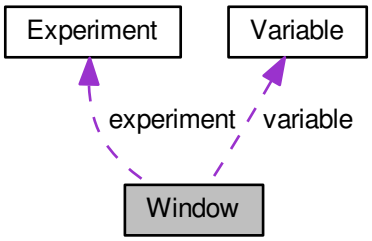
- interface.h

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_norm
  GtkFrame to set the error norm.
- GtkGrid * grid_norm
  GtkGrid to set the error norm.
- GtkRadioButton * button_norm [NNORMS]
  Array of GtkButtons to set the error norm.
- GtkLabel * label_p
  GtkLabel to set the p parameter.
- GtkSpinButton * spin_p
  GtkSpinButton to set the p parameter.
- GtkScrolledWindow * scrolled_p
  GtkScrolledWindow to set the p parameter.
- GtkFrame * frame_algorithm
  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm
  GtkGrid to set the algorithm.
- GtkRadioButton * button_algorithm [NALGORITHMS]
  Array of GtkButtons to set the algorithm.
- GtkLabel * label_simulations

GtkLabel to set the simulations number.

- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton * check_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

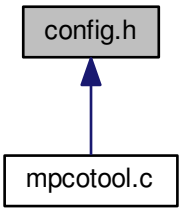
Chapter 5

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 NNORMS 4

  Number of error norms.
- #define NPRECISIONS 15

  Number of precisions.
- #define DEFAULT_PRECISION (NPRECISIONS - 1)

  Default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

  Default pseudo-random numbers seed.
- #define DEFAULT_RELAXATION 1.

  Default relaxation parameter.

- #define LOCALE_DIR "locales"

  Locales directory.
- #define PROGRAM_INTERFACE "mpcotool"

  Name of the interface program.
- #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"

  absolute minimum XML label.
- #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"

  absolute maximum XML label.
- #define XML_ADAPTATION (const xmlChar*)"adaptation"

  adaption XML label.
- #define XML_ALGORITHM (const xmlChar*)"algorithm"

  algoritm XML label.
- #define XML_CALIBRATE (const xmlChar*)"calibrate"

  calibrate XML label.
- #define XML_COORDINATES (const xmlChar*)"coordinates"

  coordinates XML label.
- #define XML_EUCLIDIAN (const xmlChar*)"euclidian"

  euclidian XML label.
- #define XML_EVALUATOR (const xmlChar*)"evaluator"

  evaluator XML label.
- #define XML_EXPERIMENT (const xmlChar*)"experiment"

  experiment XML label.
- #define XML_GENETIC (const xmlChar*)"genetic"

  genetic XML label.
- #define XML_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_NORM (const xmlChar*)"norm"

  norm XML label.
- #define XML_NPOPULATION (const xmlChar*)"npopulation"

  npopulation XML label.
- #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"

nsimulations XML label.

- #define XML_NSTEPS (const xmlChar*)"nsteps"

  nsteps XML label.
- #define XML_NSWEEPS (const xmlChar*)"nsweeps"

  nsweeps XML label.
- #define XML_P (const xmlChar*)"p"

  p XML label.
- #define XML_PRECISION (const xmlChar*)"precision"

  precision XML label.
- #define XML_RANDOM (const xmlChar*)"random"

  random XML label.
- #define XML_RELAXATION (const xmlChar*)"relaxation"

  relaxation XML label.
- #define XML_REPRODUCTION (const xmlChar*)"reproduction"

  reproduction XML label.
- #define XML_RESULT (const xmlChar*)"result"

  result XML label.
- #define XML_SIMULATOR (const xmlChar*)"simulator"

  simulator XML label.
- #define XML_SEED (const xmlChar*)"seed"

  seed XML label.
- #define XML_STEP (const xmlChar*)"step"

  step XML label.
- #define XML_SWEEP (const xmlChar*)"sweep"

  sweep XML label.
- #define XML_TAXICAB (const xmlChar*)"taxicab"

  taxicab XML label.
- #define XML_TEMPLATE1 (const xmlChar*)"template1"

  template1 XML label.
- #define XML_TEMPLATE2 (const xmlChar*)"template2"

  template2 XML label.
- #define XML_TEMPLATE3 (const xmlChar*)"template3"

  template3 XML label.
- #define XML_TEMPLATE4 (const xmlChar*)"template4"

  template4 XML label.
- #define XML_TEMPLATE5 (const xmlChar*)"template5"

  template5 XML label.
- #define XML_TEMPLATE6 (const xmlChar*)"template6"

  template6 XML label.
- #define XML_TEMPLATE7 (const xmlChar*)"template7"

  template7 XML label.
- #define XML_TEMPLATE8 (const xmlChar*)"template8"

  template8 XML label.
- #define XML_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

Definition in file config.h.

5.2 config.h

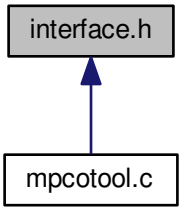
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2016, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
00045 #define NGRADIENTS 2
00046 #define NNORMS 4
00047 #define NPRECISIONS 15
00048
00049 // Default choices
00050
00051 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00052 #define DEFAULT_RANDOM_SEED 7007
00053 #define DEFAULT_RELAXATION 1.
00054
00055 // Interface labels
00056
00057 #define LOCALE_DIR "locales"
00058 #define PROGRAM_INTERFACE "mpcotool"
00059
00060 // XML labels
00061
00062 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00063 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00065 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00067 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00069 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00071 #define XML_COORDINATES (const xmlChar*)"coordinates"
00073 #define XML_EUCLIDIAN (const xmlChar*)"euclidian"
```

```
00075 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00077 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00079 #define XML_GENETIC (const xmlChar*)"genetic"
00081 #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
00082 #define XML_MINIMUM (const xmlChar*)"minimum"
00084 #define XML_MAXIMUM (const xmlChar*)"maximum"
00085 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00086 #define XML_MUTATION (const xmlChar*)"mutation"
00088 #define XML_NAME (const xmlChar*)"name"
00089 #define XML_NBEST (const xmlChar*)"nbest"
00090 #define XML_NBITS (const xmlChar*)"nbits"
00091 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00092 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00094 #define XML_NITERATIONS (const xmlChar*)"niterations"
00096 #define XML_NORM (const xmlChar*)"norm"
00098 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00099 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00101 #define XML_NSTEPS (const xmlChar*)"nsteps"
00103 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00104 #define XML_P (const xmlChar*)"p"
00105 #define XML_PRECISION (const xmlChar*)"precision"
00106 #define XML_RANDOM (const xmlChar*)"random"
00108 #define XML_RELAXATION (const xmlChar*)"relaxation"
00109 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00111 #define XML_RESULT (const xmlChar*)"result"
00113 #define XML_SIMULATOR (const xmlChar*)"simulator"
00114 #define XML_SEED (const xmlChar*)"seed"
00116 #define XML_STEP (const xmlChar*)"step"
00117 #define XML_SWEEP (const xmlChar*)"sweep"
00118 #define XML_TAXICAB (const xmlChar*)"taxicab"
00119 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00120 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00122 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00124 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00126 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00128 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00130 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00132 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00134 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00136 #define XML_VARIABLE (const xmlChar*)"variable"
00138 #define XML_VARIABLES (const xmlChar*)"variables"
00139 #define XML_WEIGHT (const xmlChar*)"weight"
00141
00142 #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

- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.
- void input_save (char *filename)

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

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

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

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

  Function to get the gradient base method number.
- unsigned int window_get_norm ()

  Function to get the norm 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

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 5147 of file mpcotool.c.

```
05148 {
05149 #ifdef G_OS_WIN32
05150   SYSTEM_INFO sysinfo;
05151   GetSystemInfo (&sysinfo);
05152   return sysinfo.dwNumberOfProcessors;
05153 #else
05154   return (int) sysconf (_SC_NPROCESSORS_ONLN);
05155 #endif
05156 }
```

5.3.2.2 unsigned int gtk_array_get_active (GtkRadioButton * *array[ ]*, unsigned int *n*)

Function to get the active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 486 of file mpcotool.c.

```
00487 {
00488   unsigned int i;
00489   for (i = 0; i < n; ++i)
00490     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00491       break;
00492   return i;
00493 }
```

5.3.2.3 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

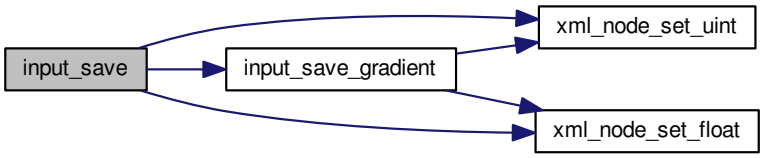
Definition at line 2874 of file mpcotool.c.

```
02875 {
02876   unsigned int i, j;
02877   char *buffer;
02878   xmlDoc *doc;
02879   xmlNode *node, *child;
02880   GFile *file, *file2;
02881
02882 #if DEBUG
02883   fprintf (stderr, "input_save: start\n");
02884 #endif
02885
```

```
02886   // Getting the input file directory
02887   input->name = g_path_get_basename (filename);
02888   input->directory = g_path_get_dirname (filename);
02889   file = g_file_new_for_path (input->directory);
02890
02891   // Opening the input file
02892   doc = xmlNewDoc ((const xmlChar *) "1.0");
02893
02894   // Setting root XML node
02895   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02896   xmlDocSetRootElement (doc, node);
02897
02898   // Adding properties to the root XML node
02899   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02900     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02901   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02902     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02903   file2 = g_file_new_for_path (input->simulator);
02904   buffer = g_file_get_relative_path (file, file2);
02905   g_object_unref (file2);
02906   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02907   g_free (buffer);
02908   if (input->evaluator)
02909     {
02910       file2 = g_file_new_for_path (input->evaluator);
02911       buffer = g_file_get_relative_path (file, file2);
02912       g_object_unref (file2);
02913       if (xmlStrlen ((xmlChar *) buffer))
02914         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02915       g_free (buffer);
02916     }
02917   if (input->seed != DEFAULT_RANDOM_SEED)
02918     xml_node_set_uint (node, XML_SEED, input->seed);
02919
02920   // Setting the algorithm
02921   buffer = (char *) g_malloc (64);
02922   switch (input->algorithm)
02923     {
02924     case ALGORITHM_MONTE_CARLO:
02925       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02926       snprintf (buffer, 64, "%u", input->nsimulations);
02927       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02928       snprintf (buffer, 64, "%u", input->niterations);
02929       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02930       snprintf (buffer, 64, "%.3lg", input->tolerance);
02931       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02932       snprintf (buffer, 64, "%u", input->nbest);
02933       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02934       input_save_gradient (node);
02935       break;
02936     case ALGORITHM_SWEEP:
02937       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02938       snprintf (buffer, 64, "%u", input->niterations);
02939       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02940       snprintf (buffer, 64, "%.3lg", input->tolerance);
02941       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02942       snprintf (buffer, 64, "%u", input->nbest);
02943       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02944       input_save_gradient (node);
02945       break;
02946     default:
02947       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02948       snprintf (buffer, 64, "%u", input->nsimulations);
02949       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02950       snprintf (buffer, 64, "%u", input->niterations);
02951       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02952       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02953       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02954       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02955       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02956       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02957       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02958       break;
02959     }
02960   g_free (buffer);
02961
02962   // Setting the experimental data
02963   for (i = 0; i < input->nexperiments; ++i)
02964     {
02965       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02966       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02967       if (input->weight[i] != 1.)
02968         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02969       for (j = 0; j < input->ninputs; ++j)
```

```
02970         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02971     }
02972
02973   // Setting the variables data
02974   for (i = 0; i < input->nvariables; ++i)
02975     {
02976       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02977       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02978       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02979       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02980         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02981       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02982       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02983         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02984       if (input->precision[i] != DEFAULT_PRECISION)
02985         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02986       if (input->algorithm == ALGORITHM_SWEEP)
02987         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02988       else if (input->algorithm == ALGORITHM_GENETIC)
02989         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02990       if (input->nsteps)
02991         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02992     }
02993
02994   // Saving the error norm
02995   switch (input->norm)
02996     {
02997     case ERROR_NORM_MAXIMUM:
02998       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
02999       break;
03000     case ERROR_NORM_P:
03001       xmlSetProp (node, XML_NORM, XML_P);
03002       xml_node_set_float (node, XML_P, input->p);
03003       break;
03004     case ERROR_NORM_TAXICAB:
03005       xmlSetProp (node, XML_NORM, XML_TAXICAB);
03006     }
03007
03008   // Saving the XML file
03009   xmlSaveFormatFile (filename, doc, 1);
03010
03011   // Freeing memory
03012   xmlFreeDoc (doc);
03013
03014 #if DEBUG
03015   fprintf (stderr, "input_save: end\n");
03016 #endif
03017 }
```

Here is the call graph for this function:



5.3.2.4 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 3121 of file mpcotool.c.

```
03122 {
03123   unsigned int i;
03124 #if DEBUG
03125   fprintf (stderr, "window_get_algorithm: start\n");
03126 #endif
03127   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
03128 #if DEBUG
03129   fprintf (stderr, "window_get_algorithm: %u\n", i);
03130   fprintf (stderr, "window_get_algorithm: end\n");
03131 #endif
03132   return i;
03133 }
```

Here is the call graph for this function:

5.3.2.5 unsigned int window_get_gradient ( )

Function to get the gradient base method number.

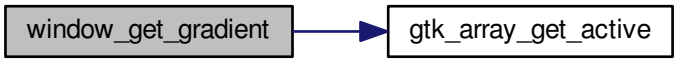
Returns

Gradient base method number.

Definition at line 3141 of file mpcotool.c.

```
03142 {
03143   unsigned int i;
03144 #if DEBUG
03145   fprintf (stderr, "window_get_gradient: start\n");
03146 #endif
03147   i = gtk_array_get_active (window->button_gradient ,
      NGRADIENTS);
03148 #if DEBUG
03149   fprintf (stderr, "window_get_gradient: %u\n", i);
03150   fprintf (stderr, "window_get_gradient: end\n");
03151 #endif
03152   return i;
03153 }
```

Here is the call graph for this function:



5.3.2.6 unsigned int window_get_norm ( )

Function to get the norm base method number.

Returns

Gradient base method number.

Definition at line 3161 of file mpcotool.c.

```
03162 {
03163   unsigned int i;
03164 #if DEBUG
03165   fprintf (stderr, "window_get_norm: start\n");
03166 #endif
03167   i = gtk_array_get_active (window->button_norm ,
      NNORMS);
03168 #if DEBUG
03169   fprintf (stderr, "window_get_norm: %u\n", i);
03170   fprintf (stderr, "window_get_norm: end\n");
03171 #endif
03172   return i;
03173 }
```

Here is the call graph for this function:

5.3.2.7 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.

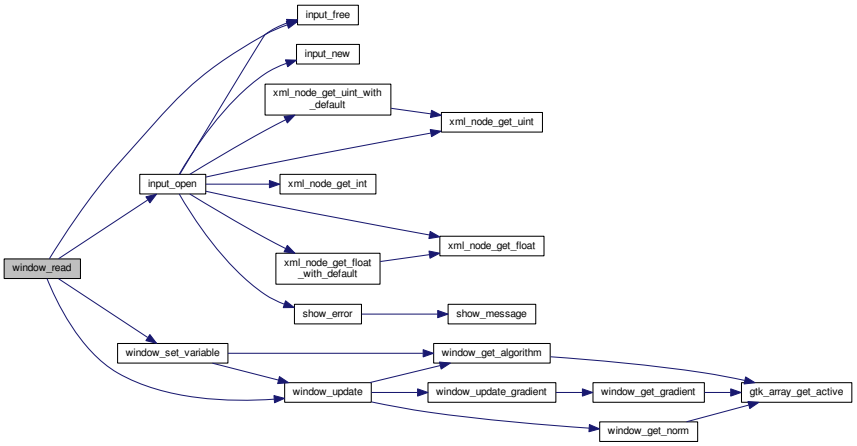
Definition at line 4264 of file mpcotool.c.

```
04265 {
04266   unsigned int i;
04267   char *buffer;
04268 #if DEBUG
04269   fprintf (stderr, "window_read: start\n");
04270 #endif
04271
04272   // Reading new input file
04273   input_free ();
04274   if (!input_open (filename))
04275     return 0;
04276
04277   // Setting GTK+ widgets data
04278   gtk_entry_set_text (window->entry_result, input->result);
04279   gtk_entry_set_text (window->entry_variables, input->
      variables);
```

```
04280   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04282                                  (window->button_simulator), buffer);
04283   g_free (buffer);
04284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04285                                 (size_t) input->evaluator);
04286   if (input->evaluator)
04287     {
04288       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04290                                      (window->button_evaluator), buffer);
04291       g_free (buffer);
04292     }
04293   gtk_toggle_button_set_active
04294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04295   switch (input->algorithm)
04296     {
04297     case ALGORITHM_MONTE_CARLO:
04298       gtk_spin_button_set_value (window->spin_simulations,
04299                                  (gdouble) input->nsimulations);
04300     case ALGORITHM_SWEEP:
04301       gtk_spin_button_set_value (window->spin_iterations,
04302                                  (gdouble) input->niterations);
04303       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04304       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04305       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04306                                     input->nsteps);
04307       if (input->nsteps)
04308         {
04309           gtk_toggle_button_set_active
04310             (GTK_TOGGLE_BUTTON (window->button_gradient
04311                                 [input->gradient_method]), TRUE);
04312           gtk_spin_button_set_value (window->spin_steps,
04313                                      (gdouble) input->nsteps);
04314           gtk_spin_button_set_value (window->spin_relaxation,
04315                                      (gdouble) input->relaxation);
04316           switch (input->gradient_method)
04317             {
04318             case GRADIENT_METHOD_RANDOM:
04319               gtk_spin_button_set_value (window->spin_estimates,
04320                                          (gdouble) input->nestimates);
04321             }
04322         }
04323       break;
04324     default:
04325       gtk_spin_button_set_value (window->spin_population,
04326                                  (gdouble) input->nsimulations);
04327       gtk_spin_button_set_value (window->spin_generations,
04328                                  (gdouble) input->niterations);
04329       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04330       gtk_spin_button_set_value (window->spin_reproduction,
04331                                  input->reproduction_ratio);
04332       gtk_spin_button_set_value (window->spin_adaptation,
04333                                  input->adaptation_ratio);
04334     }
04335   gtk_toggle_button_set_active
04336     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
04337   gtk_spin_button_set_value (window->spin_p, input->p);
04338   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04339   g_signal_handler_block (window->button_experiment,
04340                           window->id_experiment_name);
04341   gtk_combo_box_text_remove_all (window->combo_experiment);
04342   for (i = 0; i < input->nexperiments; ++i)
04343     gtk_combo_box_text_append_text (window->combo_experiment,
04344                                     input->experiment[i]);
04345   g_signal_handler_unblock
04346     (window->button_experiment, window->
      id_experiment_name);
04347   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04348   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04349   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04350   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04351   gtk_combo_box_text_remove_all (window->combo_variable);
04352   for (i = 0; i < input->nvariables; ++i)
04353     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
```

```
04354   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04355   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04356   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04357   window_set_variable ();
04358   window_update ();
04359
04360 #if DEBUG
04361   fprintf (stderr, "window_read: end\n");
04362 #endif
04363   return 1;
04364 }
```

Here is the call graph for this function:



5.3.2.8 int window_save ( )

Function to save the input file.

Returns

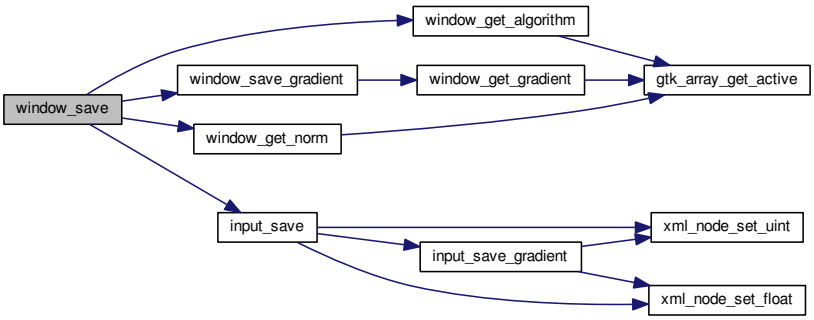
1 on OK, 0 on Cancel.

Definition at line 3213 of file mpcotool.c.

```
03214 {
03215   GtkFileChooserDialog *dlg;
03216   GtkFileFilter *filter;
03217   char *buffer;
03218
03219 #if DEBUG
03220   fprintf (stderr, "window_save: start\n");
03221 #endif
03222
03223   // Opening the saving dialog
03224   dlg = (GtkFileChooserDialog *)
03225     gtk_file_chooser_dialog_new (gettext ("Save file"),
03226                                  window->window,
03227                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03228                                  gettext ("_Cancel"),
03229                                  GTK_RESPONSE_CANCEL,
03230                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03231   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03232   buffer = g_build_filename (input->directory, input->name, NULL);
03233   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03234   g_free (buffer);
03235
03236   // Adding XML filter
03237   filter = (GtkFileFilter *) gtk_file_filter_new ();
```

```
03238   gtk_file_filter_set_name (filter, "XML");
03239   gtk_file_filter_add_pattern (filter, "*.xml");
03240   gtk_file_filter_add_pattern (filter, "*.XML");
03241   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03242
03243   // If OK response then saving
03244   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03245     {
03246
03247       // Adding properties to the root XML node
03248       input->simulator = gtk_file_chooser_get_filename
03249         (GTK_FILE_CHOOSER (window->button_simulator));
03250       if (gtk_toggle_button_get_active
03251           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03252         input->evaluator = gtk_file_chooser_get_filename
03253           (GTK_FILE_CHOOSER (window->button_evaluator));
03254       else
03255         input->evaluator = NULL;
03256       input->result
03257         = (char *) xmlStrdup ((const xmlChar *)
03258                               gtk_entry_get_text (window->entry_result));
03259       input->variables
03260         = (char *) xmlStrdup ((const xmlChar *)
03261                               gtk_entry_get_text (window->entry_variables));
03262
03263       // Setting the algorithm
03264       switch (window_get_algorithm ())
03265         {
03266         case ALGORITHM_MONTE_CARLO:
03267           input->algorithm = ALGORITHM_MONTE_CARLO;
03268           input->nsimulations
03269             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03270           input->niterations
03271             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03272           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03273           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03274           window_save_gradient ();
03275           break;
03276         case ALGORITHM_SWEEP:
03277           input->algorithm = ALGORITHM_SWEEP;
03278           input->niterations
03279             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03280           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03281           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03282           window_save_gradient ();
03283           break;
03284         default:
03285           input->algorithm = ALGORITHM_GENETIC;
03286           input->nsimulations
03287             = gtk_spin_button_get_value_as_int (window->spin_population);
03288           input->niterations
03289             = gtk_spin_button_get_value_as_int (window->spin_generations);
03290           input->mutation_ratio
03291             = gtk_spin_button_get_value (window->spin_mutation);
03292           input->reproduction_ratio
03293             = gtk_spin_button_get_value (window->spin_reproduction);
03294           input->adaptation_ratio
03295             = gtk_spin_button_get_value (window->spin_adaptation);
03296           break;
03297         }
03298       input->norm = window_get_norm ();
03299       input->p = gtk_spin_button_get_value (window->spin_p);
03300
03301       // Saving the XML file
03302       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03303       input_save (buffer);
03304
03305       // Closing and freeing memory
03306       g_free (buffer);
03307       gtk_widget_destroy (GTK_WIDGET (dlg));
03308 #if DEBUG
03309       fprintf (stderr, "window_save: end\n");
03310 #endif
03311       return 1;
03312     }
03313
03314   // Closing and freeing memory
03315   gtk_widget_destroy (GTK_WIDGET (dlg));
03316 #if DEBUG
03317   fprintf (stderr, "window_save: end\n");
03318 #endif
03319   return 0;
03320 }
```

Here is the call graph for this function:



5.3.2.9 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 3868 of file mpcotool.c.

```
03869 {
03870   unsigned int i, j;
03871   char *buffer;
03872   GFile *file1, *file2;
03873 #if DEBUG
03874   fprintf (stderr, "window_template_experiment: start\n");
03875 #endif
03876   i = (size_t) data;
03877   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03878   file1
03879     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03880   file2 = g_file_new_for_path (input->directory);
03881   buffer = g_file_get_relative_path (file2, file1);
03882   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03883   g_free (buffer);
03884   g_object_unref (file2);
03885   g_object_unref (file1);
03886 #if DEBUG
03887   fprintf (stderr, "window_template_experiment: end\n");
03888 #endif
03889 }
```

5.4 interface.h

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2016, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017
```

```
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef INTERFACE__H
00037 #define INTERFACE__H 1
00038
00039 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00040
00046 typedef struct
00047 {
00048   char *template[MAX_NINPUTS];
00049   char *name;
00050   double weight;
00052 } Experiment;
00053
00058 typedef struct
00059 {
00060   char *label;
00061   double rangemin;
00062   double rangemax;
00063   double rangeminabs;
00064   double rangemaxabs;
00065   double step;
00067   unsigned int precision;
00068   unsigned int nsweeps;
00069   unsigned int nbits;
00070 } Variable;
00071
00076 typedef struct
00077 {
00078   GtkDialog *dialog;
00079   GtkGrid *grid;
00080   GtkLabel *label_seed;
00082   GtkSpinButton *spin_seed;
00084   GtkLabel *label_threads;
00085   GtkSpinButton *spin_threads;
00086   GtkLabel *label_gradient;
00087   GtkSpinButton *spin_gradient;
00088 } Options;
00089
00094 typedef struct
00095 {
00096   GtkDialog *dialog;
00097   GtkLabel *label;
00098 } Running;
00099
00104 typedef struct
00105 {
00106   GtkWindow *window;
00107   GtkGrid *grid;
00108   GtkToolbar *bar_buttons;
00109   GtkToolButton *button_open;
00110   GtkToolButton *button_save;
00111   GtkToolButton *button_run;
00112   GtkToolButton *button_options;
00113   GtkToolButton *button_help;
00114   GtkToolButton *button_about;
00115   GtkToolButton *button_exit;
00116   GtkGrid *grid_files;
00117   GtkLabel *label_simulator;
00118   GtkFileChooserButton *button_simulator;
00120   GtkCheckButton *check_evaluator;
00121   GtkFileChooserButton *button_evaluator;
00123   GtkLabel *label_result;
00124   GtkEntry *entry_result;
00125   GtkLabel *label_variables;
00126   GtkEntry *entry_variables;
00127   GtkFrame *frame_norm;
00128   GtkGrid *grid_norm;
00129   GtkRadioButton *button_norm[NNORMS];
00131   GtkLabel *label_p;
00132   GtkSpinButton *spin_p;
00133   GtkScrolledWindow *scrolled_p;
00134   GtkFrame *frame_algorithm;
00135   GtkGrid *grid_algorithm;
00136   GtkRadioButton *button_algorithm[NALGORITHMS];
00138   GtkLabel *label_simulations;
00139   GtkSpinButton *spin_simulations;
```

```
00141   GtkLabel *label_iterations;
00142   GtkSpinButton *spin_iterations;
00144   GtkLabel *label_tolerance;
00145   GtkSpinButton *spin_tolerance;
00146   GtkLabel *label_bests;
00147   GtkSpinButton *spin_bests;
00148   GtkLabel *label_population;
00149   GtkSpinButton *spin_population;
00151   GtkLabel *label_generations;
00152   GtkSpinButton *spin_generations;
00154   GtkLabel *label_mutation;
00155   GtkSpinButton *spin_mutation;
00156   GtkLabel *label_reproduction;
00157   GtkSpinButton *spin_reproduction;
00159   GtkLabel *label_adaptation;
00160   GtkSpinButton *spin_adaptation;
00162   GtkCheckButton *check_gradient;
00164   GtkGrid *grid_gradient;
00166   GtkRadioButton *button_gradient[NGRADIENTS];
00168   GtkLabel *label_steps;
00169   GtkSpinButton *spin_steps;
00170   GtkLabel *label_estimates;
00171   GtkSpinButton *spin_estimates;
00173   GtkLabel *label_relaxation;
00175   GtkSpinButton *spin_relaxation;
00177   GtkFrame *frame_variable;
00178   GtkGrid *grid_variable;
00179   GtkComboBoxText *combo_variable;
00181   GtkButton *button_add_variable;
00182   GtkButton *button_remove_variable;
00183   GtkLabel *label_variable;
00184   GtkEntry *entry_variable;
00185   GtkLabel *label_min;
00186   GtkSpinButton *spin_min;
00187   GtkScrolledWindow *scrolled_min;
00188   GtkLabel *label_max;
00189   GtkSpinButton *spin_max;
00190   GtkScrolledWindow *scrolled_max;
00191   GtkCheckButton *check_minabs;
00192   GtkSpinButton *spin_minabs;
00193   GtkScrolledWindow *scrolled_minabs;
00194   GtkCheckButton *check_maxabs;
00195   GtkSpinButton *spin_maxabs;
00196   GtkScrolledWindow *scrolled_maxabs;
00197   GtkLabel *label_precision;
00198   GtkSpinButton *spin_precision;
00199   GtkLabel *label_sweeps;
00200   GtkSpinButton *spin_sweeps;
00201   GtkLabel *label_bits;
00202   GtkSpinButton *spin_bits;
00203   GtkLabel *label_step;
00204   GtkSpinButton *spin_step;
00205   GtkScrolledWindow *scrolled_step;
00206   GtkFrame *frame_experiment;
00207   GtkGrid *grid_experiment;
00208   GtkComboBoxText *combo_experiment;
00209   GtkButton *button_add_experiment;
00210   GtkButton *button_remove_experiment;
00211   GtkLabel *label_experiment;
00212   GtkFileChooserButton *button_experiment;
00214   GtkLabel *label_weight;
00215   GtkSpinButton *spin_weight;
00216   GtkCheckButton *check_template[MAX_NINPUTS];
00218   GtkFileChooserButton *button_template[MAX_NINPUTS];
00220   GdkPixbuf *logo;
00221   Experiment *experiment;
00222   Variable *variable;
00223   char *application_directory;
00224   gulong id_experiment;
00225   gulong id_experiment_name;
00226   gulong id_variable;
00227   gulong id_variable_label;
00228   gulong id_template[MAX_NINPUTS];
00230   gulong id_input[MAX_NINPUTS];
00232   unsigned int nexperiments;
00233   unsigned int nvariables;
00234 } Window;
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);
00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_gradient ();
00243 unsigned int window_get_norm ();
00244 void window_save_gradient ();
```

```
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_gradient ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new ();
00271 int cores_number ();
00272
00273 #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 <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.

- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.

- 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.

- double calibrate_norm_euclidian (unsigned int simulation)

  Function to calculate the Euclidian error norm.

- double calibrate_norm_maximum (unsigned int simulation)

  Function to calculate the maximum error norm.

- double calibrate_norm_p (unsigned int simulation)

  Function to calculate the P error norm.

- double calibrate_norm_taxicab (unsigned int simulation)

Function to calculate the taxicab error norm.

- 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.
- unsigned int window_get_algorithm ()

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

  Function to get the gradient base method number.
- unsigned int window_get_norm ()

  Function to get the norm 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.
- double(* calibrate_norm)(unsigned int simulation)

  Pointer to the error norm function.
- 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

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 1610 of file mpcotool.c.

```
01611 {
01612   unsigned int i, j;
01613   double e;
01614 #if DEBUG
01615   fprintf (stderr, "calibrate_best: start\n");
01616   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01617            calibrate->nsaveds, calibrate->nbest);
01618 #endif
01619   if (calibrate->nsaveds < calibrate->nbest
01620       || value < calibrate->error_best[calibrate->nsaveds - 1])
01621     {
01622       if (calibrate->nsaveds < calibrate->nbest)
01623         ++calibrate->nsaveds;
01624       calibrate->error_best[calibrate->nsaveds - 1] = value;
01625       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01626       for (i = calibrate->nsaveds; --i;)
01627         {
01628           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
```

```
01629             {
01630               j = calibrate->simulation_best[i];
01631               e = calibrate->error_best[i];
01632               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01633               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01634               calibrate->simulation_best[i - 1] = j;
01635               calibrate->error_best[i - 1] = e;
01636             }
01637           else
01638             break;
01639         }
01640     }
01641 #if DEBUG
01642   fprintf (stderr, "calibrate_best: end\n");
01643 #endif
01644 }
```

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 1919 of file mpcotool.c.

```
01920 {
01921 #if DEBUG
01922   fprintf (stderr, "calibrate_best_gradient: start\n");
01923   fprintf (stderr,
01924            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01925            simulation, value, calibrate->error_best[0]);
01926 #endif
01927   if (value < calibrate->error_best[0])
01928     {
01929       calibrate->error_best[0] = value;
01930       calibrate->simulation_best[0] = simulation;
01931 #if DEBUG
01932       fprintf (stderr,
01933                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01934                simulation, value);
01935 #endif
01936     }
01937 #if DEBUG
01938   fprintf (stderr, "calibrate_best_gradient: end\n");
01939 #endif
01940 }
```

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 2052 of file mpcotool.c.

```
02054 {
02055   double x;
02056 #if DEBUG
02057   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
02058 #endif
02059   x = calibrate->gradient[variable];
02060   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
02061     {
02062       if (estimate & 1)
02063         x += calibrate->step[variable];
```

```
02064       else
02065         x -= calibrate->step[variable];
02066     }
02067 #if DEBUG
02068   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
02069            variable, x);
02070   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
02071 #endif
02072   return x;
02073 }
```

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 2025 of file mpcotool.c.

```
02027 {
02028   double x;
02029 #if DEBUG
02030   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
02031 #endif
02032   x = calibrate->gradient[variable]
02033     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
      step[variable];
02034 #if DEBUG
02035   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
02036            variable, x);
02037   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
02038 #endif
02039   return x;
02040 }
```

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 2218 of file mpcotool.c.

```
02219 {
02220   unsigned int j;
02221   double objective;
02222   char buffer[64];
02223 #if DEBUG
02224   fprintf (stderr, "calibrate_genetic_objective: start\n");
02225 #endif
02226   for (j = 0; j < calibrate->nvariables; ++j)
02227     {
02228       calibrate->value[entity->id * calibrate->nvariables + j]
02229         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02230     }
02231   objective = calibrate_norm (entity->id);
02232   g_mutex_lock (mutex);
02233   for (j = 0; j < calibrate->nvariables; ++j)
02234     {
02235       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02236       fprintf (calibrate->file_variables, buffer,
02237                genetic_get_variable (entity, calibrate->
```

```
      genetic_variable + j));
02238     }
02239   fprintf (calibrate->file_variables, "%.14le\n", objective);
02240   g_mutex_unlock (mutex);
02241 #if DEBUG
02242   fprintf (stderr, "calibrate_genetic_objective: end\n");
02243 #endif
02244   return objective;
02245 }
```

5.5.2.6 void calibrate_gradient_sequential (unsigned int *simulation*)

Function to estimate the gradient sequentially.

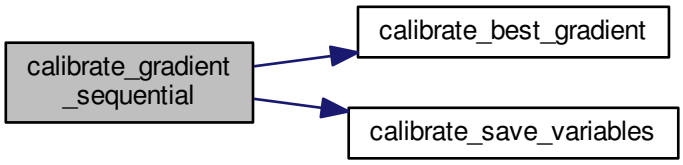
Parameters

simulation	Simulation number.

Definition at line 1949 of file mpcotool.c.

```
01950 {
01951   unsigned int i, j;
01952   double e;
01953 #if DEBUG
01954   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01955   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01956            "nend_gradient=%u\n",
01957            calibrate->nstart_gradient, calibrate->
      nend_gradient);
01958 #endif
01959   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01960     {
01961       j = simulation + i;
01962       e = calibrate_norm (j);
01963       calibrate_best_gradient (j, e);
01964       calibrate_save_variables (j, e);
01965 #if DEBUG
01966       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01967 #endif
01968     }
01969 #if DEBUG
01970   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01971 #endif
01972 }
```

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.

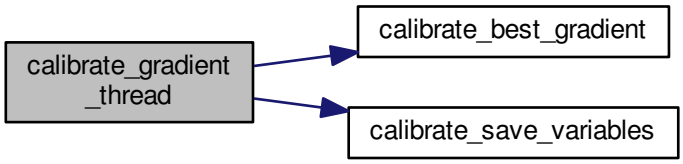
Returns

NULL

Definition at line 1982 of file mpcotool.c.

```
01983 {
01984   unsigned int i, thread;
01985   double e;
01986 #if DEBUG
01987   fprintf (stderr, "calibrate_gradient_thread: start\n");
01988 #endif
01989   thread = data->thread;
01990 #if DEBUG
01991   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01992            thread,
01993            calibrate->thread_gradient[thread],
01994            calibrate->thread_gradient[thread + 1]);
01995 #endif
01996   for (i = calibrate->thread_gradient[thread];
01997        i < calibrate->thread_gradient[thread + 1]; ++i)
01998     {
01999       e = calibrate_norm (i);
02000       g_mutex_lock (mutex);
02001       calibrate_best_gradient (i, e);
02002       calibrate_save_variables (i, e);
02003       g_mutex_unlock (mutex);
02004 #if DEBUG
02005       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
02006 #endif
02007     }
02008 #if DEBUG
02009   fprintf (stderr, "calibrate_gradient_thread: end\n");
02010 #endif
02011   g_thread_exit (NULL);
02012   return NULL;
02013 }
```

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 1252 of file mpcotool.c.

```
01253 {
01254   unsigned int i;
01255   char buffer[32], value[32], *buffer2, *buffer3, *content;
01256   FILE *file;
01257   gsize length;
01258   GRegex *regex;
01259
01260 #if DEBUG
01261   fprintf (stderr, "calibrate_input: start\n");
01262 #endif
01263
01264   // Checking the file
01265   if (!template)
01266     goto calibrate_input_end;
01267
01268   // Opening template
01269   content = g_mapped_file_get_contents (template);
01270   length = g_mapped_file_get_length (template);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01273            content);
01274 #endif
01275   file = g_fopen (input, "w");
01276
01277   // Parsing template
01278   for (i = 0; i < calibrate->nvariables; ++i)
01279     {
01280 #if DEBUG
01281       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01282 #endif
01283       snprintf (buffer, 32, "@variable%u@", i + 1);
01284       regex = g_regex_new (buffer, 0, 0, NULL);
01285       if (i == 0)
01286         {
01287           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01288                                              calibrate->label[i], 0, NULL);
01289 #if DEBUG
01290           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01291 #endif
01292         }
01293       else
01294         {
01295           length = strlen (buffer3);
01296           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01297                                              calibrate->label[i], 0, NULL);
01298           g_free (buffer3);
01299         }
01300       g_regex_unref (regex);
01301       length = strlen (buffer2);
01302       snprintf (buffer, 32, "@value%u@", i + 1);
01303       regex = g_regex_new (buffer, 0, 0, NULL);
01304       snprintf (value, 32, format[calibrate->precision[i]],
01305                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01306
01307 #if DEBUG
01308       fprintf (stderr, "calibrate_input: value=%s\n", value);
01309 #endif
01310       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01311                                          0, NULL);
01312       g_free (buffer2);
01313       g_regex_unref (regex);
01314     }
01315
01316   // Saving input file
01317   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01318   g_free (buffer3);
01319   fclose (file);
01320
01321 calibrate_input_end:
01322 #if DEBUG
01323   fprintf (stderr, "calibrate_input: end\n");
01324 #endif
01325   return;
01326 }
```

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 1724 of file mpcotool.c.

```
01726 {
01727   unsigned int i, j, k, s[calibrate->nbest];
01728   double e[calibrate->nbest];
01729 #if DEBUG
01730   fprintf (stderr, "calibrate_merge: start\n");
01731 #endif
01732   i = j = k = 0;
01733   do
01734     {
01735       if (i == calibrate->nsaveds)
01736         {
01737           s[k] = simulation_best[j];
01738           e[k] = error_best[j];
01739           ++j;
01740           ++k;
01741           if (j == nsaveds)
01742             break;
01743         }
01744       else if (j == nsaveds)
01745         {
01746           s[k] = calibrate->simulation_best[i];
01747           e[k] = calibrate->error_best[i];
01748           ++i;
01749           ++k;
01750           if (i == calibrate->nsaveds)
01751             break;
01752         }
01753       else if (calibrate->error_best[i] > error_best[j])
01754         {
01755           s[k] = simulation_best[j];
01756           e[k] = error_best[j];
01757           ++j;
01758           ++k;
01759         }
01760       else
01761         {
01762           s[k] = calibrate->simulation_best[i];
01763           e[k] = calibrate->error_best[i];
01764           ++i;
01765           ++k;
01766         }
01767     }
01768   while (k < calibrate->nbest);
01769   calibrate->nsaveds = k;
01770   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01771   memcpy (calibrate->error_best, e, k * sizeof (double));
01772 #if DEBUG
01773   fprintf (stderr, "calibrate_merge: end\n");
01774 #endif
01775 }
```

5.5.2.10 double calibrate_norm_euclidian (unsigned int *simulation*)

Function to calculate the Euclidian error norm.

Parameters

simulation	simulation number.

Returns

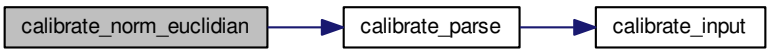
Euclidian error norm.

Definition at line 1443 of file mpcotool.c.

```
01444 {
01445   double e, ei;
```

```
01446   unsigned int i;
01447 #if DEBUG
01448   fprintf (stderr, "calibrate_norm_euclidian: start\n");
01449 #endif
01450   e = 0.;
01451   for (i = 0; i < calibrate->nexperiments; ++i)
01452     {
01453       ei = calibrate_parse (simulation, i);
01454       e += ei * ei;
01455     }
01456   e = sqrt (e);
01457 #if DEBUG
01458   fprintf (stderr, "calibrate_norm_euclidian: error=%lg\n", e);
01459   fprintf (stderr, "calibrate_norm_euclidian: end\n");
01460 #endif
01461   return e;
01462 }
```

Here is the call graph for this function:



5.5.2.11 double calibrate_norm_maximum (unsigned int *simulation*)

Function to calculate the maximum error norm.

Parameters

simulation	simulation number.

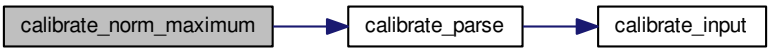
Returns

Maximum error norm.

Definition at line 1472 of file mpcotool.c.

```
01473 {
01474   double e, ei;
01475   unsigned int i;
01476 #if DEBUG
01477   fprintf (stderr, "calibrate_norm_maximum: start\n");
01478 #endif
01479   e = 0.;
01480   for (i = 0; i < calibrate->nexperiments; ++i)
01481     {
01482       ei = fabs (calibrate_parse (simulation, i));
01483       e = fmax (e, ei);
01484     }
01485 #if DEBUG
01486   fprintf (stderr, "calibrate_norm_maximum: error=%lg\n", e);
01487   fprintf (stderr, "calibrate_norm_maximum: end\n");
01488 #endif
01489   return e;
01490 }
```

Here is the call graph for this function:



5.5.2.12 double calibrate_norm_p (unsigned int *simulation*)

Function to calculate the P error norm.

Parameters

simulation	simulation number.

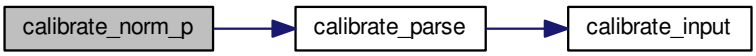
Returns

P error norm.

Definition at line 1500 of file mpcotool.c.

```
01501 {
01502   double e, ei;
01503   unsigned int i;
01504 #if DEBUG
01505   fprintf (stderr, "calibrate_norm_p: start\n");
01506 #endif
01507   e = 0.;
01508   for (i = 0; i < calibrate->nexperiments; ++i)
01509     {
01510       ei = fabs (calibrate_parse (simulation, i));
01511       e += pow (ei, calibrate->p);
01512     }
01513   e = pow (e, 1. / calibrate->p);
01514 #if DEBUG
01515   fprintf (stderr, "calibrate_norm_p: error=%lg\n", e);
01516   fprintf (stderr, "calibrate_norm_p: end\n");
01517 #endif
01518   return e;
01519 }
```

Here is the call graph for this function:



5.5.2.13 double calibrate_norm_taxicab (unsigned int *simulation*)

Function to calculate the taxicab error norm.

Parameters

simulation	simulation number.

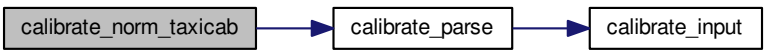
Returns

Taxicab error norm.

Definition at line 1529 of file mpcotool.c.

```
01530 {
01531   double e;
01532   unsigned int i;
01533 #if DEBUG
01534   fprintf (stderr, "calibrate_norm_taxicab: start\n");
01535 #endif
01536   e = 0.;
01537   for (i = 0; i < calibrate->nexperiments; ++i)
01538     e += fabs (calibrate_parse (simulation, i));
01539 #if DEBUG
01540   fprintf (stderr, "calibrate_norm_taxicab: error=%lg\n", e);
01541   fprintf (stderr, "calibrate_norm_taxicab: end\n");
01542 #endif
01543   return e;
01544 }
```

Here is the call graph for this function:



5.5.2.14 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 1339 of file mpcotool.c.

```
01340 {
01341   unsigned int i;
01342   double e;
01343   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01344     *buffer3, *buffer4;
01345   FILE *file_result;
01346
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_parse: start\n");
01349   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01350            experiment);
01351 #endif
01352
01353   // Opening input files
01354   for (i = 0; i < calibrate->ninputs; ++i)
```

```
01355     {
01356       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01357 #if DEBUG
01358       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01359 #endif
01360       calibrate_input (simulation, &input[i][0],
01361                        calibrate->file[i][experiment]);
01362     }
01363   for (; i < MAX_NINPUTS; ++i)
01364     strcpy (&input[i][0], "");
01365 #if DEBUG
01366   fprintf (stderr, "calibrate_parse: parsing end\n");
01367 #endif
01368
01369   // Performing the simulation
01370   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01371   buffer2 = g_path_get_dirname (calibrate->simulator);
01372   buffer3 = g_path_get_basename (calibrate->simulator);
01373   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01374   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01375             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01376             input[6], input[7], output);
01377   g_free (buffer4);
01378   g_free (buffer3);
01379   g_free (buffer2);
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01382 #endif
01383   system (buffer);
01384
01385   // Checking the objective value function
01386   if (calibrate->evaluator)
01387     {
01388       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01389       buffer2 = g_path_get_dirname (calibrate->evaluator);
01390       buffer3 = g_path_get_basename (calibrate->evaluator);
01391       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01392       snprintf (buffer, 512, "\"%s\" %s %s %s",
01393                 buffer4, output, calibrate->experiment[experiment], result);
01394       g_free (buffer4);
01395       g_free (buffer3);
01396       g_free (buffer2);
01397 #if DEBUG
01398       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01399 #endif
01400       system (buffer);
01401       file_result = g_fopen (result, "r");
01402       e = atof (fgets (buffer, 512, file_result));
01403       fclose (file_result);
01404     }
01405   else
01406     {
01407       strcpy (result, "");
01408       file_result = g_fopen (output, "r");
01409       e = atof (fgets (buffer, 512, file_result));
01410       fclose (file_result);
01411     }
01412
01413   // Removing files
01414 #if !DEBUG
01415   for (i = 0; i < calibrate->ninputs; ++i)
01416     {
01417       if (calibrate->file[i][0])
01418         {
01419           snprintf (buffer, 512, RM " %s", &input[i][0]);
01420           system (buffer);
01421         }
01422     }
01423   snprintf (buffer, 512, RM " %s %s", output, result);
01424   system (buffer);
01425 #endif
01426
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_parse: end\n");
01429 #endif
01430
01431   // Returning the objective function
01432   return e * calibrate->weight[experiment];
01433 }
```

Here is the call graph for this function:

5.5.2.15 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 1582 of file mpcotool.c.

```
01583 {
01584   unsigned int i;
01585   char buffer[64];
01586 #if DEBUG
01587   fprintf (stderr, "calibrate_save_variables: start\n");
01588 #endif
01589   for (i = 0; i < calibrate->nvariables; ++i)
01590     {
01591       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01592       fprintf (calibrate->file_variables, buffer,
01593                calibrate->value[simulation * calibrate->
      nvariables + i]);
01594     }
01595   fprintf (calibrate->file_variables, "%.14le\n", error);
01596 #if DEBUG
01597   fprintf (stderr, "calibrate_save_variables: end\n");
01598 #endif
01599 }
```

5.5.2.16 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

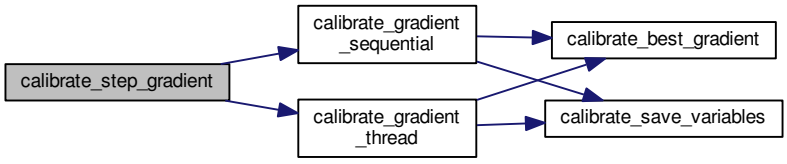
simulation	Simulation number.

Definition at line 2082 of file mpcotool.c.

```
02083 {
02084   GThread *thread[nthreads_gradient];
02085   ParallelData data[nthreads_gradient];
02086   unsigned int i, j, k, b;
02087 #if DEBUG
02088   fprintf (stderr, "calibrate_step_gradient: start\n");
02089 #endif
02090   for (i = 0; i < calibrate->nestimates; ++i)
02091     {
02092       k = (simulation + i) * calibrate->nvariables;
02093       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
02094 #if DEBUG
02095       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
02096                simulation + i, calibrate->simulation_best[0]);
02097 #endif
```

```
02098       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
02099         {
02100 #if DEBUG
02101           fprintf (stderr,
02102                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
02103                    i, j, calibrate->value[b]);
02104 #endif
02105           calibrate->value[k]
02106             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
02107           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
02108                                             calibrate->rangeminabs[j]),
02109                                       calibrate->rangemaxabs[j]);
02110 #if DEBUG
02111           fprintf (stderr,
02112                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
02113                    i, j, calibrate->value[k]);
02114 #endif
02115         }
02116     }
02117   if (nthreads_gradient == 1)
02118     calibrate_gradient_sequential (simulation);
02119   else
02120     {
02121       for (i = 0; i <= nthreads_gradient; ++i)
02122         {
02123           calibrate->thread_gradient[i]
02124             = simulation + calibrate->nstart_gradient
02125             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
02126             / nthreads_gradient;
02127 #if DEBUG
02128           fprintf (stderr,
02129                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
02130                    i, calibrate->thread_gradient[i]);
02131 #endif
02132         }
02133       for (i = 0; i < nthreads_gradient; ++i)
02134         {
02135           data[i].thread = i;
02136           thread[i] = g_thread_new
02137             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
02138         }
02139       for (i = 0; i < nthreads_gradient; ++i)
02140         g_thread_join (thread[i]);
02141     }
02142 #if DEBUG
02143   fprintf (stderr, "calibrate_step_gradient: end\n");
02144 #endif
02145 }
```

Here is the call graph for this function:



5.5.2.17 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

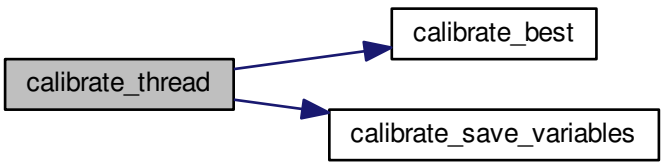
Returns

NULL

Definition at line 1682 of file mpcotool.c.

```
01683 {
01684   unsigned int i, thread;
01685   double e;
01686 #if DEBUG
01687   fprintf (stderr, "calibrate_thread: start\n");
01688 #endif
01689   thread = data->thread;
01690 #if DEBUG
01691   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01692            calibrate->thread[thread], calibrate->thread[thread + 1]);
01693 #endif
01694   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01695     {
01696       e = calibrate_norm (i);
01697       g_mutex_lock (mutex);
01698       calibrate_best (i, e);
01699       calibrate_save_variables (i, e);
01700       g_mutex_unlock (mutex);
01701 #if DEBUG
01702       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01703 #endif
01704     }
01705 #if DEBUG
01706   fprintf (stderr, "calibrate_thread: end\n");
01707 #endif
01708   g_thread_exit (NULL);
01709   return NULL;
01710 }
```

Here is the call graph for this function:



5.5.2.18 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 5147 of file mpcotool.c.

```
05148 {
05149 #ifdef G_OS_WIN32
```

```
05150   SYSTEM_INFO sysinfo;
05151   GetSystemInfo (&sysinfo);
05152   return sysinfo.dwNumberOfProcessors;
05153 #else
05154   return (int) sysconf (_SC_NPROCESSORS_ONLN);
05155 #endif
05156 }
```

5.5.2.19 unsigned int gtk_array_get_active (GtkRadioButton * *array[]*, unsigned int *n*)

Function to get the active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 486 of file mpcotool.c.

```
00487 {
00488   unsigned int i;
00489   for (i = 0; i < n; ++i)
00490     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00491       break;
00492   return i;
00493 }
```

5.5.2.20 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 574 of file mpcotool.c.

```
00575 {
00576   char buffer2[64];
00577   char *buffert[MAX_NINPUTS] =
00578     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00579   xmlDoc *doc;
00580   xmlNode *node, *child;
00581   xmlChar *buffer;
00582   char *msg;
00583   int error_code;
00584   unsigned int i;
00585
00586 #if DEBUG
00587   fprintf (stderr, "input_open: start\n");
00588 #endif
00589
00590   // Resetting input data
00591   buffer = NULL;
00592   input_new ();
00593
00594   // Parsing the input file
00595 #if DEBUG
00596   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00597 #endif
```

```
00598   doc = xmlParseFile (filename);
00599   if (!doc)
00600     {
00601       msg = gettext ("Unable to parse the input file");
00602       goto exit_on_error;
00603     }
00604
00605   // Getting the root node
00606 #if DEBUG
00607   fprintf (stderr, "input_open: getting the root node\n");
00608 #endif
00609   node = xmlDocGetRootElement (doc);
00610   if (xmlStrcmp (node->name, XML_CALIBRATE))
00611     {
00612       msg = gettext ("Bad root XML node");
00613       goto exit_on_error;
00614     }
00615
00616   // Getting results file names
00617   input->result = (char *) xmlGetProp (node, XML_RESULT);
00618   if (!input->result)
00619     input->result = (char *) xmlStrdup (result_name);
00620   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00621   if (!input->variables)
00622     input->variables = (char *) xmlStrdup (variables_name);
00623
00624   // Opening simulator program name
00625   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00626   if (!input->simulator)
00627     {
00628       msg = gettext ("Bad simulator program");
00629       goto exit_on_error;
00630     }
00631
00632   // Opening evaluator program name
00633   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00634
00635   // Obtaining pseudo-random numbers generator seed
00636   input->seed
00637     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00638                                       &error_code);
00639   if (error_code)
00640     {
00641       msg = gettext ("Bad pseudo-random numbers generator seed");
00642       goto exit_on_error;
00643     }
00644
00645   // Opening algorithm
00646   buffer = xmlGetProp (node, XML_ALGORITHM);
00647   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00648     {
00649       input->algorithm = ALGORITHM_MONTE_CARLO;
00650
00651       // Obtaining simulations number
00652       input->nsimulations
00653         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00654       if (error_code)
00655         {
00656           msg = gettext ("Bad simulations number");
00657           goto exit_on_error;
00658         }
00659     }
00660   else if (!xmlStrcmp (buffer, XML_SWEEP))
00661     input->algorithm = ALGORITHM_SWEEP;
00662   else if (!xmlStrcmp (buffer, XML_GENETIC))
00663     {
00664       input->algorithm = ALGORITHM_GENETIC;
00665
00666       // Obtaining population
00667       if (xmlHasProp (node, XML_NPOPULATION))
00668         {
00669           input->nsimulations
00670             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00671           if (error_code || input->nsimulations < 3)
00672             {
00673               msg = gettext ("Invalid population number");
00674               goto exit_on_error;
00675             }
00676         }
00677       else
00678         {
00679           msg = gettext ("No population number");
00680           goto exit_on_error;
00681         }
00682
00683       // Obtaining generations
```

```
00684       if (xmlHasProp (node, XML_NGENERATIONS))
00685         {
00686           input->niterations
00687             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00688           if (error_code || !input->niterations)
00689             {
00690               msg = gettext ("Invalid generations number");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         {
00696           msg = gettext ("No generations number");
00697           goto exit_on_error;
00698         }
00699
00700       // Obtaining mutation probability
00701       if (xmlHasProp (node, XML_MUTATION))
00702         {
00703           input->mutation_ratio
00704             = xml_node_get_float (node, XML_MUTATION, &error_code);
00705           if (error_code || input->mutation_ratio < 0.
00706               || input->mutation_ratio >= 1.)
00707             {
00708               msg = gettext ("Invalid mutation probability");
00709               goto exit_on_error;
00710             }
00711         }
00712       else
00713         {
00714           msg = gettext ("No mutation probability");
00715           goto exit_on_error;
00716         }
00717
00718       // Obtaining reproduction probability
00719       if (xmlHasProp (node, XML_REPRODUCTION))
00720         {
00721           input->reproduction_ratio
00722             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00723           if (error_code || input->reproduction_ratio < 0.
00724               || input->reproduction_ratio >= 1.0)
00725             {
00726               msg = gettext ("Invalid reproduction probability");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         {
00732           msg = gettext ("No reproduction probability");
00733           goto exit_on_error;
00734         }
00735
00736       // Obtaining adaptation probability
00737       if (xmlHasProp (node, XML_ADAPTATION))
00738         {
00739           input->adaptation_ratio
00740             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00741           if (error_code || input->adaptation_ratio < 0.
00742               || input->adaptation_ratio >= 1.)
00743             {
00744               msg = gettext ("Invalid adaptation probability");
00745               goto exit_on_error;
00746             }
00747         }
00748       else
00749         {
00750           msg = gettext ("No adaptation probability");
00751           goto exit_on_error;
00752         }
00753
00754       // Checking survivals
00755       i = input->mutation_ratio * input->nsimulations;
00756       i += input->reproduction_ratio * input->
     nsimulations;
00757       i += input->adaptation_ratio * input->
     nsimulations;
00758       if (i > input->nsimulations - 2)
00759         {
00760           msg = gettext
00761             ("No enough survival entities to reproduce the population");
00762           goto exit_on_error;
00763         }
00764     }
00765   else
00766     {
00767       msg = gettext ("Unknown algorithm");
00768       goto exit_on_error;
```

```
00769     }
00770   xmlFree (buffer);
00771   buffer = NULL;
00772
00773   if (input->algorithm == ALGORITHM_MONTE_CARLO
00774       || input->algorithm == ALGORITHM_SWEEP)
00775     {
00776
00777       // Obtaining iterations number
00778       input->niterations
00779         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00780       if (error_code == 1)
00781         input->niterations = 1;
00782       else if (error_code)
00783         {
00784           msg = gettext ("Bad iterations number");
00785           goto exit_on_error;
00786         }
00787
00788       // Obtaining best number
00789       input->nbest
00790         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00791       if (error_code || !input->nbest)
00792         {
00793           msg = gettext ("Invalid best number");
00794           goto exit_on_error;
00795         }
00796
00797       // Obtaining tolerance
00798       input->tolerance
00799         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00800                                            &error_code);
00801       if (error_code || input->tolerance < 0.)
00802         {
00803           msg = gettext ("Invalid tolerance");
00804           goto exit_on_error;
00805         }
00806
00807       // Getting gradient method parameters
00808       if (xmlHasProp (node, XML_NSTEPS))
00809         {
00810           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00811           if (error_code || !input->nsteps)
00812             {
00813               msg = gettext ("Invalid steps number");
00814               goto exit_on_error;
00815             }
00816           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00817           if (!xmlStrcmp (buffer, XML_COORDINATES))
00818             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00819           else if (!xmlStrcmp (buffer, XML_RANDOM))
00820             {
00821               input->gradient_method =
      GRADIENT_METHOD_RANDOM;
00822               input->nestimates
00823                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00824               if (error_code || !input->nestimates)
00825                 {
00826                   msg = gettext ("Invalid estimates number");
00827                   goto exit_on_error;
00828                 }
00829             }
00830           else
00831             {
00832               msg = gettext ("Unknown method to estimate the gradient");
00833               goto exit_on_error;
00834             }
00835           xmlFree (buffer);
00836           buffer = NULL;
00837           input->relaxation
00838             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00839                                                DEFAULT_RELAXATION, &error_code);
00840           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00841             {
00842               msg = gettext ("Invalid relaxation parameter");
00843               goto exit_on_error;
00844             }
00845         }
00846       else
00847         input->nsteps = 0;
00848     }
```

```
00849
00850   // Reading the experimental data
00851   for (child = node->children; child; child = child->next)
00852     {
00853       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00854         break;
00855 #if DEBUG
00856       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00857 #endif
00858       if (xmlHasProp (child, XML_NAME))
00859         buffer = xmlGetProp (child, XML_NAME);
00860       else
00861         {
00862           snprintf (buffer2, 64, "%s %u: %s",
00863                     gettext ("Experiment"),
00864                     input->nexperiments + 1, gettext ("no data file name"));
00865           msg = buffer2;
00866           goto exit_on_error;
00867         }
00868 #if DEBUG
00869       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00870 #endif
00871       input->weight = g_realloc (input->weight,
00872                                  (1 + input->nexperiments) * sizeof (double));
00873       input->weight[input->nexperiments]
00874         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00875       if (error_code)
00876         {
00877           snprintf (buffer2, 64, "%s %s: %s",
00878                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00879           msg = buffer2;
00880           goto exit_on_error;
00881         }
00882 #if DEBUG
00883       fprintf (stderr, "input_open: weight=%lg\n",
00884                input->weight[input->nexperiments]);
00885 #endif
00886       if (!input->nexperiments)
00887         input->ninputs = 0;
00888 #if DEBUG
00889       fprintf (stderr, "input_open: template[0]\n");
00890 #endif
00891       if (xmlHasProp (child, XML_TEMPLATE1))
00892         {
00893           input->template[0]
00894             = (char **) g_realloc (input->template[0],
00895                                    (1 + input->nexperiments) * sizeof (char *));
00896           buffert[0] = (char *) xmlGetProp (child, template[0]);
00897 #if DEBUG
00898           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00899                    input->nexperiments, buffert[0]);
00900 #endif
00901           if (!input->nexperiments)
00902             ++input->ninputs;
00903 #if DEBUG
00904           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00905 #endif
00906         }
00907       else
00908         {
00909           snprintf (buffer2, 64, "%s %s: %s",
00910                     gettext ("Experiment"), buffer, gettext ("no template"));
00911           msg = buffer2;
00912           goto exit_on_error;
00913         }
00914       for (i = 1; i < MAX_NINPUTS; ++i)
00915         {
00916 #if DEBUG
00917           fprintf (stderr, "input_open: template%u\n", i + 1);
00918 #endif
00919           if (xmlHasProp (child, template[i]))
00920             {
00921               if (input->nexperiments && input->ninputs <= i)
00922                 {
00923                   snprintf (buffer2, 64, "%s %s: %s",
00924                             gettext ("Experiment"),
00925                             buffer, gettext ("bad templates number"));
00926                   msg = buffer2;
00927                   while (i-- > 0)
00928                     xmlFree (buffert[i]);
00929                   goto exit_on_error;
00930                 }
00931               input->template[i] = (char **)
00932                 g_realloc (input->template[i],
00933                            (1 + input->nexperiments) * sizeof (char *));
00934               buffert[i] = (char *) xmlGetProp (child, template[i]);
```

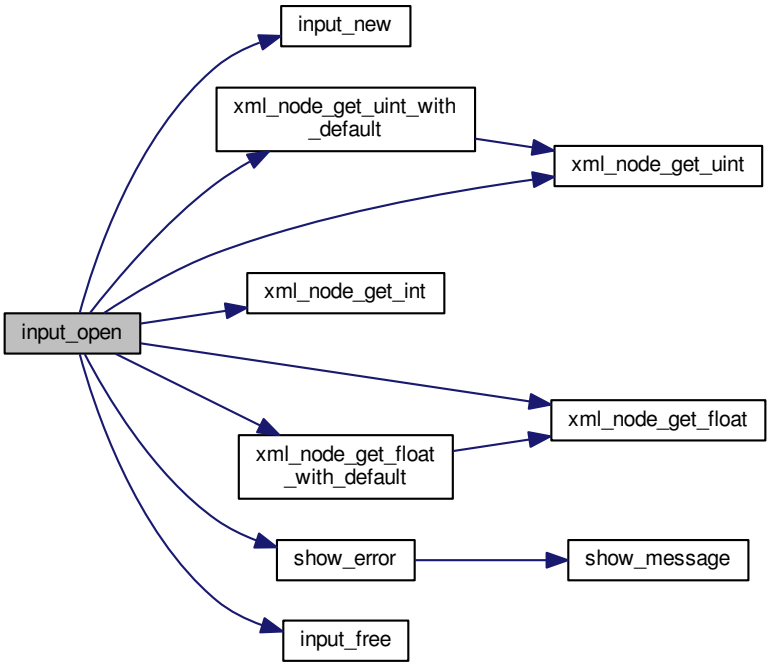
```
00935 #if DEBUG
00936               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00937                        input->nexperiments, i + 1,
00938                        input->template[i][input->nexperiments]);
00939 #endif
00940               if (!input->nexperiments)
00941                 ++input->ninputs;
00942 #if DEBUG
00943               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00944 #endif
00945             }
00946           else if (input->nexperiments && input->ninputs > i)
00947             {
00948               snprintf (buffer2, 64, "%s %s: %s%u",
00949                         gettext ("Experiment"),
00950                         buffer, gettext ("no template"), i + 1);
00951               msg = buffer2;
00952               while (i-- > 0)
00953                 xmlFree (buffert[i]);
00954               goto exit_on_error;
00955             }
00956           else
00957             break;
00958         }
00959       input->experiment
00960         = g_realloc (input->experiment,
00961                      (1 + input->nexperiments) * sizeof (char *));
00962       input->experiment[input->nexperiments] = (char *) buffer;
00963       for (i = 0; i < input->ninputs; ++i)
00964         input->template[i][input->nexperiments] = buffert[i];
00965       ++input->nexperiments;
00966 #if DEBUG
00967       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00968 #endif
00969     }
00970   if (!input->nexperiments)
00971     {
00972       msg = gettext ("No calibration experiments");
00973       goto exit_on_error;
00974     }
00975   buffer = NULL;
00976
00977   // Reading the variables data
00978   for (; child; child = child->next)
00979     {
00980       if (xmlStrcmp (child->name, XML_VARIABLE))
00981         {
00982           snprintf (buffer2, 64, "%s %u: %s",
00983                     gettext ("Variable"),
00984                     input->nvariables + 1, gettext ("bad XML node"));
00985           msg = buffer2;
00986           goto exit_on_error;
00987         }
00988       if (xmlHasProp (child, XML_NAME))
00989         buffer = xmlGetProp (child, XML_NAME);
00990       else
00991         {
00992           snprintf (buffer2, 64, "%s %u: %s",
00993                     gettext ("Variable"),
00994                     input->nvariables + 1, gettext ("no name"));
00995           msg = buffer2;
00996           goto exit_on_error;
00997         }
00998       if (xmlHasProp (child, XML_MINIMUM))
00999         {
01000           input->rangemin = g_realloc
01001             (input->rangemin, (1 + input->nvariables) * sizeof (double));
01002           input->rangeminabs = g_realloc
01003             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
01004           input->rangemin[input->nvariables]
01005             = xml_node_get_float (child, XML_MINIMUM, &error_code);
01006           if (error_code)
01007             {
01008               snprintf (buffer2, 64, "%s %s: %s",
01009                         gettext ("Variable"), buffer, gettext ("bad minimum"));
01010               msg = buffer2;
01011               goto exit_on_error;
01012             }
01013           input->rangeminabs[input->nvariables]
01014             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MINIMUM,
01015                                                -G_MAXDOUBLE, &error_code);
01016           if (error_code)
01017             {
01018               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01019                         gettext ("bad absolute minimum"));
01020               msg = buffer2;
```

```
01021               goto exit_on_error;
01022             }
01023           if (input->rangemin[input->nvariables]
01024               < input->rangeminabs[input->nvariables])
01025             {
01026               snprintf (buffer2, 64, "%s %s: %s",
01027                         gettext ("Variable"),
01028                         buffer, gettext ("minimum range not allowed"));
01029               msg = buffer2;
01030               goto exit_on_error;
01031             }
01032         }
01033       else
01034         {
01035           snprintf (buffer2, 64, "%s %s: %s",
01036                     gettext ("Variable"), buffer, gettext ("no minimum range"));
01037           msg = buffer2;
01038           goto exit_on_error;
01039         }
01040       if (xmlHasProp (child, XML_MAXIMUM))
01041         {
01042           input->rangemax = g_realloc
01043             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01044           input->rangemaxabs = g_realloc
01045             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01046           input->rangemax[input->nvariables]
01047             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01048           if (error_code)
01049             {
01050               snprintf (buffer2, 64, "%s %s: %s",
01051                         gettext ("Variable"), buffer, gettext ("bad maximum"));
01052               msg = buffer2;
01053               goto exit_on_error;
01054             }
01055           input->rangemaxabs[input->nvariables]
01056             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MAXIMUM,
01057                                                G_MAXDOUBLE, &error_code);
01058           if (error_code)
01059             {
01060               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01061                         gettext ("bad absolute maximum"));
01062               msg = buffer2;
01063               goto exit_on_error;
01064             }
01065           if (input->rangemax[input->nvariables]
01066               > input->rangemaxabs[input->nvariables])
01067             {
01068               snprintf (buffer2, 64, "%s %s: %s",
01069                         gettext ("Variable"),
01070                         buffer, gettext ("maximum range not allowed"));
01071               msg = buffer2;
01072               goto exit_on_error;
01073             }
01074         }
01075       else
01076         {
01077           snprintf (buffer2, 64, "%s %s: %s",
01078                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01079           msg = buffer2;
01080           goto exit_on_error;
01081         }
01082       if (input->rangemax[input->nvariables]
01083           < input->rangemin[input->nvariables])
01084         {
01085           snprintf (buffer2, 64, "%s %s: %s",
01086                     gettext ("Variable"), buffer, gettext ("bad range"));
01087           msg = buffer2;
01088           goto exit_on_error;
01089         }
01090       input->precision = g_realloc
01091         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01092       input->precision[input->nvariables]
01093         = xml_node_get_uint_with_default (child,
    XML_PRECISION,
01094                                           DEFAULT_PRECISION, &error_code);
01095       if (error_code || input->precision[input->nvariables] >=
    NPRECISIONS)
01096         {
01097           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                     gettext ("bad precision"));
01099           msg = buffer2;
01100           goto exit_on_error;
01101         }
01102       if (input->algorithm == ALGORITHM_SWEEP)
01103         {
01104           if (xmlHasProp (child, XML_NSWEEPS))
```

```
01105             {
01106               input->nsweeps = (unsigned int *)
01107                 g_realloc (input->nsweeps,
01108                            (1 + input->nvariables) * sizeof (unsigned int));
01109               input->nsweeps[input->nvariables]
01110                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01111               if (error_code || !input->nsweeps[input->
      nvariables])
01112                 {
01113                   snprintf (buffer2, 64, "%s %s: %s",
01114                             gettext ("Variable"),
01115                             buffer, gettext ("bad sweeps"));
01116                   msg = buffer2;
01117                   goto exit_on_error;
01118                 }
01119             }
01120           else
01121             {
01122               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01123                         gettext ("no sweeps number"));
01124               msg = buffer2;
01125               goto exit_on_error;
01126             }
01127 #if DEBUG
01128           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01129                    input->nsweeps[input->nvariables],
      input->nsimulations);
01130 #endif
01131         }
01132       if (input->algorithm == ALGORITHM_GENETIC)
01133         {
01134           // Obtaining bits representing each variable
01135           if (xmlHasProp (child, XML_NBITS))
01136             {
01137               input->nbits = (unsigned int *)
01138                 g_realloc (input->nbits,
01139                            (1 + input->nvariables) * sizeof (unsigned int));
01140               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01141               if (error_code || !i)
01142                 {
01143                   snprintf (buffer2, 64, "%s %s: %s",
01144                             gettext ("Variable"),
01145                             buffer, gettext ("invalid bits number"));
01146                   msg = buffer2;
01147                   goto exit_on_error;
01148                 }
01149               input->nbits[input->nvariables] = i;
01150             }
01151           else
01152             {
01153               snprintf (buffer2, 64, "%s %s: %s",
01154                         gettext ("Variable"),
01155                         buffer, gettext ("no bits number"));
01156               msg = buffer2;
01157               goto exit_on_error;
01158             }
01159         }
01160       else if (input->nsteps)
01161         {
01162           input->step = (double *)
01163             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01164           input->step[input->nvariables]
01165             = xml_node_get_float (child, XML_STEP, &error_code);
01166           if (error_code || input->step[input->nvariables] < 0.)
01167             {
01168               snprintf (buffer2, 64, "%s %s: %s",
01169                         gettext ("Variable"),
01170                         buffer, gettext ("bad step size"));
01171               msg = buffer2;
01172               goto exit_on_error;
01173             }
01174         }
01175       input->label = g_realloc
01176         (input->label, (1 + input->nvariables) * sizeof (char *));
01177       input->label[input->nvariables] = (char *) buffer;
01178       ++input->nvariables;
01179     }
01180   if (!input->nvariables)
01181     {
01182       msg = gettext ("No calibration variables");
01183       goto exit_on_error;
01184     }
01185   buffer = NULL;
01186
01187   // Obtaining the error norm
01188   if (xmlHasProp (node, XML_NORM))
01189     {
```

```
01190       buffer = xmlGetProp (node, XML_NORM);
01191       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
01192         input->norm = ERROR_NORM_EUCLIDIAN;
01193       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
01194         input->norm = ERROR_NORM_MAXIMUM;
01195       else if (!xmlStrcmp (buffer, XML_P))
01196         {
01197           input->norm = ERROR_NORM_P;
01198           input->p = xml_node_get_float (node, XML_P, &error_code);
01199           if (!error_code)
01200             {
01201               msg = gettext ("Bad P parameter");
01202               goto exit_on_error;
01203             }
01204         }
01205       else if (!xmlStrcmp (buffer, XML_TAXICAB))
01206         input->norm = ERROR_NORM_TAXICAB;
01207       else
01208         {
01209           msg = gettext ("Unknown error norm");
01210           goto exit_on_error;
01211         }
01212       xmlFree (buffer);
01213     }
01214   else
01215     input->norm = ERROR_NORM_EUCLIDIAN;
01216
01217   // Getting the working directory
01218   input->directory = g_path_get_dirname (filename);
01219   input->name = g_path_get_basename (filename);
01220
01221   // Closing the XML document
01222   xmlFreeDoc (doc);
01223
01224 #if DEBUG
01225   fprintf (stderr, "input_open: end\n");
01226 #endif
01227   return 1;
01228
01229 exit_on_error:
01230   xmlFree (buffer);
01231   xmlFreeDoc (doc);
01232   show_error (msg);
01233   input_free ();
01234 #if DEBUG
01235   fprintf (stderr, "input_open: end\n");
01236 #endif
01237   return 0;
01238 }
```

Here is the call graph for this function:



5.5.2.21 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

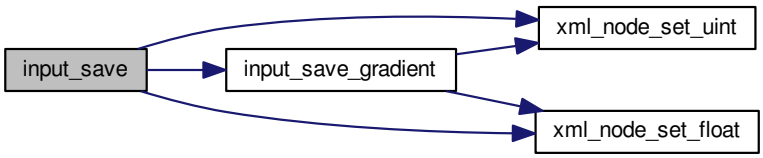
Definition at line 2874 of file mpcotool.c.

```
02875 {
02876   unsigned int i, j;
02877   char *buffer;
02878   xmlDoc *doc;
02879   xmlNode *node, *child;
02880   GFile *file, *file2;
02881
02882 #if DEBUG
02883   fprintf (stderr, "input_save: start\n");
02884 #endif
02885
02886   // Getting the input file directory
02887   input->name = g_path_get_basename (filename);
02888   input->directory = g_path_get_dirname (filename);
02889   file = g_file_new_for_path (input->directory);
02890
02891   // Opening the input file
02892   doc = xmlNewDoc ((const xmlChar *) "1.0");
02893
02894   // Setting root XML node
02895   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02896   xmlDocSetRootElement (doc, node);
02897
02898   // Adding properties to the root XML node
02899   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02900     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02901   if (xmlStrcmp ((const xmlChar *) input->variables,
```

```
      variables_name))
02902     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02903   file2 = g_file_new_for_path (input->simulator);
02904   buffer = g_file_get_relative_path (file, file2);
02905   g_object_unref (file2);
02906   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02907   g_free (buffer);
02908   if (input->evaluator)
02909     {
02910       file2 = g_file_new_for_path (input->evaluator);
02911       buffer = g_file_get_relative_path (file, file2);
02912       g_object_unref (file2);
02913       if (xmlStrlen ((xmlChar *) buffer))
02914         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02915       g_free (buffer);
02916     }
02917   if (input->seed != DEFAULT_RANDOM_SEED)
02918     xml_node_set_uint (node, XML_SEED, input->seed);
02919
02920   // Setting the algorithm
02921   buffer = (char *) g_malloc (64);
02922   switch (input->algorithm)
02923     {
02924     case ALGORITHM_MONTE_CARLO:
02925       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02926       snprintf (buffer, 64, "%u", input->nsimulations);
02927       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02928       snprintf (buffer, 64, "%u", input->niterations);
02929       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02930       snprintf (buffer, 64, "%.3lg", input->tolerance);
02931       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02932       snprintf (buffer, 64, "%u", input->nbest);
02933       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02934       input_save_gradient (node);
02935       break;
02936     case ALGORITHM_SWEEP:
02937       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02938       snprintf (buffer, 64, "%u", input->niterations);
02939       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02940       snprintf (buffer, 64, "%.3lg", input->tolerance);
02941       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02942       snprintf (buffer, 64, "%u", input->nbest);
02943       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02944       input_save_gradient (node);
02945       break;
02946     default:
02947       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02948       snprintf (buffer, 64, "%u", input->nsimulations);
02949       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02950       snprintf (buffer, 64, "%u", input->niterations);
02951       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02952       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02953       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02954       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02955       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02956       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02957       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02958       break;
02959     }
02960   g_free (buffer);
02961
02962   // Setting the experimental data
02963   for (i = 0; i < input->nexperiments; ++i)
02964     {
02965       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02966       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02967       if (input->weight[i] != 1.)
02968         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02969       for (j = 0; j < input->ninputs; ++j)
02970         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02971     }
02972
02973   // Setting the variables data
02974   for (i = 0; i < input->nvariables; ++i)
02975     {
02976       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02977       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02978       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02979       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02980         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02981       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02982       if (input->rangemaxabs[i] != G_MAXDOUBLE)
```

```
02983         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02984       if (input->precision[i] != DEFAULT_PRECISION)
02985         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02986       if (input->algorithm == ALGORITHM_SWEEP)
02987         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02988       else if (input->algorithm == ALGORITHM_GENETIC)
02989         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02990       if (input->nsteps)
02991         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02992     }
02993
02994   // Saving the error norm
02995   switch (input->norm)
02996     {
02997     case ERROR_NORM_MAXIMUM:
02998       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
02999       break;
03000     case ERROR_NORM_P:
03001       xmlSetProp (node, XML_NORM, XML_P);
03002       xml_node_set_float (node, XML_P, input->p);
03003       break;
03004     case ERROR_NORM_TAXICAB:
03005       xmlSetProp (node, XML_NORM, XML_TAXICAB);
03006     }
03007
03008   // Saving the XML file
03009   xmlSaveFormatFile (filename, doc, 1);
03010
03011   // Freeing memory
03012   xmlFreeDoc (doc);
03013
03014 #if DEBUG
03015   fprintf (stderr, "input_save: end\n");
03016 #endif
03017 }
```

Here is the call graph for this function:



5.5.2.22 void input_save_gradient (xmlNode * *node*)

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

Parameters

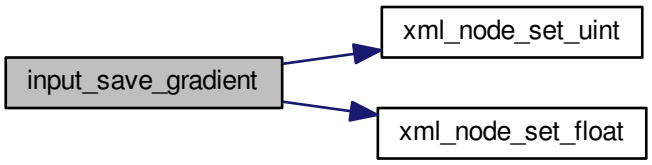
node	XML node.

Definition at line 2842 of file mpcotool.c.

```
02843 {
02844 #if DEBUG
02845   fprintf (stderr, "input_save_gradient: start\n");
02846 #endif
02847   if (input->nsteps)
02848     {
02849       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
```

```
02850       if (input->relaxation != DEFAULT_RELAXATION)
02851         xml_node_set_float (node, XML_RELAXATION,
      input->relaxation);
02852       switch (input->gradient_method)
02853         {
02854         case GRADIENT_METHOD_COORDINATES:
02855           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02856           break;
02857         default:
02858           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02859           xml_node_set_uint (node, XML_NESTIMATES,
      input->nestimates);
02860         }
02861     }
02862 #if DEBUG
02863   fprintf (stderr, "input_save_gradient: end\n");
02864 #endif
02865 }
```

Here is the call graph for this function:



5.5.2.23 int main (int *argn*, char ** *argc*)

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

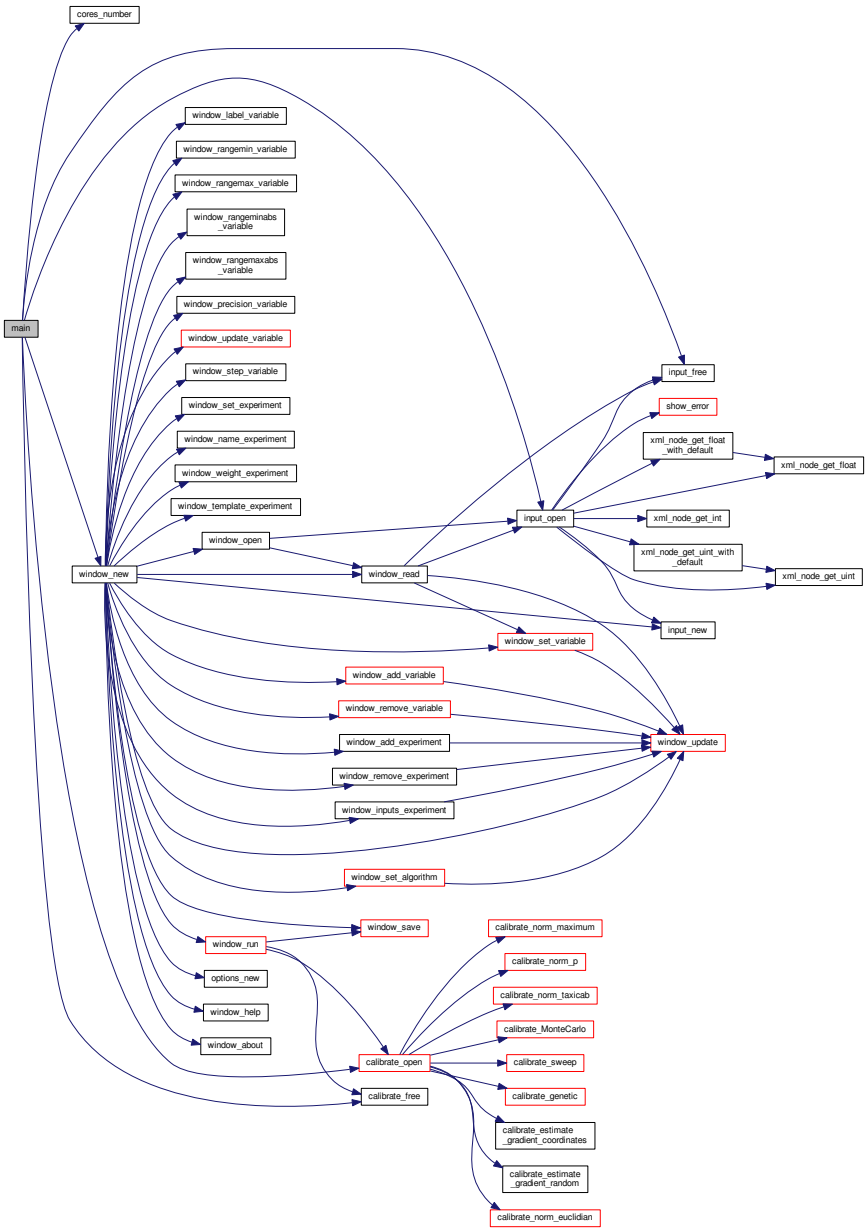
0 on success, >0 on error.

Definition at line 5168 of file mpcotool.c.

```
05169 {
05170 #if HAVE_GTK
05171   char *buffer;
05172 #endif
05173
05174   // Starting pseudo-random numbers generator
05175   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
05176   calibrate->seed = DEFAULT_RANDOM_SEED;
05177
05178   // Allowing spaces in the XML data file
05179   xmlKeepBlanksDefault (0);
05180
05181   // Starting MPI
05182 #if HAVE_MPI
05183   MPI_Init (&argn, &argc);
05184   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
05185   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
05186   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
05187 #else
```

```
05188   ntasks = 1;
05189 #endif
05190
05191 #if HAVE_GTK
05192
05193   // Getting threads number
05194   nthreads_gradient = nthreads = cores_number ();
05195
05196   // Setting local language and international floating point numbers notation
05197   setlocale (LC_ALL, "");
05198   setlocale (LC_NUMERIC, "C");
05199   window->application_directory = g_get_current_dir ();
05200   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
05201   bindtextdomain (PROGRAM_INTERFACE, buffer);
05202   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
05203   textdomain (PROGRAM_INTERFACE);
05204
05205   // Initing GTK+
05206   gtk_disable_setlocale ();
05207   gtk_init (&argn, &argc);
05208
05209   // Opening the main window
05210   window_new ();
05211   gtk_main ();
05212
05213   // Freeing memory
05214   input_free ();
05215   g_free (buffer);
05216   gtk_widget_destroy (GTK_WIDGET (window->window));
05217   g_free (window->application_directory);
05218
05219 #else
05220
05221   // Checking syntax
05222   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
05223     {
05224       printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
05225       return 1;
05226     }
05227
05228   // Getting threads number
05229   if (argn == 2)
05230     nthreads_gradient = nthreads = cores_number ();
05231   else
05232     {
05233       nthreads_gradient = nthreads = atoi (argc[2]);
05234       if (!nthreads)
05235         {
05236           printf ("Bad threads number\n");
05237           return 2;
05238         }
05239     }
05240   printf ("nthreads=%u\n", nthreads);
05241
05242   // Making calibration
05243   if (input_open (argc[argn - 1]))
05244     calibrate_open ();
05245
05246   // Freeing memory
05247   calibrate_free ();
05248
05249 #endif
05250
05251   // Closing MPI
05252 #if HAVE_MPI
05253   MPI_Finalize ();
05254 #endif
05255
05256   // Freeing memory
05257   gsl_rng_free (calibrate->rng);
05258
05259   // Closing
05260   return 0;
05261 }
```

Here is the call graph for this function:



5.5.2.24 void show_error (char * *msg*)

Function to show a dialog with an error message.

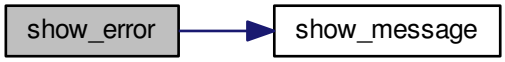
Parameters

msg	Error message.

Definition at line 259 of file mpcotool.c.

```
00260 {
00261   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00262 }
```

Here is the call graph for this function:



5.5.2.25 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 229 of file mpcotool.c.

```
00230 {
00231 #if HAVE_GTK
00232   GtkMessageDialog *dlg;
00233
00234   // Creating the dialog
00235   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00236     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00237
00238   // Setting the dialog title
00239   gtk_window_set_title (GTK_WINDOW (dlg), title);
00240
00241   // Showing the dialog and waiting response
00242   gtk_dialog_run (GTK_DIALOG (dlg));
00243
00244   // Closing and freeing memory
00245   gtk_widget_destroy (GTK_WIDGET (dlg));
00246
00247 #else
00248   printf ("%s: %s\n", title, msg);
00249 #endif
00250 }
```

5.5.2.26 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 3121 of file mpcotool.c.

```
03122 {
03123   unsigned int i;
03124 #if DEBUG
03125   fprintf (stderr, "window_get_algorithm: start\n");
03126 #endif
03127   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
03128 #if DEBUG
03129   fprintf (stderr, "window_get_algorithm: %u\n", i);
03130   fprintf (stderr, "window_get_algorithm: end\n");
```

```
03131 #endif
03132   return i;
03133 }
```

Here is the call graph for this function:

5.5.2.27 unsigned int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 3141 of file mpcotool.c.

```
03142 {
03143   unsigned int i;
03144 #if DEBUG
03145   fprintf (stderr, "window_get_gradient: start\n");
03146 #endif
03147   i = gtk_array_get_active (window->button_gradient ,
      NGRADIENTS);
03148 #if DEBUG
03149   fprintf (stderr, "window_get_gradient: %u\n", i);
03150   fprintf (stderr, "window_get_gradient: end\n");
03151 #endif
03152   return i;
03153 }
```

Here is the call graph for this function:

5.5.2.28 unsigned int window_get_norm ( )

Function to get the norm base method number.

Returns

Gradient base method number.

Definition at line 3161 of file mpcotool.c.

```
03162 {
03163   unsigned int i;
03164 #if DEBUG
03165   fprintf (stderr, "window_get_norm: start\n");
03166 #endif
03167   i = gtk_array_get_active (window->button_norm ,
      NNORMS);
03168 #if DEBUG
03169   fprintf (stderr, "window_get_norm: %u\n", i);
03170   fprintf (stderr, "window_get_norm: end\n");
03171 #endif
03172   return i;
03173 }
```

Here is the call graph for this function:

5.5.2.29 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.

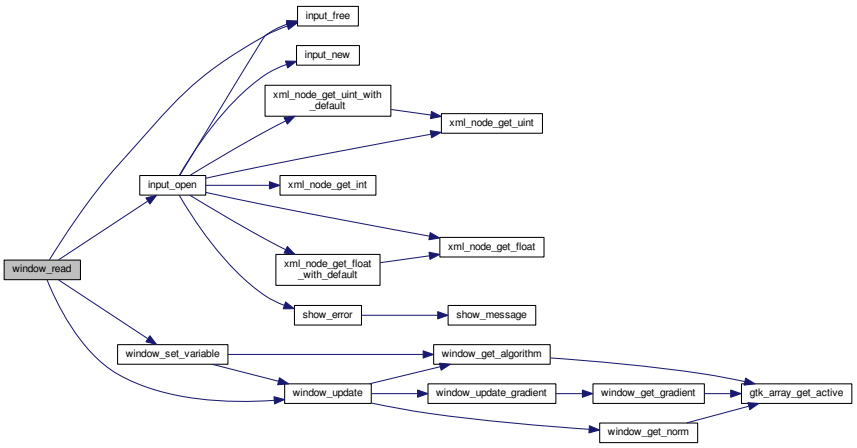
Definition at line 4264 of file mpcotool.c.

```
04265 {
04266   unsigned int i;
04267   char *buffer;
04268 #if DEBUG
04269   fprintf (stderr, "window_read: start\n");
04270 #endif
04271
04272   // Reading new input file
04273   input_free ();
04274   if (!input_open (filename))
04275     return 0;
04276
04277   // Setting GTK+ widgets data
04278   gtk_entry_set_text (window->entry_result, input->result);
04279   gtk_entry_set_text (window->entry_variables, input->
      variables);
04280   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04282                                  (window->button_simulator), buffer);
04283   g_free (buffer);
04284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04285                                 (size_t) input->evaluator);
```

```
04286   if (input->evaluator)
04287     {
04288       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04290                                      (window->button_evaluator), buffer);
04291       g_free (buffer);
04292     }
04293   gtk_toggle_button_set_active
04294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04295   switch (input->algorithm)
04296     {
04297     case ALGORITHM_MONTE_CARLO:
04298       gtk_spin_button_set_value (window->spin_simulations,
04299                                  (gdouble) input->nsimulations);
04300     case ALGORITHM_SWEEP:
04301       gtk_spin_button_set_value (window->spin_iterations,
04302                                  (gdouble) input->niterations);
04303       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04304       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04305       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04306                                     input->nsteps);
04307       if (input->nsteps)
04308         {
04309           gtk_toggle_button_set_active
04310             (GTK_TOGGLE_BUTTON (window->button_gradient
04311                                 [input->gradient_method]), TRUE);
04312           gtk_spin_button_set_value (window->spin_steps,
04313                                      (gdouble) input->nsteps);
04314           gtk_spin_button_set_value (window->spin_relaxation,
04315                                      (gdouble) input->relaxation);
04316           switch (input->gradient_method)
04317             {
04318             case GRADIENT_METHOD_RANDOM:
04319               gtk_spin_button_set_value (window->spin_estimates,
04320                                          (gdouble) input->nestimates);
04321             }
04322         }
04323       break;
04324     default:
04325       gtk_spin_button_set_value (window->spin_population,
04326                                  (gdouble) input->nsimulations);
04327       gtk_spin_button_set_value (window->spin_generations,
04328                                  (gdouble) input->niterations);
04329       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04330       gtk_spin_button_set_value (window->spin_reproduction,
04331                                  input->reproduction_ratio);
04332       gtk_spin_button_set_value (window->spin_adaptation,
04333                                  input->adaptation_ratio);
04334     }
04335   gtk_toggle_button_set_active
04336     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
04337   gtk_spin_button_set_value (window->spin_p, input->p);
04338   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04339   g_signal_handler_block (window->button_experiment,
04340                           window->id_experiment_name);
04341   gtk_combo_box_text_remove_all (window->combo_experiment);
04342   for (i = 0; i < input->nexperiments; ++i)
04343     gtk_combo_box_text_append_text (window->combo_experiment,
04344                                     input->experiment[i]);
04345   g_signal_handler_unblock
04346     (window->button_experiment, window->
      id_experiment_name);
04347   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04348   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04349   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04350   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04351   gtk_combo_box_text_remove_all (window->combo_variable);
04352   for (i = 0; i < input->nvariables; ++i)
04353     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
04354   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04355   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04356   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04357   window_set_variable ();
04358   window_update ();
```

```
04359
04360 #if DEBUG
04361   fprintf (stderr, "window_read: end\n");
04362 #endif
04363   return 1;
04364 }
```

Here is the call graph for this function:



5.5.2.30 int window_save ( )

Function to save the input file.

Returns

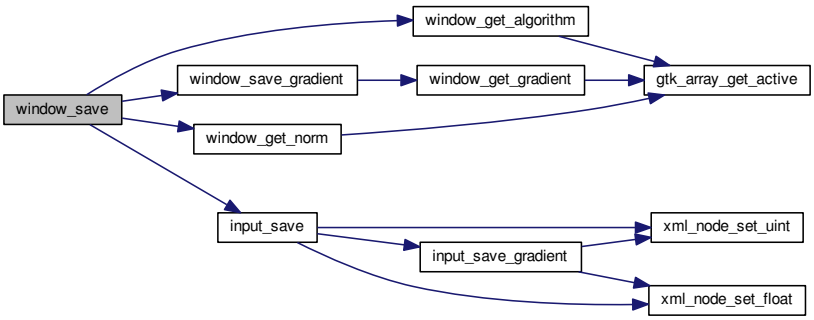
1 on OK, 0 on Cancel.

Definition at line 3213 of file mpcotool.c.

```
03214 {
03215   GtkFileChooserDialog *dlg;
03216   GtkFileFilter *filter;
03217   char *buffer;
03218
03219 #if DEBUG
03220   fprintf (stderr, "window_save: start\n");
03221 #endif
03222
03223   // Opening the saving dialog
03224   dlg = (GtkFileChooserDialog *)
03225     gtk_file_chooser_dialog_new (gettext ("Save file"),
03226                                  window->window,
03227                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03228                                  gettext ("_Cancel"),
03229                                  GTK_RESPONSE_CANCEL,
03230                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03231   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03232   buffer = g_build_filename (input->directory, input->name, NULL);
03233   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03234   g_free (buffer);
03235
03236   // Adding XML filter
03237   filter = (GtkFileFilter *) gtk_file_filter_new ();
03238   gtk_file_filter_set_name (filter, "XML");
03239   gtk_file_filter_add_pattern (filter, "*.xml");
03240   gtk_file_filter_add_pattern (filter, "*.XML");
03241   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03242
03243   // If OK response then saving
03244   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
```

```
03245     {
03246
03247       // Adding properties to the root XML node
03248       input->simulator = gtk_file_chooser_get_filename
03249         (GTK_FILE_CHOOSER (window->button_simulator));
03250       if (gtk_toggle_button_get_active
03251           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03252         input->evaluator = gtk_file_chooser_get_filename
03253           (GTK_FILE_CHOOSER (window->button_evaluator));
03254       else
03255         input->evaluator = NULL;
03256       input->result
03257         = (char *) xmlStrdup ((const xmlChar *)
03258                               gtk_entry_get_text (window->entry_result));
03259       input->variables
03260         = (char *) xmlStrdup ((const xmlChar *)
03261                               gtk_entry_get_text (window->entry_variables));
03262
03263       // Setting the algorithm
03264       switch (window_get_algorithm ())
03265         {
03266         case ALGORITHM_MONTE_CARLO:
03267           input->algorithm = ALGORITHM_MONTE_CARLO;
03268           input->nsimulations
03269             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03270           input->niterations
03271             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03272           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03273           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03274           window_save_gradient ();
03275           break;
03276         case ALGORITHM_SWEEP:
03277           input->algorithm = ALGORITHM_SWEEP;
03278           input->niterations
03279             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03280           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03281           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03282           window_save_gradient ();
03283           break;
03284         default:
03285           input->algorithm = ALGORITHM_GENETIC;
03286           input->nsimulations
03287             = gtk_spin_button_get_value_as_int (window->spin_population);
03288           input->niterations
03289             = gtk_spin_button_get_value_as_int (window->spin_generations);
03290           input->mutation_ratio
03291             = gtk_spin_button_get_value (window->spin_mutation);
03292           input->reproduction_ratio
03293             = gtk_spin_button_get_value (window->spin_reproduction);
03294           input->adaptation_ratio
03295             = gtk_spin_button_get_value (window->spin_adaptation);
03296           break;
03297         }
03298       input->norm = window_get_norm ();
03299       input->p = gtk_spin_button_get_value (window->spin_p);
03300
03301       // Saving the XML file
03302       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03303       input_save (buffer);
03304
03305       // Closing and freeing memory
03306       g_free (buffer);
03307       gtk_widget_destroy (GTK_WIDGET (dlg));
03308 #if DEBUG
03309       fprintf (stderr, "window_save: end\n");
03310 #endif
03311       return 1;
03312     }
03313
03314   // Closing and freeing memory
03315   gtk_widget_destroy (GTK_WIDGET (dlg));
03316 #if DEBUG
03317   fprintf (stderr, "window_save: end\n");
03318 #endif
03319   return 0;
03320 }
```

Here is the call graph for this function:



5.5.2.31 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 3868 of file mpcotool.c.

```
03869 {
03870   unsigned int i, j;
03871   char *buffer;
03872   GFile *file1, *file2;
03873 #if DEBUG
03874   fprintf (stderr, "window_template_experiment: start\n");
03875 #endif
03876   i = (size_t) data;
03877   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03878   file1
03879     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03880   file2 = g_file_new_for_path (input->directory);
03881   buffer = g_file_get_relative_path (file2, file1);
03882   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03883   g_free (buffer);
03884   g_object_unref (file2);
03885   g_object_unref (file1);
03886 #if DEBUG
03887   fprintf (stderr, "window_template_experiment: end\n");
03888 #endif
03889 }
```

5.5.2.32 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 369 of file mpcotool.c.

```
00370 {
00371   double x = 0.;
00372   xmlChar *buffer;
00373   buffer = xmlGetProp (node, prop);
00374   if (!buffer)
00375     *error_code = 1;
00376   else
00377     {
00378       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00379         *error_code = 2;
00380       else
00381         *error_code = 0;
00382       xmlFree (buffer);
00383     }
00384   return x;
00385 }
```

5.5.2.33 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.

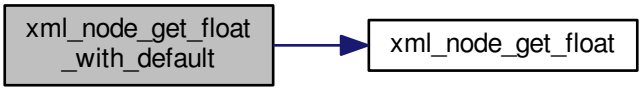
Returns

Floating point number value.

Definition at line 403 of file mpcotool.c.

```
00405 {
00406   double x;
00407   if (xmlHasProp (node, prop))
00408     x = xml_node_get_float (node, prop, error_code);
00409   else
00410     {
00411       x = default_value;
00412       *error_code = 0;
00413     }
00414   return x;
00415 }
```

Here is the call graph for this function:



5.5.2.34 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 277 of file mpcotool.c.

```
00278 {
00279   int i = 0;
00280   xmlChar *buffer;
00281   buffer = xmlGetProp (node, prop);
00282   if (!buffer)
00283     *error_code = 1;
00284   else
00285     {
00286       if (sscanf ((char *) buffer, "%d", &i) != 1)
00287         *error_code = 2;
00288       else
00289         *error_code = 0;
00290       xmlFree (buffer);
00291     }
00292   return i;
00293 }
```

5.5.2.35 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 308 of file mpcotool.c.

```
00309 {
00310   unsigned int i = 0;
00311   xmlChar *buffer;
00312   buffer = xmlGetProp (node, prop);
00313   if (!buffer)
00314     *error_code = 1;
00315   else
00316     {
00317       if (sscanf ((char *) buffer, "%u", &i) != 1)
00318         *error_code = 2;
00319       else
00320         *error_code = 0;
00321       xmlFree (buffer);
00322     }
00323   return i;
00324 }
```

5.5.2.36 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.

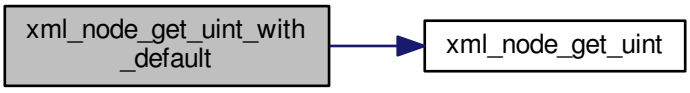
Returns

Unsigned integer number value.

Definition at line 342 of file mpcotool.c.

```
00344 {
00345   unsigned int i;
00346   if (xmlHasProp (node, prop))
00347     i = xml_node_get_uint (node, prop, error_code);
00348   else
00349     {
00350       i = default_value;
00351       *error_code = 0;
00352     }
00353   return i;
00354 }
```

Here is the call graph for this function:



5.5.2.37 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 466 of file mpcotool.c.

```
00467 {
00468   xmlChar buffer[64];
00469   snprintf ((char *) buffer, 64, "%.14lg", value);
00470   xmlSetProp (node, prop, buffer);
00471 }
```

5.5.2.38 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 428 of file mpcotool.c.

```
00429 {
00430   xmlChar buffer[64];
00431   snprintf ((char *) buffer, 64, "%d", value);
00432   xmlSetProp (node, prop, buffer);
00433 }
```

5.5.2.39 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 447 of file mpcotool.c.

```
00448 {
00449   xmlChar buffer[64];
00450   snprintf ((char *) buffer, 64, "%u", value);
00451   xmlSetProp (node, prop, buffer);
00452 }
```

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 120 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 125 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 113 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 2012-2016, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #define _GNU_SOURCE
00037 #include "config.h"
00038 #include <stdio.h>
00039 #include <stdlib.h>
00040 #include <string.h>
00041 #include <math.h>
00042 #include <unistd.h>
00043 #include <locale.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elifndef (BSD)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "mpcotool.h"
00059 #if HAVE_GTK
00060 #include <gio/gio.h>
00061 #include <gtk/gtk.h>
00062 #include "interface.h"
00063 #endif
00064
00065 #define DEBUG 0
00066
00067
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
```

```
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00092 int ntasks;
00093 unsigned int nthreads;
00094 unsigned int nthreads_gradient;
00096 GMutex mutex[1];
00097 void (*calibrate_algorithm) ();
00099 double (*calibrate_estimate_gradient) (unsigned int variable,
00100                                        unsigned int estimate);
00102 double (*calibrate_norm) (unsigned int simulation);
00104 Input input[1];
00106 Calibrate calibrate[1];
00107
00108 const xmlChar *result_name = (xmlChar *) "result";
00110 const xmlChar *variables_name = (xmlChar *) "variables";
00112
00113 const xmlChar *template[MAX_NINPUTS] = {
00114   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00115   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00116 };
00117
00119
00120 const char *format[NPRECISIONS] = {
00121   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00122   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00123 };
00124
00125 const double precision[NPRECISIONS] = {
00126   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,
00127   1e-13, 1e-14
00128 };
00129
00130 const char *logo[] = {
00131   "32 32 3 1",
00132   "     c None",
00133   ".    c #0000FF",
00134   "+    c #FF0000",
00135   "                                ",
00136   "                                ",
00137   "                                ",
00138   "     .      .      .      .     ",
00139   "     .      .      .      .     ",
00140   "     .      .      .      .     ",
00141   "     .      .      .      .     ",
00142   "     .      .     +++     .     ",
00143   "     .      .    +++++    .     ",
00144   "     .      .    +++++    .     ",
00145   "     .      .    +++++    .     ",
00146   "    +++     .     +++    +++    ",
00147   "   +++++    .      .    +++++   ",
00148   "   +++++    .      .    +++++   ",
00149   "   +++++    .      .    +++++   ",
00150   "    +++     .      .     +++    ",
00151   "     .      .      .      .     ",
00152   "     .     +++     .      .     ",
00153   "     .    +++++    .      .     ",
00154   "     .    +++++    .      .     ",
00155   "     .    +++++    .      .     ",
00156   "     .     +++     .      .     ",
00157   "     .      .      .      .     ",
00158   "     .      .      .      .     ",
00159   "     .      .      .      .     ",
00160   "     .      .      .      .     ",
00161   "     .      .      .      .     ",
00162   "     .      .      .      .     ",
00163   "     .      .      .      .     ",
00164   "                                ",
00165   "                                ",
00166   "                                "
00167 };
00168
00169 /*
00170 const char * logo[] = {
00171 "32 32 3 1",
```

```
00172 "     c #FFFFFFFFFFFF",
00173 ".    c #00000000FFFF",
00174 "X    c #FFFF00000000",
00175 "                              ",
00176 "                              ",
00177 "                              ",
00178 "     .      .      .      .     ",
00179 "     .      .      .      .     ",
00180 "     .      .      .      .     ",
00181 "     .      .      .      .     ",
00182 "     .      .     XXX     .     ",
00183 "     .      .    XXXXX    .     ",
00184 "     .      .    XXXXX    .     ",
00185 "     .      .    XXXXX    .     ",
00186 "    XXX     .     XXX    XXX    ",
00187 "   XXXXX    .      .    XXXXX   ",
00188 "   XXXXX    .      .    XXXXX   ",
00189 "   XXXXX    .      .    XXXXX   ",
00190 "    XXX     .      .     XXX    ",
00191 "     .      .      .      .     ",
00192 "     .     XXX     .      .     ",
00193 "     .    XXXXX    .      .     ",
00194 "     .    XXXXX    .      .     ",
00195 "     .    XXXXX    .      .     ",
00196 "     .     XXX     .      .     ",
00197 "     .      .      .      .     ",
00198 "     .      .      .      .     ",
00199 "     .      .      .      .     ",
00200 "     .      .      .      .     ",
00201 "     .      .      .      .     ",
00202 "     .      .      .      .     ",
00203 "     .      .      .      .     ",
00204 "                              ",
00205 "                              ",
00206 "                              "};
00207 */
00208
00209 #if HAVE_GTK
00210 Options options[1];
00212 Running running[1];
00214 Window window[1];
00216 #endif
00217
00228 void
00229 show_message (char *title, char *msg, int type)
00230 {
00231 #if HAVE_GTK
00232   GtkMessageDialog *dlg;
00233
00234   // Creating the dialog
00235   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00236     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00237
00238   // Setting the dialog title
00239   gtk_window_set_title (GTK_WINDOW (dlg), title);
00240
00241   // Showing the dialog and waiting response
00242   gtk_dialog_run (GTK_DIALOG (dlg));
00243
00244   // Closing and freeing memory
00245   gtk_widget_destroy (GTK_WIDGET (dlg));
00246
00247 #else
00248   printf ("%s: %s\n", title, msg);
00249 #endif
00250 }
00251
00258 void
00259 show_error (char *msg)
00260 {
00261   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00262 }
00263
00276 int
00277 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00278 {
00279   int i = 0;
00280   xmlChar *buffer;
00281   buffer = xmlGetProp (node, prop);
00282   if (!buffer)
00283     *error_code = 1;
00284   else
00285     {
00286       if (sscanf ((char *) buffer, "%d", &i) != 1)
00287         *error_code = 2;
00288       else
00289         *error_code = 0;
```

```
00290       xmlFree (buffer);
00291     }
00292   return i;
00293 }
00294
00307 unsigned int
00308 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00309 {
00310   unsigned int i = 0;
00311   xmlChar *buffer;
00312   buffer = xmlGetProp (node, prop);
00313   if (!buffer)
00314     *error_code = 1;
00315   else
00316     {
00317       if (sscanf ((char *) buffer, "%u", &i) != 1)
00318         *error_code = 2;
00319       else
00320         *error_code = 0;
00321       xmlFree (buffer);
00322     }
00323   return i;
00324 }
00325
00341 unsigned int
00342 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00343                                 unsigned int default_value, int *error_code)
00344 {
00345   unsigned int i;
00346   if (xmlHasProp (node, prop))
00347     i = xml_node_get_uint (node, prop, error_code);
00348   else
00349     {
00350       i = default_value;
00351       *error_code = 0;
00352     }
00353   return i;
00354 }
00355
00368 double
00369 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00370 {
00371   double x = 0.;
00372   xmlChar *buffer;
00373   buffer = xmlGetProp (node, prop);
00374   if (!buffer)
00375     *error_code = 1;
00376   else
00377     {
00378       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00379         *error_code = 2;
00380       else
00381         *error_code = 0;
00382       xmlFree (buffer);
00383     }
00384   return x;
00385 }
00386
00402 double
00403 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00404                                  double default_value, int *error_code)
00405 {
00406   double x;
00407   if (xmlHasProp (node, prop))
00408     x = xml_node_get_float (node, prop, error_code);
00409   else
00410     {
00411       x = default_value;
00412       *error_code = 0;
00413     }
00414   return x;
00415 }
00416
00427 void
00428 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00429 {
00430   xmlChar buffer[64];
00431   snprintf ((char *) buffer, 64, "%d", value);
00432   xmlSetProp (node, prop, buffer);
00433 }
00434
00446 void
00447 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00448 {
00449   xmlChar buffer[64];
00450   snprintf ((char *) buffer, 64, "%u", value);
00451   xmlSetProp (node, prop, buffer);
```

```
00452 }
00453
00465 void
00466 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00467 {
00468   xmlChar buffer[64];
00469   snprintf ((char *) buffer, 64, "%.14lg", value);
00470   xmlSetProp (node, prop, buffer);
00471 }
00472
00473 #if HAVE_GTK
00474
00485 unsigned int
00486 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00487 {
00488   unsigned int i;
00489   for (i = 0; i < n; ++i)
00490     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00491       break;
00492   return i;
00493 }
00494
00495 #endif
00496
00500 void
00501 input_new ()
00502 {
00503   unsigned int i;
00504 #if DEBUG
00505   fprintf (stderr, "input_new: start\n");
00506 #endif
00507   input->nvariables = input->nexperiments = input->ninputs = input->
      nsteps = 0;
00508   input->simulator = input->evaluator = input->directory = input->
      name
00509     = input->result = input->variables = NULL;
00510   input->experiment = input->label = NULL;
00511   input->precision = input->nsweeps = input->nbits = NULL;
00512   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00513     = input->weight = input->step = NULL;
00514   for (i = 0; i < MAX_NINPUTS; ++i)
00515     input->template[i] = NULL;
00516 #if DEBUG
00517   fprintf (stderr, "input_new: end\n");
00518 #endif
00519 }
00520
00525 void
00526 input_free ()
00527 {
00528   unsigned int i, j;
00529 #if DEBUG
00530   fprintf (stderr, "input_free: start\n");
00531 #endif
00532   g_free (input->name);
00533   g_free (input->directory);
00534   for (i = 0; i < input->nexperiments; ++i)
00535     {
00536       xmlFree (input->experiment[i]);
00537       for (j = 0; j < input->ninputs; ++j)
00538         xmlFree (input->template[j][i]);
00539       g_free (input->template[j]);
00540     }
00541   g_free (input->experiment);
00542   for (i = 0; i < input->ninputs; ++i)
00543     g_free (input->template[i]);
00544   for (i = 0; i < input->nvariables; ++i)
00545     xmlFree (input->label[i]);
00546   g_free (input->label);
00547   g_free (input->precision);
00548   g_free (input->rangemin);
00549   g_free (input->rangemax);
00550   g_free (input->rangeminabs);
00551   g_free (input->rangemaxabs);
00552   g_free (input->weight);
00553   g_free (input->step);
00554   g_free (input->nsweeps);
00555   g_free (input->nbits);
00556   xmlFree (input->evaluator);
00557   xmlFree (input->simulator);
00558   xmlFree (input->result);
00559   xmlFree (input->variables);
00560   input->nexperiments = input->ninputs = input->nvariables = input->
      nsteps = 0;
00561 #if DEBUG
00562   fprintf (stderr, "input_free: end\n");
```

```
00563 #endif
00564 }
00565
00573 int
00574 input_open (char *filename)
00575 {
00576   char buffer2[64];
00577   char *buffert[MAX_NINPUTS] =
00578     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00579   xmlDoc *doc;
00580   xmlNode *node, *child;
00581   xmlChar *buffer;
00582   char *msg;
00583   int error_code;
00584   unsigned int i;
00585
00586 #if DEBUG
00587   fprintf (stderr, "input_open: start\n");
00588 #endif
00589
00590   // Resetting input data
00591   buffer = NULL;
00592   input_new ();
00593
00594   // Parsing the input file
00595 #if DEBUG
00596   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00597 #endif
00598   doc = xmlParseFile (filename);
00599   if (!doc)
00600     {
00601       msg = gettext ("Unable to parse the input file");
00602       goto exit_on_error;
00603     }
00604
00605   // Getting the root node
00606 #if DEBUG
00607   fprintf (stderr, "input_open: getting the root node\n");
00608 #endif
00609   node = xmlDocGetRootElement (doc);
00610   if (xmlStrcmp (node->name, XML_CALIBRATE))
00611     {
00612       msg = gettext ("Bad root XML node");
00613       goto exit_on_error;
00614     }
00615
00616   // Getting results file names
00617   input->result = (char *) xmlGetProp (node, XML_RESULT);
00618   if (!input->result)
00619     input->result = (char *) xmlStrdup (result_name);
00620   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00621   if (!input->variables)
00622     input->variables = (char *) xmlStrdup (variables_name);
00623
00624   // Opening simulator program name
00625   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00626   if (!input->simulator)
00627     {
00628       msg = gettext ("Bad simulator program");
00629       goto exit_on_error;
00630     }
00631
00632   // Opening evaluator program name
00633   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00634
00635   // Obtaining pseudo-random numbers generator seed
00636   input->seed
00637     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00638                                       &error_code);
00639   if (error_code)
00640     {
00641       msg = gettext ("Bad pseudo-random numbers generator seed");
00642       goto exit_on_error;
00643     }
00644
00645   // Opening algorithm
00646   buffer = xmlGetProp (node, XML_ALGORITHM);
00647   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00648     {
00649       input->algorithm = ALGORITHM_MONTE_CARLO;
00650
00651       // Obtaining simulations number
00652       input->nsimulations
00653         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00654       if (error_code)
00655         {
```

```
00656           msg = gettext ("Bad simulations number");
00657           goto exit_on_error;
00658         }
00659     }
00660   else if (!xmlStrcmp (buffer, XML_SWEEP))
00661     input->algorithm = ALGORITHM_SWEEP;
00662   else if (!xmlStrcmp (buffer, XML_GENETIC))
00663     {
00664       input->algorithm = ALGORITHM_GENETIC;
00665
00666       // Obtaining population
00667       if (xmlHasProp (node, XML_NPOPULATION))
00668         {
00669           input->nsimulations
00670             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00671           if (error_code || input->nsimulations < 3)
00672             {
00673               msg = gettext ("Invalid population number");
00674               goto exit_on_error;
00675             }
00676         }
00677       else
00678         {
00679           msg = gettext ("No population number");
00680           goto exit_on_error;
00681         }
00682
00683       // Obtaining generations
00684       if (xmlHasProp (node, XML_NGENERATIONS))
00685         {
00686           input->niterations
00687             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00688           if (error_code || !input->niterations)
00689             {
00690               msg = gettext ("Invalid generations number");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         {
00696           msg = gettext ("No generations number");
00697           goto exit_on_error;
00698         }
00699
00700       // Obtaining mutation probability
00701       if (xmlHasProp (node, XML_MUTATION))
00702         {
00703           input->mutation_ratio
00704             = xml_node_get_float (node, XML_MUTATION, &error_code);
00705           if (error_code || input->mutation_ratio < 0.
00706               || input->mutation_ratio >= 1.)
00707             {
00708               msg = gettext ("Invalid mutation probability");
00709               goto exit_on_error;
00710             }
00711         }
00712       else
00713         {
00714           msg = gettext ("No mutation probability");
00715           goto exit_on_error;
00716         }
00717
00718       // Obtaining reproduction probability
00719       if (xmlHasProp (node, XML_REPRODUCTION))
00720         {
00721           input->reproduction_ratio
00722             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00723           if (error_code || input->reproduction_ratio < 0.
00724               || input->reproduction_ratio >= 1.0)
00725             {
00726               msg = gettext ("Invalid reproduction probability");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         {
00732           msg = gettext ("No reproduction probability");
00733           goto exit_on_error;
00734         }
00735
00736       // Obtaining adaptation probability
00737       if (xmlHasProp (node, XML_ADAPTATION))
00738         {
00739           input->adaptation_ratio
00740             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00741           if (error_code || input->adaptation_ratio < 0.
00742               || input->adaptation_ratio >= 1.)
```

```
00743             {
00744               msg = gettext ("Invalid adaptation probability");
00745               goto exit_on_error;
00746             }
00747         }
00748       else
00749         {
00750           msg = gettext ("No adaptation probability");
00751           goto exit_on_error;
00752         }
00753
00754       // Checking survivals
00755       i = input->mutation_ratio * input->nsimulations;
00756       i += input->reproduction_ratio * input->nsimulations;
00757       i += input->adaptation_ratio * input->nsimulations;
00758       if (i > input->nsimulations - 2)
00759         {
00760           msg = gettext
00761             ("No enough survival entities to reproduce the population");
00762           goto exit_on_error;
00763         }
00764     }
00765   else
00766     {
00767       msg = gettext ("Unknown algorithm");
00768       goto exit_on_error;
00769     }
00770   xmlFree (buffer);
00771   buffer = NULL;
00772
00773   if (input->algorithm == ALGORITHM_MONTE_CARLO
00774       || input->algorithm == ALGORITHM_SWEEP)
00775     {
00776
00777       // Obtaining iterations number
00778       input->niterations
00779         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00780       if (error_code == 1)
00781         input->niterations = 1;
00782       else if (error_code)
00783         {
00784           msg = gettext ("Bad iterations number");
00785           goto exit_on_error;
00786         }
00787
00788       // Obtaining best number
00789       input->nbest
00790         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00791       if (error_code || !input->nbest)
00792         {
00793           msg = gettext ("Invalid best number");
00794           goto exit_on_error;
00795         }
00796
00797       // Obtaining tolerance
00798       input->tolerance
00799         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00800                                          &error_code);
00801       if (error_code || input->tolerance < 0.)
00802         {
00803           msg = gettext ("Invalid tolerance");
00804           goto exit_on_error;
00805         }
00806
00807       // Getting gradient method parameters
00808       if (xmlHasProp (node, XML_NSTEPS))
00809         {
00810           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00811           if (error_code || !input->nsteps)
00812             {
00813               msg = gettext ("Invalid steps number");
00814               goto exit_on_error;
00815             }
00816           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00817           if (!xmlStrcmp (buffer, XML_COORDINATES))
00818             input->gradient_method = GRADIENT_METHOD_COORDINATES;
00819           else if (!xmlStrcmp (buffer, XML_RANDOM))
00820             {
00821               input->gradient_method = GRADIENT_METHOD_RANDOM;
00822               input->nestimates
00823                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00824               if (error_code || !input->nestimates)
00825                 {
00826                   msg = gettext ("Invalid estimates number");
```

```
00827                   goto exit_on_error;
00828                 }
00829             }
00830           else
00831             {
00832               msg = gettext ("Unknown method to estimate the gradient");
00833               goto exit_on_error;
00834             }
00835           xmlFree (buffer);
00836           buffer = NULL;
00837           input->relaxation
00838             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00839                                                DEFAULT_RELAXATION, &error_code);
00840           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00841             {
00842               msg = gettext ("Invalid relaxation parameter");
00843               goto exit_on_error;
00844             }
00845         }
00846       else
00847         input->nsteps = 0;
00848     }
00849
00850   // Reading the experimental data
00851   for (child = node->children; child; child = child->next)
00852     {
00853       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00854         break;
00855 #if DEBUG
00856       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00857 #endif
00858       if (xmlHasProp (child, XML_NAME))
00859         buffer = xmlGetProp (child, XML_NAME);
00860       else
00861         {
00862           snprintf (buffer2, 64, "%s %u: %s",
00863                     gettext ("Experiment"),
00864                     input->nexperiments + 1, gettext ("no data file name"));
00865           msg = buffer2;
00866           goto exit_on_error;
00867         }
00868 #if DEBUG
00869       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00870 #endif
00871       input->weight = g_realloc (input->weight,
00872                                  (1 + input->nexperiments) * sizeof (double));
00873       input->weight[input->nexperiments]
00874         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00875       if (error_code)
00876         {
00877           snprintf (buffer2, 64, "%s %s: %s",
00878                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00879           msg = buffer2;
00880           goto exit_on_error;
00881         }
00882 #if DEBUG
00883       fprintf (stderr, "input_open: weight=%lg\n",
00884                input->weight[input->nexperiments]);
00885 #endif
00886       if (!input->nexperiments)
00887         input->ninputs = 0;
00888 #if DEBUG
00889       fprintf (stderr, "input_open: template[0]\n");
00890 #endif
00891       if (xmlHasProp (child, XML_TEMPLATE1))
00892         {
00893           input->template[0]
00894             = (char **) g_realloc (input->template[0],
00895                                    (1 + input->nexperiments) * sizeof (char *));
00896           buffert[0] = (char *) xmlGetProp (child, template[0]);
00897 #if DEBUG
00898           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00899                    input->nexperiments, buffert[0]);
00900 #endif
00901           if (!input->nexperiments)
00902             ++input->ninputs;
00903 #if DEBUG
00904           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00905 #endif
00906         }
00907       else
00908         {
00909           snprintf (buffer2, 64, "%s %s: %s",
00910                     gettext ("Experiment"), buffer, gettext ("no template"));
```

```
00911           msg = buffer2;
00912           goto exit_on_error;
00913         }
00914       for (i = 1; i < MAX_NINPUTS; ++i)
00915         {
00916 #if DEBUG
00917           fprintf (stderr, "input_open: template%u\n", i + 1);
00918 #endif
00919           if (xmlHasProp (child, template[i]))
00920             {
00921               if (input->nexperiments && input->ninputs <= i)
00922                 {
00923                   snprintf (buffer2, 64, "%s %s: %s",
00924                             gettext ("Experiment"),
00925                             buffer, gettext ("bad templates number"));
00926                   msg = buffer2;
00927                   while (i-- > 0)
00928                     xmlFree (buffert[i]);
00929                   goto exit_on_error;
00930                 }
00931               input->template[i] = (char **)
00932                 g_realloc (input->template[i],
00933                            (1 + input->nexperiments) * sizeof (char *));
00934               buffert[i] = (char *) xmlGetProp (child, template[i]);
00935 #if DEBUG
00936               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00937                        input->nexperiments, i + 1,
00938                        input->template[i][input->nexperiments]);
00939 #endif
00940               if (!input->nexperiments)
00941                 ++input->ninputs;
00942 #if DEBUG
00943               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00944 #endif
00945             }
00946           else if (input->nexperiments && input->ninputs > i)
00947             {
00948               snprintf (buffer2, 64, "%s %s: %s%u",
00949                         gettext ("Experiment"),
00950                         buffer, gettext ("no template"), i + 1);
00951               msg = buffer2;
00952               while (i-- > 0)
00953                 xmlFree (buffert[i]);
00954               goto exit_on_error;
00955             }
00956           else
00957             break;
00958         }
00959       input->experiment
00960         = g_realloc (input->experiment,
00961                      (1 + input->nexperiments) * sizeof (char *));
00962       input->experiment[input->nexperiments] = (char *) buffer;
00963       for (i = 0; i < input->ninputs; ++i)
00964         input->template[i][input->nexperiments] = buffert[i];
00965       ++input->nexperiments;
00966 #if DEBUG
00967       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00968 #endif
00969     }
00970   if (!input->nexperiments)
00971     {
00972       msg = gettext ("No calibration experiments");
00973       goto exit_on_error;
00974     }
00975   buffer = NULL;
00976
00977   // Reading the variables data
00978   for (; child; child = child->next)
00979     {
00980       if (xmlStrcmp (child->name, XML_VARIABLE))
00981         {
00982           snprintf (buffer2, 64, "%s %u: %s",
00983                     gettext ("Variable"),
00984                     input->nvariables + 1, gettext ("bad XML node"));
00985           msg = buffer2;
00986           goto exit_on_error;
00987         }
00988       if (xmlHasProp (child, XML_NAME))
00989         buffer = xmlGetProp (child, XML_NAME);
00990       else
00991         {
00992           snprintf (buffer2, 64, "%s %u: %s",
00993                     gettext ("Variable"),
00994                     input->nvariables + 1, gettext ("no name"));
00995           msg = buffer2;
00996           goto exit_on_error;
00997         }
```

```
00998       if (xmlHasProp (child, XML_MINIMUM))
00999         {
01000           input->rangemin = g_realloc
01001             (input->rangemin, (1 + input->nvariables) * sizeof (double));
01002           input->rangeminabs = g_realloc
01003             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
01004           input->rangemin[input->nvariables]
01005             = xml_node_get_float (child, XML_MINIMUM, &error_code);
01006           if (error_code)
01007             {
01008               snprintf (buffer2, 64, "%s %s: %s",
01009                         gettext ("Variable"), buffer, gettext ("bad minimum"));
01010               msg = buffer2;
01011               goto exit_on_error;
01012             }
01013           input->rangeminabs[input->nvariables]
01014             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MINIMUM,
01015                                                -G_MAXDOUBLE, &error_code);
01016           if (error_code)
01017             {
01018               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01019                         gettext ("bad absolute minimum"));
01020               msg = buffer2;
01021               goto exit_on_error;
01022             }
01023           if (input->rangemin[input->nvariables]
01024               < input->rangeminabs[input->nvariables])
01025             {
01026               snprintf (buffer2, 64, "%s %s: %s",
01027                         gettext ("Variable"),
01028                         buffer, gettext ("minimum range not allowed"));
01029               msg = buffer2;
01030               goto exit_on_error;
01031             }
01032         }
01033       else
01034         {
01035           snprintf (buffer2, 64, "%s %s: %s",
01036                     gettext ("Variable"), buffer, gettext ("no minimum range"));
01037           msg = buffer2;
01038           goto exit_on_error;
01039         }
01040       if (xmlHasProp (child, XML_MAXIMUM))
01041         {
01042           input->rangemax = g_realloc
01043             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01044           input->rangemaxabs = g_realloc
01045             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01046           input->rangemax[input->nvariables]
01047             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01048           if (error_code)
01049             {
01050               snprintf (buffer2, 64, "%s %s: %s",
01051                         gettext ("Variable"), buffer, gettext ("bad maximum"));
01052               msg = buffer2;
01053               goto exit_on_error;
01054             }
01055           input->rangemaxabs[input->nvariables]
01056             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MAXIMUM,
01057                                                G_MAXDOUBLE, &error_code);
01058           if (error_code)
01059             {
01060               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01061                         gettext ("bad absolute maximum"));
01062               msg = buffer2;
01063               goto exit_on_error;
01064             }
01065           if (input->rangemax[input->nvariables]
01066               > input->rangemaxabs[input->nvariables])
01067             {
01068               snprintf (buffer2, 64, "%s %s: %s",
01069                         gettext ("Variable"),
01070                         buffer, gettext ("maximum range not allowed"));
01071               msg = buffer2;
01072               goto exit_on_error;
01073             }
01074         }
01075       else
01076         {
01077           snprintf (buffer2, 64, "%s %s: %s",
01078                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01079           msg = buffer2;
01080           goto exit_on_error;
01081         }
01082       if (input->rangemax[input->nvariables]
```

```
01083           < input->rangemin[input->nvariables])
01084         {
01085           snprintf (buffer2, 64, "%s %s: %s",
01086                     gettext ("Variable"), buffer, gettext ("bad range"));
01087           msg = buffer2;
01088           goto exit_on_error;
01089         }
01090       input->precision = g_realloc
01091         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01092       input->precision[input->nvariables]
01093         = xml_node_get_uint_with_default (child,
      XML_PRECISION,
01094                                           DEFAULT_PRECISION, &error_code);
01095       if (error_code || input->precision[input->nvariables] >=
      NPRECISIONS)
01096         {
01097           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                     gettext ("bad precision"));
01099           msg = buffer2;
01100           goto exit_on_error;
01101         }
01102       if (input->algorithm == ALGORITHM_SWEEP)
01103         {
01104           if (xmlHasProp (child, XML_NSWEEPS))
01105             {
01106               input->nsweeps = (unsigned int *)
01107                 g_realloc (input->nsweeps,
01108                            (1 + input->nvariables) * sizeof (unsigned int));
01109               input->nsweeps[input->nvariables]
01110                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01111               if (error_code || !input->nsweeps[input->nvariables])
01112                 {
01113                   snprintf (buffer2, 64, "%s %s: %s",
01114                             gettext ("Variable"),
01115                             buffer, gettext ("bad sweeps"));
01116                   msg = buffer2;
01117                   goto exit_on_error;
01118                 }
01119             }
01120           else
01121             {
01122               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01123                         gettext ("no sweeps number"));
01124               msg = buffer2;
01125               goto exit_on_error;
01126             }
01127 #if DEBUG
01128           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01129                    input->nsweeps[input->nvariables], input->
      nsimulations);
01130 #endif
01131         }
01132       if (input->algorithm == ALGORITHM_GENETIC)
01133         {
01134           // Obtaining bits representing each variable
01135           if (xmlHasProp (child, XML_NBITS))
01136             {
01137               input->nbits = (unsigned int *)
01138                 g_realloc (input->nbits,
01139                            (1 + input->nvariables) * sizeof (unsigned int));
01140               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01141               if (error_code || !i)
01142                 {
01143                   snprintf (buffer2, 64, "%s %s: %s",
01144                             gettext ("Variable"),
01145                             buffer, gettext ("invalid bits number"));
01146                   msg = buffer2;
01147                   goto exit_on_error;
01148                 }
01149               input->nbits[input->nvariables] = i;
01150             }
01151           else
01152             {
01153               snprintf (buffer2, 64, "%s %s: %s",
01154                         gettext ("Variable"),
01155                         buffer, gettext ("no bits number"));
01156               msg = buffer2;
01157               goto exit_on_error;
01158             }
01159         }
01160       else if (input->nsteps)
01161         {
01162           input->step = (double *)
01163             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01164           input->step[input->nvariables]
01165             = xml_node_get_float (child, XML_STEP, &error_code);
01166           if (error_code || input->step[input->nvariables] < 0.)
```

```
01167             {
01168               snprintf (buffer2, 64, "%s %s: %s",
01169                         gettext ("Variable"),
01170                         buffer, gettext ("bad step size"));
01171               msg = buffer2;
01172               goto exit_on_error;
01173             }
01174         }
01175       input->label = g_realloc
01176         (input->label, (1 + input->nvariables) * sizeof (char *));
01177       input->label[input->nvariables] = (char *) buffer;
01178       ++input->nvariables;
01179     }
01180   if (!input->nvariables)
01181     {
01182       msg = gettext ("No calibration variables");
01183       goto exit_on_error;
01184     }
01185   buffer = NULL;
01186
01187   // Obtaining the error norm
01188   if (xmlHasProp (node, XML_NORM))
01189     {
01190       buffer = xmlGetProp (node, XML_NORM);
01191       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
01192         input->norm = ERROR_NORM_EUCLIDIAN;
01193       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
01194         input->norm = ERROR_NORM_MAXIMUM;
01195       else if (!xmlStrcmp (buffer, XML_P))
01196         {
01197           input->norm = ERROR_NORM_P;
01198           input->p = xml_node_get_float (node, XML_P, &error_code);
01199           if (!error_code)
01200             {
01201               msg = gettext ("Bad P parameter");
01202               goto exit_on_error;
01203             }
01204         }
01205       else if (!xmlStrcmp (buffer, XML_TAXICAB))
01206         input->norm = ERROR_NORM_TAXICAB;
01207       else
01208         {
01209           msg = gettext ("Unknown error norm");
01210           goto exit_on_error;
01211         }
01212       xmlFree (buffer);
01213     }
01214   else
01215     input->norm = ERROR_NORM_EUCLIDIAN;
01216
01217   // Getting the working directory
01218   input->directory = g_path_get_dirname (filename);
01219   input->name = g_path_get_basename (filename);
01220
01221   // Closing the XML document
01222   xmlFreeDoc (doc);
01223
01224 #if DEBUG
01225   fprintf (stderr, "input_open: end\n");
01226 #endif
01227   return 1;
01228
01229 exit_on_error:
01230   xmlFree (buffer);
01231   xmlFreeDoc (doc);
01232   show_error (msg);
01233   input_free ();
01234 #if DEBUG
01235   fprintf (stderr, "input_open: end\n");
01236 #endif
01237   return 0;
01238 }
01239
01251 void
01252 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01253 {
01254   unsigned int i;
01255   char buffer[32], value[32], *buffer2, *buffer3, *content;
01256   FILE *file;
01257   gsize length;
01258   GRegex *regex;
01259
01260 #if DEBUG
01261   fprintf (stderr, "calibrate_input: start\n");
01262 #endif
01263
01264   // Checking the file
```

```
01265   if (!template)
01266     goto calibrate_input_end;
01267
01268   // Opening template
01269   content = g_mapped_file_get_contents (template);
01270   length = g_mapped_file_get_length (template);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01273            content);
01274 #endif
01275   file = g_fopen (input, "w");
01276
01277   // Parsing template
01278   for (i = 0; i < calibrate->nvariables; ++i)
01279     {
01280 #if DEBUG
01281       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01282 #endif
01283       snprintf (buffer, 32, "@variable%u@", i + 1);
01284       regex = g_regex_new (buffer, 0, 0, NULL);
01285       if (i == 0)
01286         {
01287           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01288                                              calibrate->label[i], 0, NULL);
01289 #if DEBUG
01290           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01291 #endif
01292         }
01293       else
01294         {
01295           length = strlen (buffer3);
01296           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01297                                              calibrate->label[i], 0, NULL);
01298           g_free (buffer3);
01299         }
01300       g_regex_unref (regex);
01301       length = strlen (buffer2);
01302       snprintf (buffer, 32, "@value%u@", i + 1);
01303       regex = g_regex_new (buffer, 0, 0, NULL);
01304       snprintf (value, 32, format[calibrate->precision[i]],
01305                 calibrate->value[simulation * calibrate->nvariables + i]);
01306
01307 #if DEBUG
01308       fprintf (stderr, "calibrate_input: value=%s\n", value);
01309 #endif
01310       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01311                                          0, NULL);
01312       g_free (buffer2);
01313       g_regex_unref (regex);
01314     }
01315
01316   // Saving input file
01317   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01318   g_free (buffer3);
01319   fclose (file);
01320
01321 calibrate_input_end:
01322 #if DEBUG
01323   fprintf (stderr, "calibrate_input: end\n");
01324 #endif
01325   return;
01326 }
01327
01338 double
01339 calibrate_parse (unsigned int simulation, unsigned int experiment)
01340 {
01341   unsigned int i;
01342   double e;
01343   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01344     *buffer3, *buffer4;
01345   FILE *file_result;
01346
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_parse: start\n");
01349   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01350            experiment);
01351 #endif
01352
01353   // Opening input files
01354   for (i = 0; i < calibrate->ninputs; ++i)
01355     {
01356       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01357 #if DEBUG
01358       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01359 #endif
01360       calibrate_input (simulation, &input[i][0],
01361                        calibrate->file[i][experiment]);
```

```
01362     }
01363   for (; i < MAX_NINPUTS; ++i)
01364     strcpy (&input[i][0], "");
01365 #if DEBUG
01366   fprintf (stderr, "calibrate_parse: parsing end\n");
01367 #endif
01368
01369   // Performing the simulation
01370   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01371   buffer2 = g_path_get_dirname (calibrate->simulator);
01372   buffer3 = g_path_get_basename (calibrate->simulator);
01373   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01374   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01375             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01376             input[6], input[7], output);
01377   g_free (buffer4);
01378   g_free (buffer3);
01379   g_free (buffer2);
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01382 #endif
01383   system (buffer);
01384
01385   // Checking the objective value function
01386   if (calibrate->evaluator)
01387     {
01388       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01389       buffer2 = g_path_get_dirname (calibrate->evaluator);
01390       buffer3 = g_path_get_basename (calibrate->evaluator);
01391       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01392       snprintf (buffer, 512, "\"%s\" %s %s %s",
01393                 buffer4, output, calibrate->experiment[experiment], result);
01394       g_free (buffer4);
01395       g_free (buffer3);
01396       g_free (buffer2);
01397 #if DEBUG
01398       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01399 #endif
01400       system (buffer);
01401       file_result = g_fopen (result, "r");
01402       e = atof (fgets (buffer, 512, file_result));
01403       fclose (file_result);
01404     }
01405   else
01406     {
01407       strcpy (result, "");
01408       file_result = g_fopen (output, "r");
01409       e = atof (fgets (buffer, 512, file_result));
01410       fclose (file_result);
01411     }
01412
01413   // Removing files
01414 #if !DEBUG
01415   for (i = 0; i < calibrate->ninputs; ++i)
01416     {
01417       if (calibrate->file[i][0])
01418         {
01419           snprintf (buffer, 512, RM " %s", &input[i][0]);
01420           system (buffer);
01421         }
01422     }
01423   snprintf (buffer, 512, RM " %s %s", output, result);
01424   system (buffer);
01425 #endif
01426
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_parse: end\n");
01429 #endif
01430
01431   // Returning the objective function
01432   return e * calibrate->weight[experiment];
01433 }
01434
01442 double
01443 calibrate_norm_euclidian (unsigned int simulation)
01444 {
01445   double e, ei;
01446   unsigned int i;
01447 #if DEBUG
01448   fprintf (stderr, "calibrate_norm_euclidian: start\n");
01449 #endif
01450   e = 0.;
01451   for (i = 0; i < calibrate->nexperiments; ++i)
01452     {
01453       ei = calibrate_parse (simulation, i);
01454       e += ei * ei;
01455     }
```

```
01456   e = sqrt (e);
01457 #if DEBUG
01458   fprintf (stderr, "calibrate_norm_euclidian: error=%lg\n", e);
01459   fprintf (stderr, "calibrate_norm_euclidian: end\n");
01460 #endif
01461   return e;
01462 }
01463
01471 double
01472 calibrate_norm_maximum (unsigned int simulation)
01473 {
01474   double e, ei;
01475   unsigned int i;
01476 #if DEBUG
01477   fprintf (stderr, "calibrate_norm_maximum: start\n");
01478 #endif
01479   e = 0.;
01480   for (i = 0; i < calibrate->nexperiments; ++i)
01481     {
01482       ei = fabs (calibrate_parse (simulation, i));
01483       e = fmax (e, ei);
01484     }
01485 #if DEBUG
01486   fprintf (stderr, "calibrate_norm_maximum: error=%lg\n", e);
01487   fprintf (stderr, "calibrate_norm_maximum: end\n");
01488 #endif
01489   return e;
01490 }
01491
01499 double
01500 calibrate_norm_p (unsigned int simulation)
01501 {
01502   double e, ei;
01503   unsigned int i;
01504 #if DEBUG
01505   fprintf (stderr, "calibrate_norm_p: start\n");
01506 #endif
01507   e = 0.;
01508   for (i = 0; i < calibrate->nexperiments; ++i)
01509     {
01510       ei = fabs (calibrate_parse (simulation, i));
01511       e += pow (ei, calibrate->p);
01512     }
01513   e = pow (e, 1. / calibrate->p);
01514 #if DEBUG
01515   fprintf (stderr, "calibrate_norm_p: error=%lg\n", e);
01516   fprintf (stderr, "calibrate_norm_p: end\n");
01517 #endif
01518   return e;
01519 }
01520
01528 double
01529 calibrate_norm_taxicab (unsigned int simulation)
01530 {
01531   double e;
01532   unsigned int i;
01533 #if DEBUG
01534   fprintf (stderr, "calibrate_norm_taxicab: start\n");
01535 #endif
01536   e = 0.;
01537   for (i = 0; i < calibrate->nexperiments; ++i)
01538     e += fabs (calibrate_parse (simulation, i));
01539 #if DEBUG
01540   fprintf (stderr, "calibrate_norm_taxicab: error=%lg\n", e);
01541   fprintf (stderr, "calibrate_norm_taxicab: end\n");
01542 #endif
01543   return e;
01544 }
01545
01550 void
01551 calibrate_print ()
01552 {
01553   unsigned int i;
01554   char buffer[512];
01555 #if HAVE_MPI
01556   if (calibrate->mpi_rank)
01557     return;
01558 #endif
01559   printf ("%s\n", gettext ("Best result"));
01560   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01561   printf ("error = %.15le\n", calibrate->error_old[0]);
01562   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
      error_old[0]);
01563   for (i = 0; i < calibrate->nvariables; ++i)
01564     {
01565       snprintf (buffer, 512, "%s = %s\n",
01566                 calibrate->label[i], format[calibrate->precision[i]]);
```

```
01567       printf (buffer, calibrate->value_old[i]);
01568       fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01569     }
01570   fflush (calibrate->file_result);
01571 }
01572
01581 void
01582 calibrate_save_variables (unsigned int simulation, double error)
01583 {
01584   unsigned int i;
01585   char buffer[64];
01586 #if DEBUG
01587   fprintf (stderr, "calibrate_save_variables: start\n");
01588 #endif
01589   for (i = 0; i < calibrate->nvariables; ++i)
01590     {
01591       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01592       fprintf (calibrate->file_variables, buffer,
01593                calibrate->value[simulation * calibrate->nvariables + i]);
01594     }
01595   fprintf (calibrate->file_variables, "%.14le\n", error);
01596 #if DEBUG
01597   fprintf (stderr, "calibrate_save_variables: end\n");
01598 #endif
01599 }
01600
01609 void
01610 calibrate_best (unsigned int simulation, double value)
01611 {
01612   unsigned int i, j;
01613   double e;
01614 #if DEBUG
01615   fprintf (stderr, "calibrate_best: start\n");
01616   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01617            calibrate->nsaveds, calibrate->nbest);
01618 #endif
01619   if (calibrate->nsaveds < calibrate->nbest
01620       || value < calibrate->error_best[calibrate->nsaveds - 1])
01621     {
01622       if (calibrate->nsaveds < calibrate->nbest)
01623         ++calibrate->nsaveds;
01624       calibrate->error_best[calibrate->nsaveds - 1] = value;
01625       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01626       for (i = calibrate->nsaveds; --i;)
01627         {
01628           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01629             {
01630               j = calibrate->simulation_best[i];
01631               e = calibrate->error_best[i];
01632               calibrate->simulation_best[i] = calibrate->
    simulation_best[i - 1];
01633               calibrate->error_best[i] = calibrate->error_best[i - 1];
01634               calibrate->simulation_best[i - 1] = j;
01635               calibrate->error_best[i - 1] = e;
01636             }
01637           else
01638             break;
01639         }
01640     }
01641 #if DEBUG
01642   fprintf (stderr, "calibrate_best: end\n");
01643 #endif
01644 }
01645
01650 void
01651 calibrate_sequential ()
01652 {
01653   unsigned int i;
01654   double e;
01655 #if DEBUG
01656   fprintf (stderr, "calibrate_sequential: start\n");
01657   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01658            calibrate->nstart, calibrate->nend);
01659 #endif
01660   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01661     {
01662       e = calibrate_norm (i);
01663       calibrate_best (i, e);
01664       calibrate_save_variables (i, e);
01665 #if DEBUG
01666       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01667 #endif
01668     }
01669 #if DEBUG
01670   fprintf (stderr, "calibrate_sequential: end\n");
01671 #endif
01672 }
```

```
01673
01681 void *
01682 calibrate_thread (ParallelData * data)
01683 {
01684   unsigned int i, thread;
01685   double e;
01686 #if DEBUG
01687   fprintf (stderr, "calibrate_thread: start\n");
01688 #endif
01689   thread = data->thread;
01690 #if DEBUG
01691   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01692            calibrate->thread[thread], calibrate->thread[thread + 1]);
01693 #endif
01694   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01695     {
01696       e = calibrate_norm (i);
01697       g_mutex_lock (mutex);
01698       calibrate_best (i, e);
01699       calibrate_save_variables (i, e);
01700       g_mutex_unlock (mutex);
01701 #if DEBUG
01702       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01703 #endif
01704     }
01705 #if DEBUG
01706   fprintf (stderr, "calibrate_thread: end\n");
01707 #endif
01708   g_thread_exit (NULL);
01709   return NULL;
01710 }
01711
01723 void
01724 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01725                  double *error_best)
01726 {
01727   unsigned int i, j, k, s[calibrate->nbest];
01728   double e[calibrate->nbest];
01729 #if DEBUG
01730   fprintf (stderr, "calibrate_merge: start\n");
01731 #endif
01732   i = j = k = 0;
01733   do
01734     {
01735       if (i == calibrate->nsaveds)
01736         {
01737           s[k] = simulation_best[j];
01738           e[k] = error_best[j];
01739           ++j;
01740           ++k;
01741           if (j == nsaveds)
01742             break;
01743         }
01744       else if (j == nsaveds)
01745         {
01746           s[k] = calibrate->simulation_best[i];
01747           e[k] = calibrate->error_best[i];
01748           ++i;
01749           ++k;
01750           if (i == calibrate->nsaveds)
01751             break;
01752         }
01753       else if (calibrate->error_best[i] > error_best[j])
01754         {
01755           s[k] = simulation_best[j];
01756           e[k] = error_best[j];
01757           ++j;
01758           ++k;
01759         }
01760       else
01761         {
01762           s[k] = calibrate->simulation_best[i];
01763           e[k] = calibrate->error_best[i];
01764           ++i;
01765           ++k;
01766         }
01767     }
01768   while (k < calibrate->nbest);
01769   calibrate->nsaveds = k;
01770   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01771   memcpy (calibrate->error_best, e, k * sizeof (double));
01772 #if DEBUG
01773   fprintf (stderr, "calibrate_merge: end\n");
01774 #endif
01775 }
01776
01781 #if HAVE_MPI
```

```
01782 void
01783 calibrate_synchronise ()
01784 {
01785   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01786   double error_best[calibrate->nbest];
01787   MPI_Status mpi_stat;
01788 #if DEBUG
01789   fprintf (stderr, "calibrate_synchronise: start\n");
01790 #endif
01791   if (calibrate->mpi_rank == 0)
01792     {
01793       for (i = 1; i < ntasks; ++i)
01794         {
01795           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01796           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01797                     MPI_COMM_WORLD, &mpi_stat);
01798           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01799                     MPI_COMM_WORLD, &mpi_stat);
01800           calibrate_merge (nsaveds, simulation_best, error_best);
01801         }
01802     }
01803   else
01804     {
01805       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01806       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01807                 MPI_COMM_WORLD);
01808       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01809                 MPI_COMM_WORLD);
01810     }
01811 #if DEBUG
01812   fprintf (stderr, "calibrate_synchronise: end\n");
01813 #endif
01814 }
01815 #endif
01816
01821 void
01822 calibrate_sweep ()
01823 {
01824   unsigned int i, j, k, l;
01825   double e;
01826   GThread *thread[nthreads];
01827   ParallelData data[nthreads];
01828 #if DEBUG
01829   fprintf (stderr, "calibrate_sweep: start\n");
01830 #endif
01831   for (i = 0; i < calibrate->nsimulations; ++i)
01832     {
01833       k = i;
01834       for (j = 0; j < calibrate->nvariables; ++j)
01835         {
01836           l = k % calibrate->nsweeps[j];
01837           k /= calibrate->nsweeps[j];
01838           e = calibrate->rangemin[j];
01839           if (calibrate->nsweeps[j] > 1)
01840             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01841               / (calibrate->nsweeps[j] - 1);
01842           calibrate->value[i * calibrate->nvariables + j] = e;
01843         }
01844     }
01845   calibrate->nsaveds = 0;
01846   if (nthreads <= 1)
01847     calibrate_sequential ();
01848   else
01849     {
01850       for (i = 0; i < nthreads; ++i)
01851         {
01852           data[i].thread = i;
01853           thread[i]
01854             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01855         }
01856       for (i = 0; i < nthreads; ++i)
01857         g_thread_join (thread[i]);
01858     }
01859 #if HAVE_MPI
01860   // Communicating tasks results
01861   calibrate_synchronise ();
01862 #endif
01863 #if DEBUG
01864   fprintf (stderr, "calibrate_sweep: end\n");
01865 #endif
01866 }
01867
01872 void
01873 calibrate_MonteCarlo ()
01874 {
01875   unsigned int i, j;
01876   GThread *thread[nthreads];
```

```
01877   ParallelData data[nthreads];
01878 #if DEBUG
01879   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01880 #endif
01881   for (i = 0; i < calibrate->nsimulations; ++i)
01882     for (j = 0; j < calibrate->nvariables; ++j)
01883       calibrate->value[i * calibrate->nvariables + j]
01884         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01885         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01886   calibrate->nsaveds = 0;
01887   if (nthreads <= 1)
01888     calibrate_sequential ();
01889   else
01890     {
01891       for (i = 0; i < nthreads; ++i)
01892         {
01893           data[i].thread = i;
01894           thread[i]
01895             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01896         }
01897       for (i = 0; i < nthreads; ++i)
01898         g_thread_join (thread[i]);
01899     }
01900 #if HAVE_MPI
01901   // Communicating tasks results
01902   calibrate_synchronise ();
01903 #endif
01904 #if DEBUG
01905   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01906 #endif
01907 }
01908
01918 void
01919 calibrate_best_gradient (unsigned int simulation, double value)
01920 {
01921 #if DEBUG
01922   fprintf (stderr, "calibrate_best_gradient: start\n");
01923   fprintf (stderr,
01924            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01925            simulation, value, calibrate->error_best[0]);
01926 #endif
01927   if (value < calibrate->error_best[0])
01928     {
01929       calibrate->error_best[0] = value;
01930       calibrate->simulation_best[0] = simulation;
01931 #if DEBUG
01932       fprintf (stderr,
01933                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01934                simulation, value);
01935 #endif
01936     }
01937 #if DEBUG
01938   fprintf (stderr, "calibrate_best_gradient: end\n");
01939 #endif
01940 }
01941
01948 void
01949 calibrate_gradient_sequential (unsigned int simulation)
01950 {
01951   unsigned int i, j;
01952   double e;
01953 #if DEBUG
01954   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01955   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01956            "nend_gradient=%u\n",
01957            calibrate->nstart_gradient, calibrate->nend_gradient);
01958 #endif
01959   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01960     {
01961       j = simulation + i;
01962       e = calibrate_norm (j);
01963       calibrate_best_gradient (j, e);
01964       calibrate_save_variables (j, e);
01965 #if DEBUG
01966       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01967 #endif
01968     }
01969 #if DEBUG
01970   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01971 #endif
01972 }
01973
01981 void *
01982 calibrate_gradient_thread (ParallelData * data)
01983 {
01984   unsigned int i, thread;
01985   double e;
```

```
01986 #if DEBUG
01987   fprintf (stderr, "calibrate_gradient_thread: start\n");
01988 #endif
01989   thread = data->thread;
01990 #if DEBUG
01991   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01992            thread,
01993            calibrate->thread_gradient[thread],
01994            calibrate->thread_gradient[thread + 1]);
01995 #endif
01996   for (i = calibrate->thread_gradient[thread];
01997        i < calibrate->thread_gradient[thread + 1]; ++i)
01998     {
01999       e = calibrate_norm (i);
02000       g_mutex_lock (mutex);
02001       calibrate_best_gradient (i, e);
02002       calibrate_save_variables (i, e);
02003       g_mutex_unlock (mutex);
02004 #if DEBUG
02005       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
02006 #endif
02007     }
02008 #if DEBUG
02009   fprintf (stderr, "calibrate_gradient_thread: end\n");
02010 #endif
02011   g_thread_exit (NULL);
02012   return NULL;
02013 }
02014
02024 double
02025 calibrate_estimate_gradient_random (unsigned int variable,
02026                                     unsigned int estimate)
02027 {
02028   double x;
02029 #if DEBUG
02030   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
02031 #endif
02032   x = calibrate->gradient[variable]
02033     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
02034 #if DEBUG
02035   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
02036            variable, x);
02037   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
02038 #endif
02039   return x;
02040 }
02041
02051 double
02052 calibrate_estimate_gradient_coordinates (unsigned int variable,
02053                                          unsigned int estimate)
02054 {
02055   double x;
02056 #if DEBUG
02057   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
02058 #endif
02059   x = calibrate->gradient[variable];
02060   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
02061     {
02062       if (estimate & 1)
02063         x += calibrate->step[variable];
02064       else
02065         x -= calibrate->step[variable];
02066     }
02067 #if DEBUG
02068   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
02069            variable, x);
02070   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
02071 #endif
02072   return x;
02073 }
02074
02081 void
02082 calibrate_step_gradient (unsigned int simulation)
02083 {
02084   GThread *thread[nthreads_gradient];
02085   ParallelData data[nthreads_gradient];
02086   unsigned int i, j, k, b;
02087 #if DEBUG
02088   fprintf (stderr, "calibrate_step_gradient: start\n");
02089 #endif
02090   for (i = 0; i < calibrate->nestimates; ++i)
02091     {
02092       k = (simulation + i) * calibrate->nvariables;
02093       b = calibrate->simulation_best[0] * calibrate->nvariables;
02094 #if DEBUG
02095       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
02096                simulation + i, calibrate->simulation_best[0]);
```

```
02097 #endif
02098       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
02099         {
02100 #if DEBUG
02101           fprintf (stderr,
02102                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
02103                    i, j, calibrate->value[b]);
02104 #endif
02105           calibrate->value[k]
02106             = calibrate->value[b] + calibrate_estimate_gradient (j, i);
02107           calibrate->value[k] = fmin (fmax (calibrate->value[k],
02108                                             calibrate->rangeminabs[j]),
02109                                       calibrate->rangemaxabs[j]);
02110 #if DEBUG
02111           fprintf (stderr,
02112                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
02113                    i, j, calibrate->value[k]);
02114 #endif
02115         }
02116     }
02117   if (nthreads_gradient == 1)
02118     calibrate_gradient_sequential (simulation);
02119   else
02120     {
02121       for (i = 0; i <= nthreads_gradient; ++i)
02122         {
02123           calibrate->thread_gradient[i]
02124             = simulation + calibrate->nstart_gradient
02125             + i * (calibrate->nend_gradient - calibrate->
    nstart_gradient)
02126             / nthreads_gradient;
02127 #if DEBUG
02128           fprintf (stderr,
02129                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
02130                    i, calibrate->thread_gradient[i]);
02131 #endif
02132         }
02133       for (i = 0; i < nthreads_gradient; ++i)
02134         {
02135           data[i].thread = i;
02136           thread[i] = g_thread_new
02137             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
02138         }
02139       for (i = 0; i < nthreads_gradient; ++i)
02140         g_thread_join (thread[i]);
02141     }
02142 #if DEBUG
02143   fprintf (stderr, "calibrate_step_gradient: end\n");
02144 #endif
02145 }
02146
02151 void
02152 calibrate_gradient ()
02153 {
02154   unsigned int i, j, k, b, s, adjust;
02155 #if DEBUG
02156   fprintf (stderr, "calibrate_gradient: start\n");
02157 #endif
02158   for (i = 0; i < calibrate->nvariables; ++i)
02159     calibrate->gradient[i] = 0.;
02160   b = calibrate->simulation_best[0] * calibrate->nvariables;
02161   s = calibrate->nsimulations;
02162   adjust = 1;
02163   for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
02164     {
02165 #if DEBUG
02166       fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
02167                i, calibrate->simulation_best[0]);
02168 #endif
02169       calibrate_step_gradient (s);
02170       k = calibrate->simulation_best[0] * calibrate->nvariables;
02171 #if DEBUG
02172       fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
02173                i, calibrate->simulation_best[0]);
02174 #endif
02175       if (k == b)
02176         {
02177           if (adjust)
02178             for (j = 0; j < calibrate->nvariables; ++j)
02179               calibrate->step[j] *= 0.5;
02180           for (j = 0; j < calibrate->nvariables; ++j)
02181             calibrate->gradient[j] = 0.;
02182           adjust = 1;
02183         }
02184       else
02185         {
02186           for (j = 0; j < calibrate->nvariables; ++j)
```

```
02187             {
02188 #if DEBUG
02189               fprintf (stderr,
02190                        "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02191                        j, calibrate->value[k + j], j, calibrate->value[b + j]);
02192 #endif
02193               calibrate->gradient[j]
02194                 = (1. - calibrate->relaxation) * calibrate->gradient[j]
02195                 + calibrate->relaxation
02196                 * (calibrate->value[k + j] - calibrate->value[b + j]);
02197 #if DEBUG
02198               fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02199                        j, calibrate->gradient[j]);
02200 #endif
02201             }
02202           adjust = 0;
02203         }
02204     }
02205 #if DEBUG
02206   fprintf (stderr, "calibrate_gradient: end\n");
02207 #endif
02208 }
02209
02217 double
02218 calibrate_genetic_objective (Entity * entity)
02219 {
02220   unsigned int j;
02221   double objective;
02222   char buffer[64];
02223 #if DEBUG
02224   fprintf (stderr, "calibrate_genetic_objective: start\n");
02225 #endif
02226   for (j = 0; j < calibrate->nvariables; ++j)
02227     {
02228       calibrate->value[entity->id * calibrate->nvariables + j]
02229         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02230     }
02231   objective = calibrate_norm (entity->id);
02232   g_mutex_lock (mutex);
02233   for (j = 0; j < calibrate->nvariables; ++j)
02234     {
02235       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02236       fprintf (calibrate->file_variables, buffer,
02237                genetic_get_variable (entity, calibrate->genetic_variable + j));
02238     }
02239   fprintf (calibrate->file_variables, "%.14le\n", objective);
02240   g_mutex_unlock (mutex);
02241 #if DEBUG
02242   fprintf (stderr, "calibrate_genetic_objective: end\n");
02243 #endif
02244   return objective;
02245 }
02246
02251 void
02252 calibrate_genetic ()
02253 {
02254   char *best_genome;
02255   double best_objective, *best_variable;
02256 #if DEBUG
02257   fprintf (stderr, "calibrate_genetic: start\n");
02258   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02259            nthreads);
02260   fprintf (stderr,
02261            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02262            calibrate->nvariables, calibrate->nsimulations,
02263            calibrate->niterations);
02264   fprintf (stderr,
02265            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02266            calibrate->mutation_ratio, calibrate->
    reproduction_ratio,
02267            calibrate->adaptation_ratio);
02268 #endif
02269   genetic_algorithm_default (calibrate->nvariables,
02270                              calibrate->genetic_variable,
02271                              calibrate->nsimulations,
02272                              calibrate->niterations,
02273                              calibrate->mutation_ratio,
02274                              calibrate->reproduction_ratio,
02275                              calibrate->adaptation_ratio,
02276                              &calibrate_genetic_objective,
02277                              &best_genome, &best_variable, &best_objective);
02278 #if DEBUG
02279   fprintf (stderr, "calibrate_genetic: the best\n");
02280 #endif
02281   calibrate->error_old = (double *) g_malloc (sizeof (double));
02282   calibrate->value_old
02283     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
```

```
02284   calibrate->error_old[0] = best_objective;
02285   memcpy (calibrate->value_old, best_variable,
02286           calibrate->nvariables * sizeof (double));
02287   g_free (best_genome);
02288   g_free (best_variable);
02289   calibrate_print ();
02290 #if DEBUG
02291   fprintf (stderr, "calibrate_genetic: end\n");
02292 #endif
02293 }
02294
02299 void
02300 calibrate_save_old ()
02301 {
02302   unsigned int i, j;
02303 #if DEBUG
02304   fprintf (stderr, "calibrate_save_old: start\n");
02305   fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02306 #endif
02307   memcpy (calibrate->error_old, calibrate->error_best,
02308           calibrate->nbest * sizeof (double));
02309   for (i = 0; i < calibrate->nbest; ++i)
02310     {
02311       j = calibrate->simulation_best[i];
02312 #if DEBUG
02313       fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02314 #endif
02315       memcpy (calibrate->value_old + i * calibrate->nvariables,
02316               calibrate->value + j * calibrate->nvariables,
02317               calibrate->nvariables * sizeof (double));
02318     }
02319 #if DEBUG
02320   for (i = 0; i < calibrate->nvariables; ++i)
02321     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02322              i, calibrate->value_old[i]);
02323   fprintf (stderr, "calibrate_save_old: end\n");
02324 #endif
02325 }
02326
02332 void
02333 calibrate_merge_old ()
02334 {
02335   unsigned int i, j, k;
02336   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
02337     *enew, *eold;
02338 #if DEBUG
02339   fprintf (stderr, "calibrate_merge_old: start\n");
02340 #endif
02341   enew = calibrate->error_best;
02342   eold = calibrate->error_old;
02343   i = j = k = 0;
02344   do
02345     {
02346       if (*enew < *eold)
02347         {
02348           memcpy (v + k * calibrate->nvariables,
02349                   calibrate->value
02350                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
02351                   calibrate->nvariables * sizeof (double));
02352           e[k] = *enew;
02353           ++k;
02354           ++enew;
02355           ++i;
02356         }
02357       else
02358         {
02359           memcpy (v + k * calibrate->nvariables,
02360                   calibrate->value_old + j * calibrate->nvariables,
02361                   calibrate->nvariables * sizeof (double));
02362           e[k] = *eold;
02363           ++k;
02364           ++eold;
02365           ++j;
02366         }
02367     }
02368   while (k < calibrate->nbest);
02369   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02370   memcpy (calibrate->error_old, e, k * sizeof (double));
02371 #if DEBUG
02372   fprintf (stderr, "calibrate_merge_old: end\n");
02373 #endif
02374 }
02375
02381 void
02382 calibrate_refine ()
```

```
02383 {
02384   unsigned int i, j;
02385   double d;
02386 #if HAVE_MPI
02387   MPI_Status mpi_stat;
02388 #endif
02389 #if DEBUG
02390   fprintf (stderr, "calibrate_refine: start\n");
02391 #endif
02392 #if HAVE_MPI
02393   if (!calibrate->mpi_rank)
02394     {
02395 #endif
02396       for (j = 0; j < calibrate->nvariables; ++j)
02397         {
02398           calibrate->rangemin[j] = calibrate->rangemax[j]
02399             = calibrate->value_old[j];
02400         }
02401       for (i = 0; ++i < calibrate->nbest;)
02402         {
02403           for (j = 0; j < calibrate->nvariables; ++j)
02404             {
02405               calibrate->rangemin[j]
02406                 = fmin (calibrate->rangemin[j],
02407                         calibrate->value_old[i * calibrate->nvariables + j]);
02408               calibrate->rangemax[j]
02409                 = fmax (calibrate->rangemax[j],
02410                         calibrate->value_old[i * calibrate->nvariables + j]);
02411             }
02412         }
02413       for (j = 0; j < calibrate->nvariables; ++j)
02414         {
02415           d = calibrate->tolerance
02416             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02417           switch (calibrate->algorithm)
02418             {
02419             case ALGORITHM_MONTE_CARLO:
02420               d *= 0.5;
02421               break;
02422             default:
02423               if (calibrate->nsweeps[j] > 1)
02424                 d /= calibrate->nsweeps[j] - 1;
02425               else
02426                 d = 0.;
02427             }
02428           calibrate->rangemin[j] -= d;
02429           calibrate->rangemin[j]
02430             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02431           calibrate->rangemax[j] += d;
02432           calibrate->rangemax[j]
02433             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02434           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02435                   calibrate->rangemin[j], calibrate->rangemax[j]);
02436           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02437                    calibrate->label[j], calibrate->rangemin[j],
02438                    calibrate->rangemax[j]);
02439         }
02440 #if HAVE_MPI
02441       for (i = 1; i < ntasks; ++i)
02442         {
02443           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02444                     1, MPI_COMM_WORLD);
02445           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02446                     1, MPI_COMM_WORLD);
02447         }
02448     }
02449   else
02450     {
02451       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02452                 MPI_COMM_WORLD, &mpi_stat);
02453       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02454                 MPI_COMM_WORLD, &mpi_stat);
02455     }
02456 #endif
02457 #if DEBUG
02458   fprintf (stderr, "calibrate_refine: end\n");
02459 #endif
02460 }
02461
02466 void
02467 calibrate_step ()
02468 {
02469 #if DEBUG
02470   fprintf (stderr, "calibrate_step: start\n");
02471 #endif
02472   calibrate_algorithm ();
02473   if (calibrate->nsteps)
```

```
02474     calibrate_gradient ();
02475 #if DEBUG
02476   fprintf (stderr, "calibrate_step: end\n");
02477 #endif
02478 }
02479
02484 void
02485 calibrate_iterate ()
02486 {
02487   unsigned int i;
02488 #if DEBUG
02489   fprintf (stderr, "calibrate_iterate: start\n");
02490 #endif
02491   calibrate->error_old
02492     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02493   calibrate->value_old = (double *)
02494     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02495   calibrate_step ();
02496   calibrate_save_old ();
02497   calibrate_refine ();
02498   calibrate_print ();
02499   for (i = 1; i < calibrate->niterations; ++i)
02500     {
02501       calibrate_step ();
02502       calibrate_merge_old ();
02503       calibrate_refine ();
02504       calibrate_print ();
02505     }
02506 #if DEBUG
02507   fprintf (stderr, "calibrate_iterate: end\n");
02508 #endif
02509 }
02510
02515 void
02516 calibrate_free ()
02517 {
02518   unsigned int i, j;
02519 #if DEBUG
02520   fprintf (stderr, "calibrate_free: start\n");
02521 #endif
02522   for (j = 0; j < calibrate->ninputs; ++j)
02523     {
02524       for (i = 0; i < calibrate->nexperiments; ++i)
02525         g_mapped_file_unref (calibrate->file[j][i]);
02526       g_free (calibrate->file[j]);
02527     }
02528   g_free (calibrate->error_old);
02529   g_free (calibrate->value_old);
02530   g_free (calibrate->value);
02531   g_free (calibrate->genetic_variable);
02532   g_free (calibrate->rangemax);
02533   g_free (calibrate->rangemin);
02534 #if DEBUG
02535   fprintf (stderr, "calibrate_free: end\n");
02536 #endif
02537 }
02538
02543 void
02544 calibrate_open ()
02545 {
02546   GTimeZone *tz;
02547   GDateTime *t0, *t;
02548   unsigned int i, j, *nbits;
02549
02550 #if DEBUG
02551   char *buffer;
02552   fprintf (stderr, "calibrate_open: start\n");
02553 #endif
02554
02555   // Getting initial time
02556 #if DEBUG
02557   fprintf (stderr, "calibrate_open: getting initial time\n");
02558 #endif
02559   tz = g_time_zone_new_utc ();
02560   t0 = g_date_time_new_now (tz);
02561
02562   // Obtaining and initing the pseudo-random numbers generator seed
02563 #if DEBUG
02564   fprintf (stderr, "calibrate_open: getting initial seed\n");
02565 #endif
02566   calibrate->seed = input->seed;
02567   gsl_rng_set (calibrate->rng, calibrate->seed);
02568
02569   // Replacing the working directory
02570 #if DEBUG
02571   fprintf (stderr, "calibrate_open: replacing the working directory\n");
02572 #endif
```

```
02573   g_chdir (input->directory);
02574
02575   // Getting results file names
02576   calibrate->result = input->result;
02577   calibrate->variables = input->variables;
02578
02579   // Obtaining the simulator file
02580   calibrate->simulator = input->simulator;
02581
02582   // Obtaining the evaluator file
02583   calibrate->evaluator = input->evaluator;
02584
02585   // Reading the algorithm
02586   calibrate->algorithm = input->algorithm;
02587   switch (calibrate->algorithm)
02588     {
02589     case ALGORITHM_MONTE_CARLO:
02590       calibrate_algorithm = calibrate_MonteCarlo;
02591       break;
02592     case ALGORITHM_SWEEP:
02593       calibrate_algorithm = calibrate_sweep;
02594       break;
02595     default:
02596       calibrate_algorithm = calibrate_genetic;
02597       calibrate->mutation_ratio = input->mutation_ratio;
02598       calibrate->reproduction_ratio = input->
      reproduction_ratio;
02599       calibrate->adaptation_ratio = input->adaptation_ratio;
02600     }
02601   calibrate->nvariables = input->nvariables;
02602   calibrate->nsimulations = input->nsimulations;
02603   calibrate->niterations = input->niterations;
02604   calibrate->nbest = input->nbest;
02605   calibrate->tolerance = input->tolerance;
02606   calibrate->nsteps = input->nsteps;
02607   calibrate->nestimates = 0;
02608   if (input->nsteps)
02609     {
02610       calibrate->gradient_method = input->gradient_method;
02611       calibrate->relaxation = input->relaxation;
02612       switch (input->gradient_method)
02613         {
02614         case GRADIENT_METHOD_COORDINATES:
02615           calibrate->nestimates = 2 * calibrate->nvariables;
02616           calibrate_estimate_gradient =
      calibrate_estimate_gradient_coordinates;
02617           break;
02618         default:
02619           calibrate->nestimates = input->nestimates;
02620           calibrate_estimate_gradient =
      calibrate_estimate_gradient_random;
02621         }
02622     }
02623
02624 #if DEBUG
02625   fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02626 #endif
02627   calibrate->simulation_best
02628     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02629   calibrate->error_best
02630     = (double *) alloca (calibrate->nbest * sizeof (double));
02631
02632   // Reading the experimental data
02633 #if DEBUG
02634   buffer = g_get_current_dir ();
02635   fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02636   g_free (buffer);
02637 #endif
02638   calibrate->nexperiments = input->nexperiments;
02639   calibrate->ninputs = input->ninputs;
02640   calibrate->experiment = input->experiment;
02641   calibrate->weight = input->weight;
02642   for (i = 0; i < input->ninputs; ++i)
02643     {
02644       calibrate->template[i] = input->template[i];
02645       calibrate->file[i]
02646         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02647     }
02648   for (i = 0; i < input->nexperiments; ++i)
02649     {
02650 #if DEBUG
02651       fprintf (stderr, "calibrate_open: i=%u\n", i);
02652       fprintf (stderr, "calibrate_open: experiment=%s\n",
02653                calibrate->experiment[i]);
02654       fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
02655 #endif
02656       for (j = 0; j < input->ninputs; ++j)
```

```
02657         {
02658 #if DEBUG
02659           fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02660           fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
02661                    i, j + 1, calibrate->template[j][i]);
02662 #endif
02663           calibrate->file[j][i]
02664             = g_mapped_file_new (input->template[j][i], 0, NULL);
02665         }
02666     }
02667
02668   // Reading the variables data
02669 #if DEBUG
02670   fprintf (stderr, "calibrate_open: reading variables\n");
02671 #endif
02672   calibrate->label = input->label;
02673   j = input->nvariables * sizeof (double);
02674   calibrate->rangemin = (double *) g_malloc (j);
02675   calibrate->rangemax = (double *) g_malloc (j);
02676   memcpy (calibrate->rangemin, input->rangemin, j);
02677   memcpy (calibrate->rangemax, input->rangemax, j);
02678   calibrate->rangeminabs = input->rangeminabs;
02679   calibrate->rangemaxabs = input->rangemaxabs;
02680   calibrate->precision = input->precision;
02681   calibrate->nsweeps = input->nsweeps;
02682   calibrate->step = input->step;
02683   nbits = input->nbits;
02684   if (input->algorithm == ALGORITHM_SWEEP)
02685     {
02686       calibrate->nsimulations = 1;
02687       for (i = 0; i < input->nvariables; ++i)
02688         {
02689           if (input->algorithm == ALGORITHM_SWEEP)
02690             {
02691               calibrate->nsimulations *= input->nsweeps[i];
02692 #if DEBUG
02693               fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
02694                        calibrate->nsweeps[i], calibrate->nsimulations);
02695 #endif
02696             }
02697         }
02698     }
02699   if (calibrate->nsteps)
02700     calibrate->gradient
02701       = (double *) alloca (calibrate->nvariables * sizeof (double));
02702
02703   // Setting error norm
02704   switch (input->norm)
02705     {
02706     case ERROR_NORM_EUCLIDIAN:
02707       calibrate_norm = calibrate_norm_euclidian;
02708       break;
02709     case ERROR_NORM_MAXIMUM:
02710       calibrate_norm = calibrate_norm_maximum;
02711       break;
02712     case ERROR_NORM_P:
02713       calibrate_norm = calibrate_norm_p;
02714       calibrate->p = input->p;
02715       break;
02716     default:
02717       calibrate_norm = calibrate_norm_taxicab;
02718     }
02719
02720   // Allocating values
02721 #if DEBUG
02722   fprintf (stderr, "calibrate_open: allocating variables\n");
02723   fprintf (stderr, "calibrate_open: nvariables=%u\n", calibrate->nvariables);
02724 #endif
02725   calibrate->genetic_variable = NULL;
02726   if (calibrate->algorithm == ALGORITHM_GENETIC)
02727     {
02728       calibrate->genetic_variable = (GeneticVariable *)
02729         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02730       for (i = 0; i < calibrate->nvariables; ++i)
02731         {
02732 #if DEBUG
02733           fprintf (stderr, "calibrate_open: i=%u min=%lg max=%lg nbits=%u\n",
02734                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02735 #endif
02736           calibrate->genetic_variable[i].minimum = calibrate->
      rangemin[i];
02737           calibrate->genetic_variable[i].maximum = calibrate->
      rangemax[i];
02738           calibrate->genetic_variable[i].nbits = nbits[i];
02739         }
02740     }
02741 #if DEBUG
```

```
02742   fprintf (stderr, "calibrate_open: nvariables=%u nsimulations=%u\n",
02743            calibrate->nvariables, calibrate->nsimulations);
02744 #endif
02745   calibrate->value = (double *)
02746     g_malloc ((calibrate->nsimulations
02747                + calibrate->nestimates * calibrate->nsteps)
02748               * calibrate->nvariables * sizeof (double));
02749
02750   // Calculating simulations to perform for each task
02751 #if HAVE_MPI
02752 #if DEBUG
02753   fprintf (stderr, "calibrate_open: rank=%u ntasks=%u\n",
02754            calibrate->mpi_rank, ntasks);
02755 #endif
02756   calibrate->nstart = calibrate->mpi_rank * calibrate->
      nsimulations / ntasks;
02757   calibrate->nend
02758     = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
      ntasks;
02759   if (calibrate->nsteps)
02760     {
02761       calibrate->nstart_gradient
02762         = calibrate->mpi_rank * calibrate->nestimates / ntasks;
02763       calibrate->nend_gradient
02764         = (1 + calibrate->mpi_rank) * calibrate->nestimates /
      ntasks;
02765     }
02766 #else
02767   calibrate->nstart = 0;
02768   calibrate->nend = calibrate->nsimulations;
02769   if (calibrate->nsteps)
02770     {
02771       calibrate->nstart_gradient = 0;
02772       calibrate->nend_gradient = calibrate->nestimates;
02773     }
02774 #endif
02775 #if DEBUG
02776   fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02777            calibrate->nend);
02778 #endif
02779
02780   // Calculating simulations to perform for each thread
02781   calibrate->thread
02782     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02783   for (i = 0; i <= nthreads; ++i)
02784     {
02785       calibrate->thread[i] = calibrate->nstart
02786         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02787 #if DEBUG
02788       fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02789                calibrate->thread[i]);
02790 #endif
02791     }
02792   if (calibrate->nsteps)
02793     calibrate->thread_gradient = (unsigned int *)
02794       alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02795
02796   // Opening result files
02797   calibrate->file_result = g_fopen (calibrate->result, "w");
02798   calibrate->file_variables = g_fopen (calibrate->variables, "w");
02799
02800   // Performing the algorithm
02801   switch (calibrate->algorithm)
02802     {
02803       // Genetic algorithm
02804     case ALGORITHM_GENETIC:
02805       calibrate_genetic ();
02806       break;
02807
02808       // Iterative algorithm
02809     default:
02810       calibrate_iterate ();
02811     }
02812
02813   // Getting calculation time
02814   t = g_date_time_new_now (tz);
02815   calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02816   g_date_time_unref (t);
02817   g_date_time_unref (t0);
02818   g_time_zone_unref (tz);
02819   printf ("%s = %.6lg s\n",
02820           gettext ("Calculation time"), calibrate->calculation_time);
02821   fprintf (calibrate->file_result, "%s = %.6lg s\n",
02822            gettext ("Calculation time"), calibrate->calculation_time);
02823
02824   // Closing result files
02825   fclose (calibrate->file_variables);
```

```
02826   fclose (calibrate->file_result);
02827
02828 #if DEBUG
02829   fprintf (stderr, "calibrate_open: end\n");
02830 #endif
02831 }
02832
02833 #if HAVE_GTK
02834
02841 void
02842 input_save_gradient (xmlNode * node)
02843 {
02844 #if DEBUG
02845   fprintf (stderr, "input_save_gradient: start\n");
02846 #endif
02847   if (input->nsteps)
02848     {
02849       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
02850       if (input->relaxation != DEFAULT_RELAXATION)
02851         xml_node_set_float (node, XML_RELAXATION, input->
      relaxation);
02852       switch (input->gradient_method)
02853         {
02854         case GRADIENT_METHOD_COORDINATES:
02855           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02856           break;
02857         default:
02858           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02859           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
02860         }
02861     }
02862 #if DEBUG
02863   fprintf (stderr, "input_save_gradient: end\n");
02864 #endif
02865 }
02866
02873 void
02874 input_save (char *filename)
02875 {
02876   unsigned int i, j;
02877   char *buffer;
02878   xmlDoc *doc;
02879   xmlNode *node, *child;
02880   GFile *file, *file2;
02881
02882 #if DEBUG
02883   fprintf (stderr, "input_save: start\n");
02884 #endif
02885
02886   // Getting the input file directory
02887   input->name = g_path_get_basename (filename);
02888   input->directory = g_path_get_dirname (filename);
02889   file = g_file_new_for_path (input->directory);
02890
02891   // Opening the input file
02892   doc = xmlNewDoc ((const xmlChar *) "1.0");
02893
02894   // Setting root XML node
02895   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02896   xmlDocSetRootElement (doc, node);
02897
02898   // Adding properties to the root XML node
02899   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02900     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02901   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02902     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02903   file2 = g_file_new_for_path (input->simulator);
02904   buffer = g_file_get_relative_path (file, file2);
02905   g_object_unref (file2);
02906   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02907   g_free (buffer);
02908   if (input->evaluator)
02909     {
02910       file2 = g_file_new_for_path (input->evaluator);
02911       buffer = g_file_get_relative_path (file, file2);
02912       g_object_unref (file2);
02913       if (xmlStrlen ((xmlChar *) buffer))
02914         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02915       g_free (buffer);
02916     }
02917   if (input->seed != DEFAULT_RANDOM_SEED)
02918     xml_node_set_uint (node, XML_SEED, input->seed);
02919
02920   // Setting the algorithm
```

```
02921   buffer = (char *) g_malloc (64);
02922   switch (input->algorithm)
02923     {
02924     case ALGORITHM_MONTE_CARLO:
02925       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02926       snprintf (buffer, 64, "%u", input->nsimulations);
02927       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02928       snprintf (buffer, 64, "%u", input->niterations);
02929       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02930       snprintf (buffer, 64, "%.3lg", input->tolerance);
02931       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02932       snprintf (buffer, 64, "%u", input->nbest);
02933       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02934       input_save_gradient (node);
02935       break;
02936     case ALGORITHM_SWEEP:
02937       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02938       snprintf (buffer, 64, "%u", input->niterations);
02939       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02940       snprintf (buffer, 64, "%.3lg", input->tolerance);
02941       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02942       snprintf (buffer, 64, "%u", input->nbest);
02943       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02944       input_save_gradient (node);
02945       break;
02946     default:
02947       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02948       snprintf (buffer, 64, "%u", input->nsimulations);
02949       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02950       snprintf (buffer, 64, "%u", input->niterations);
02951       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02952       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02953       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02954       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02955       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02956       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02957       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02958       break;
02959     }
02960   g_free (buffer);
02961
02962   // Setting the experimental data
02963   for (i = 0; i < input->nexperiments; ++i)
02964     {
02965       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02966       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02967       if (input->weight[i] != 1.)
02968         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02969       for (j = 0; j < input->ninputs; ++j)
02970         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02971     }
02972
02973   // Setting the variables data
02974   for (i = 0; i < input->nvariables; ++i)
02975     {
02976       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02977       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02978       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02979       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02980         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02981       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02982       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02983         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02984       if (input->precision[i] != DEFAULT_PRECISION)
02985         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02986       if (input->algorithm == ALGORITHM_SWEEP)
02987         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02988       else if (input->algorithm == ALGORITHM_GENETIC)
02989         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02990       if (input->nsteps)
02991         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02992     }
02993
02994   // Saving the error norm
02995   switch (input->norm)
02996     {
02997     case ERROR_NORM_MAXIMUM:
02998       xmlSetProp (node, XML_NORM, XML_MAXIMUM);
```

```
02999       break;
03000     case ERROR_NORM_P:
03001       xmlSetProp (node, XML_NORM, XML_P);
03002       xml_node_set_float (node, XML_P, input->p);
03003       break;
03004     case ERROR_NORM_TAXICAB:
03005       xmlSetProp (node, XML_NORM, XML_TAXICAB);
03006     }
03007
03008   // Saving the XML file
03009   xmlSaveFormatFile (filename, doc, 1);
03010
03011   // Freeing memory
03012   xmlFreeDoc (doc);
03013
03014 #if DEBUG
03015   fprintf (stderr, "input_save: end\n");
03016 #endif
03017 }
03018
03023 void
03024 options_new ()
03025 {
03026 #if DEBUG
03027   fprintf (stderr, "options_new: start\n");
03028 #endif
03029   options->label_seed = (GtkLabel *)
03030     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
03031   options->spin_seed = (GtkSpinButton *)
03032     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
03033   gtk_widget_set_tooltip_text
03034     (GTK_WIDGET (options->spin_seed),
03035      gettext ("Seed to init the pseudo-random numbers generator"));
03036   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
03037   options->label_threads = (GtkLabel *)
03038     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
03039   options->spin_threads
03040     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03041   gtk_widget_set_tooltip_text
03042     (GTK_WIDGET (options->spin_threads),
03043      gettext ("Number of threads to perform the calibration/optimization for "
03044               "the stochastic algorithm"));
03045   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
03046   options->label_gradient = (GtkLabel *)
03047     gtk_label_new (gettext ("Threads number for the gradient based method"));
03048   options->spin_gradient
03049     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03050   gtk_widget_set_tooltip_text
03051     (GTK_WIDGET (options->spin_gradient),
03052      gettext ("Number of threads to perform the calibration/optimization for "
03053               "the gradient based method"));
03054   gtk_spin_button_set_value (options->spin_gradient,
03055                              (gdouble) nthreads_gradient);
03056   options->grid = (GtkGrid *) gtk_grid_new ();
03057   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
03058   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
03059   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
03060                    0, 1, 1, 1);
03061   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
03062                    1, 1, 1, 1);
03063   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
03064                    0, 2, 1, 1);
03065   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
03066                    1, 2, 1, 1);
03067   gtk_widget_show_all (GTK_WIDGET (options->grid));
03068   options->dialog = (GtkDialog *)
03069     gtk_dialog_new_with_buttons (gettext ("Options"),
03070                                  window->window,
03071                                  GTK_DIALOG_MODAL,
03072                                  gettext ("_OK"), GTK_RESPONSE_OK,
03073                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03074                                  NULL);
03075   gtk_container_add
03076     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
03077      GTK_WIDGET (options->grid));
03078   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
03079     {
03080       input->seed
03081         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
03082       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
03083       nthreads_gradient
03084         = gtk_spin_button_get_value_as_int (options->spin_gradient);
03085     }
03086   gtk_widget_destroy (GTK_WIDGET (options->dialog));
03087 #if DEBUG
03088   fprintf (stderr, "options_new: end\n");
```

```
03089 #endif
03090 }
03091
03096 void
03097 running_new ()
03098 {
03099 #if DEBUG
03100   fprintf (stderr, "running_new: start\n");
03101 #endif
03102   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
03103   running->dialog = (GtkDialog *)
03104     gtk_dialog_new_with_buttons (gettext ("Calculating"),
03105                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
03106   gtk_container_add
03107     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
03108      GTK_WIDGET (running->label));
03109   gtk_widget_show_all (GTK_WIDGET (running->dialog));
03110 #if DEBUG
03111   fprintf (stderr, "running_new: end\n");
03112 #endif
03113 }
03114
03120 unsigned int
03121 window_get_algorithm ()
03122 {
03123   unsigned int i;
03124 #if DEBUG
03125   fprintf (stderr, "window_get_algorithm: start\n");
03126 #endif
03127   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
03128 #if DEBUG
03129   fprintf (stderr, "window_get_algorithm: %u\n", i);
03130   fprintf (stderr, "window_get_algorithm: end\n");
03131 #endif
03132   return i;
03133 }
03134
03140 unsigned int
03141 window_get_gradient ()
03142 {
03143   unsigned int i;
03144 #if DEBUG
03145   fprintf (stderr, "window_get_gradient: start\n");
03146 #endif
03147   i = gtk_array_get_active (window->button_gradient ,
      NGRADIENTS);
03148 #if DEBUG
03149   fprintf (stderr, "window_get_gradient: %u\n", i);
03150   fprintf (stderr, "window_get_gradient: end\n");
03151 #endif
03152   return i;
03153 }
03154
03160 unsigned int
03161 window_get_norm ()
03162 {
03163   unsigned int i;
03164 #if DEBUG
03165   fprintf (stderr, "window_get_norm: start\n");
03166 #endif
03167   i = gtk_array_get_active (window->button_norm ,
      NNORMS);
03168 #if DEBUG
03169   fprintf (stderr, "window_get_norm: %u\n", i);
03170   fprintf (stderr, "window_get_norm: end\n");
03171 #endif
03172   return i;
03173 }
03174
03179 void
03180 window_save_gradient ()
03181 {
03182 #if DEBUG
03183   fprintf (stderr, "window_save_gradient: start\n");
03184 #endif
03185   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03186     {
03187       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
03188       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
03189       switch (window_get_gradient ())
03190         {
03191         case GRADIENT_METHOD_COORDINATES:
03192           input->gradient_method = GRADIENT_METHOD_COORDINATES;
03193           break;
03194         default:
```

```
03195           input->gradient_method = GRADIENT_METHOD_RANDOM;
03196           input->nestimates
03197             = gtk_spin_button_get_value_as_int (window->spin_estimates);
03198         }
03199     }
03200   else
03201     input->nsteps = 0;
03202 #if DEBUG
03203   fprintf (stderr, "window_save_gradient: end\n");
03204 #endif
03205 }
03206
03212 int
03213 window_save ()
03214 {
03215   GtkFileChooserDialog *dlg;
03216   GtkFileFilter *filter;
03217   char *buffer;
03218
03219 #if DEBUG
03220   fprintf (stderr, "window_save: start\n");
03221 #endif
03222
03223   // Opening the saving dialog
03224   dlg = (GtkFileChooserDialog *)
03225     gtk_file_chooser_dialog_new (gettext ("Save file"),
03226                                  window->window,
03227                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03228                                  gettext ("_Cancel"),
03229                                  GTK_RESPONSE_CANCEL,
03230                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03231   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03232   buffer = g_build_filename (input->directory, input->name, NULL);
03233   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03234   g_free (buffer);
03235
03236   // Adding XML filter
03237   filter = (GtkFileFilter *) gtk_file_filter_new ();
03238   gtk_file_filter_set_name (filter, "XML");
03239   gtk_file_filter_add_pattern (filter, "*.xml");
03240   gtk_file_filter_add_pattern (filter, "*.XML");
03241   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03242
03243   // If OK response then saving
03244   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03245     {
03246
03247       // Adding properties to the root XML node
03248       input->simulator = gtk_file_chooser_get_filename
03249         (GTK_FILE_CHOOSER (window->button_simulator));
03250       if (gtk_toggle_button_get_active
03251           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03252         input->evaluator = gtk_file_chooser_get_filename
03253           (GTK_FILE_CHOOSER (window->button_evaluator));
03254       else
03255         input->evaluator = NULL;
03256       input->result
03257         = (char *) xmlStrdup ((const xmlChar *)
03258                               gtk_entry_get_text (window->entry_result));
03259       input->variables
03260         = (char *) xmlStrdup ((const xmlChar *)
03261                               gtk_entry_get_text (window->entry_variables));
03262
03263       // Setting the algorithm
03264       switch (window_get_algorithm ())
03265         {
03266         case ALGORITHM_MONTE_CARLO:
03267           input->algorithm = ALGORITHM_MONTE_CARLO;
03268           input->nsimulations
03269             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03270           input->niterations
03271             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03272           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03273           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03274           window_save_gradient ();
03275           break;
03276         case ALGORITHM_SWEEP:
03277           input->algorithm = ALGORITHM_SWEEP;
03278           input->niterations
03279             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03280           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03281           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03282           window_save_gradient ();
```

```
03283           break;
03284         default:
03285           input->algorithm = ALGORITHM_GENETIC;
03286           input->nsimulations
03287             = gtk_spin_button_get_value_as_int (window->spin_population);
03288           input->niterations
03289             = gtk_spin_button_get_value_as_int (window->spin_generations);
03290           input->mutation_ratio
03291             = gtk_spin_button_get_value (window->spin_mutation);
03292           input->reproduction_ratio
03293             = gtk_spin_button_get_value (window->spin_reproduction);
03294           input->adaptation_ratio
03295             = gtk_spin_button_get_value (window->spin_adaptation);
03296           break;
03297         }
03298       input->norm = window_get_norm ();
03299       input->p = gtk_spin_button_get_value (window->spin_p);
03300
03301       // Saving the XML file
03302       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03303       input_save (buffer);
03304
03305       // Closing and freeing memory
03306       g_free (buffer);
03307       gtk_widget_destroy (GTK_WIDGET (dlg));
03308 #if DEBUG
03309       fprintf (stderr, "window_save: end\n");
03310 #endif
03311       return 1;
03312     }
03313
03314   // Closing and freeing memory
03315   gtk_widget_destroy (GTK_WIDGET (dlg));
03316 #if DEBUG
03317   fprintf (stderr, "window_save: end\n");
03318 #endif
03319   return 0;
03320 }
03321
03326 void
03327 window_run ()
03328 {
03329   unsigned int i;
03330   char *msg, *msg2, buffer[64], buffer2[64];
03331 #if DEBUG
03332   fprintf (stderr, "window_run: start\n");
03333 #endif
03334   if (!window_save ())
03335     {
03336 #if DEBUG
03337       fprintf (stderr, "window_run: end\n");
03338 #endif
03339       return;
03340     }
03341   running_new ();
03342   while (gtk_events_pending ())
03343     gtk_main_iteration ();
03344   calibrate_open ();
03345   gtk_widget_destroy (GTK_WIDGET (running->dialog));
03346   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03347   msg2 = g_strdup (buffer);
03348   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03349     {
03350       snprintf (buffer, 64, "%s = %s\n",
03351                 calibrate->label[i], format[calibrate->precision[i]]);
03352       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03353       msg = g_strconcat (msg2, buffer2, NULL);
03354       g_free (msg2);
03355     }
03356   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03357             calibrate->calculation_time);
03358   msg = g_strconcat (msg2, buffer, NULL);
03359   g_free (msg2);
03360   show_message (gettext ("Best result"), msg, INFO_TYPE);
03361   g_free (msg);
03362   calibrate_free ();
03363 #if DEBUG
03364   fprintf (stderr, "window_run: end\n");
03365 #endif
03366 }
03367
03372 void
03373 window_help ()
03374 {
03375   char *buffer, *buffer2;
03376 #if DEBUG
03377   fprintf (stderr, "window_help: start\n");
```

```
03378 #endif
03379   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03380                               gettext ("user-manual.pdf"), NULL);
03381   buffer = g_filename_to_uri (buffer2, NULL, NULL);
03382   g_free (buffer2);
03383   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03384 #if DEBUG
03385   fprintf (stderr, "window_help: uri=%s\n", buffer);
03386 #endif
03387   g_free (buffer);
03388 #if DEBUG
03389   fprintf (stderr, "window_help: end\n");
03390 #endif
03391 }
03392
03397 void
03398 window_about ()
03399 {
03400   static const gchar *authors[] = {
03401     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03402     "Borja Latorre Garcés <borja.latorre@csic.es>",
03403     NULL
03404   };
03405 #if DEBUG
03406   fprintf (stderr, "window_about: start\n");
03407 #endif
03408   gtk_show_about_dialog
03409     (window->window,
03410      "program_name", "MPCOTool",
03411      "comments",
03412      gettext ("A software to perform calibrations/optimizations of empirical "
03413               "parameters"),
03414      "authors", authors,
03415      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03416      "version", "1.5.3",
03417      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
03418      "logo", window->logo,
03419      "website", "https://github.com/jburguete/mpcotool",
03420      "license-type", GTK_LICENSE_BSD, NULL);
03421 #if DEBUG
03422   fprintf (stderr, "window_about: end\n");
03423 #endif
03424 }
03425
03431 void
03432 window_update_gradient ()
03433 {
03434 #if DEBUG
03435   fprintf (stderr, "window_update_gradient: start\n");
03436 #endif
03437   gtk_widget_show (GTK_WIDGET (window->check_gradient));
03438   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03439     {
03440       gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03441       gtk_widget_show (GTK_WIDGET (window->label_step));
03442       gtk_widget_show (GTK_WIDGET (window->spin_step));
03443     }
03444   switch (window_get_gradient ())
03445     {
03446     case GRADIENT_METHOD_COORDINATES:
03447       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03448       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03449       break;
03450     default:
03451       gtk_widget_show (GTK_WIDGET (window->label_estimates));
03452       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03453     }
03454 #if DEBUG
03455   fprintf (stderr, "window_update_gradient: end\n");
03456 #endif
03457 }
03458
03463 void
03464 window_update ()
03465 {
03466   unsigned int i;
03467 #if DEBUG
03468   fprintf (stderr, "window_update: start\n");
03469 #endif
03470   gtk_widget_set_sensitive
03471     (GTK_WIDGET (window->button_evaluator),
03472      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03473                                    (window->check_evaluator)));
03474   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03475   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03476   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03477   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
```

```
03478   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03479   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03480   gtk_widget_hide (GTK_WIDGET (window->label_bests));
03481   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03482   gtk_widget_hide (GTK_WIDGET (window->label_population));
03483   gtk_widget_hide (GTK_WIDGET (window->spin_population));
03484   gtk_widget_hide (GTK_WIDGET (window->label_generations));
03485   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03486   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03487   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03488   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03489   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03490   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03491   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03492   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03493   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03494   gtk_widget_hide (GTK_WIDGET (window->label_bits));
03495   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03496   gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03497   gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03498   gtk_widget_hide (GTK_WIDGET (window->label_step));
03499   gtk_widget_hide (GTK_WIDGET (window->spin_step));
03500   gtk_widget_hide (GTK_WIDGET (window->label_p));
03501   gtk_widget_hide (GTK_WIDGET (window->spin_p));
03502   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03503   switch (window_get_algorithm ())
03504     {
03505     case ALGORITHM_MONTE_CARLO:
03506       gtk_widget_show (GTK_WIDGET (window->label_simulations));
03507       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03508       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03509       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03510       if (i > 1)
03511         {
03512           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03513           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03514           gtk_widget_show (GTK_WIDGET (window->label_bests));
03515           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03516         }
03517       window_update_gradient ();
03518       break;
03519     case ALGORITHM_SWEEP:
03520       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03521       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03522       if (i > 1)
03523         {
03524           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03525           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03526           gtk_widget_show (GTK_WIDGET (window->label_bests));
03527           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03528         }
03529       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03530       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03531       gtk_widget_show (GTK_WIDGET (window->check_gradient));
03532       window_update_gradient ();
03533       break;
03534     default:
03535       gtk_widget_show (GTK_WIDGET (window->label_population));
03536       gtk_widget_show (GTK_WIDGET (window->spin_population));
03537       gtk_widget_show (GTK_WIDGET (window->label_generations));
03538       gtk_widget_show (GTK_WIDGET (window->spin_generations));
03539       gtk_widget_show (GTK_WIDGET (window->label_mutation));
03540       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03541       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03542       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03543       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03544       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03545       gtk_widget_show (GTK_WIDGET (window->label_bits));
03546       gtk_widget_show (GTK_WIDGET (window->spin_bits));
03547     }
03548   gtk_widget_set_sensitive
03549     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
03550   gtk_widget_set_sensitive
03551     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
03552   for (i = 0; i < input->ninputs; ++i)
03553     {
03554       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03555       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03556       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03557       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03558       g_signal_handler_block
03559         (window->check_template[i], window->id_template[i]);
03560       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
03561       gtk_toggle_button_set_active
```

```
03562         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03563       g_signal_handler_unblock
03564         (window->button_template[i], window->id_input[i]);
03565       g_signal_handler_unblock
03566         (window->check_template[i], window->id_template[i]);
03567     }
03568   if (i > 0)
03569     {
03570       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03571       gtk_widget_set_sensitive
03572         (GTK_WIDGET (window->button_template[i - 1]),
03573          gtk_toggle_button_get_active
03574          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
03575     }
03576   if (i < MAX_NINPUTS)
03577     {
03578       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03579       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03580       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03581       gtk_widget_set_sensitive
03582         (GTK_WIDGET (window->button_template[i]),
03583          gtk_toggle_button_get_active
03584          GTK_TOGGLE_BUTTON (window->check_template[i]));
03585       g_signal_handler_block
03586         (window->check_template[i], window->id_template[i]);
03587       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
03588       gtk_toggle_button_set_active
03589         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03590       g_signal_handler_unblock
03591         (window->button_template[i], window->id_input[i]);
03592       g_signal_handler_unblock
03593         (window->check_template[i], window->id_template[i]);
03594     }
03595   while (++i < MAX_NINPUTS)
03596     {
03597       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03598       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03599     }
03600   gtk_widget_set_sensitive
03601     (GTK_WIDGET (window->spin_minabs),
03602      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03603   gtk_widget_set_sensitive
03604     (GTK_WIDGET (window->spin_maxabs),
03605      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03606   if (window_get_norm () == ERROR_NORM_P)
03607     {
03608       gtk_widget_show (GTK_WIDGET (window->label_p));
03609       gtk_widget_show (GTK_WIDGET (window->spin_p));
03610     }
03611 #if DEBUG
03612   fprintf (stderr, "window_update: end\n");
03613 #endif
03614 }
03615
03620 void
03621 window_set_algorithm ()
03622 {
03623   int i;
03624 #if DEBUG
03625   fprintf (stderr, "window_set_algorithm: start\n");
03626 #endif
03627   i = window_get_algorithm ();
03628   switch (i)
03629     {
03630     case ALGORITHM_SWEEP:
03631       input->nsweeps = (unsigned int *) g_realloc
03632         (input->nsweeps, input->nvariables * sizeof (unsigned int));
03633       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03634       if (i < 0)
03635         i = 0;
03636       gtk_spin_button_set_value (window->spin_sweeps,
03637                                  (gdouble) input->nsweeps[i]);
03638       break;
03639     case ALGORITHM_GENETIC:
03640       input->nbits = (unsigned int *) g_realloc
03641         (input->nbits, input->nvariables * sizeof (unsigned int));
03642       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03643       if (i < 0)
03644         i = 0;
03645       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03646     }
03647   window_update ();
03648 #if DEBUG
03649   fprintf (stderr, "window_set_algorithm: end\n");
03650 #endif
```

```
03651 }
03652
03657 void
03658 window_set_experiment ()
03659 {
03660   unsigned int i, j;
03661   char *buffer1, *buffer2;
03662 #if DEBUG
03663   fprintf (stderr, "window_set_experiment: start\n");
03664 #endif
03665   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03666   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03667   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03668   buffer2 = g_build_filename (input->directory, buffer1, NULL);
03669   g_free (buffer1);
03670   g_signal_handler_block
03671     (window->button_experiment, window->id_experiment_name);
03672   gtk_file_chooser_set_filename
03673     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03674   g_signal_handler_unblock
03675     (window->button_experiment, window->id_experiment_name);
03676   g_free (buffer2);
03677   for (j = 0; j < input->ninputs; ++j)
03678     {
03679       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03680       buffer2
03681         = g_build_filename (input->directory, input->template[j][i], NULL);
03682       gtk_file_chooser_set_filename
03683         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03684       g_free (buffer2);
03685       g_signal_handler_unblock
03686         (window->button_template[j], window->id_input[j]);
03687     }
03688 #if DEBUG
03689   fprintf (stderr, "window_set_experiment: end\n");
03690 #endif
03691 }
03692
03697 void
03698 window_remove_experiment ()
03699 {
03700   unsigned int i, j;
03701 #if DEBUG
03702   fprintf (stderr, "window_remove_experiment: start\n");
03703 #endif
03704   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03705   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03706   gtk_combo_box_text_remove (window->combo_experiment, i);
03707   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03708   xmlFree (input->experiment[i]);
03709   --input->nexperiments;
03710   for (j = i; j < input->nexperiments; ++j)
03711     {
03712       input->experiment[j] = input->experiment[j + 1];
03713       input->weight[j] = input->weight[j + 1];
03714     }
03715   j = input->nexperiments - 1;
03716   if (i > j)
03717     i = j;
03718   for (j = 0; j < input->ninputs; ++j)
03719     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03720   g_signal_handler_block
03721     (window->button_experiment, window->id_experiment_name);
03722   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03723   g_signal_handler_unblock
03724     (window->button_experiment, window->id_experiment_name);
03725   for (j = 0; j < input->ninputs; ++j)
03726     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03727   window_update ();
03728 #if DEBUG
03729   fprintf (stderr, "window_remove_experiment: end\n");
03730 #endif
03731 }
03732
03737 void
03738 window_add_experiment ()
03739 {
03740   unsigned int i, j;
03741 #if DEBUG
03742   fprintf (stderr, "window_add_experiment: start\n");
03743 #endif
03744   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
```

```
03745   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03746   gtk_combo_box_text_insert_text
03747     (window->combo_experiment, i, input->experiment[i]);
03748   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03749   input->experiment = (char **) g_realloc
03750     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03751   input->weight = (double *) g_realloc
03752     (input->weight, (input->nexperiments + 1) * sizeof (double));
03753   for (j = input->nexperiments - 1; j > i; --j)
03754     {
03755       input->experiment[j + 1] = input->experiment[j];
03756       input->weight[j + 1] = input->weight[j];
03757     }
03758   input->experiment[j + 1]
03759     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03760   input->weight[j + 1] = input->weight[j];
03761   ++input->nexperiments;
03762   for (j = 0; j < input->ninputs; ++j)
03763     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03764   g_signal_handler_block
03765     (window->button_experiment, window->id_experiment_name);
03766   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03767   g_signal_handler_unblock
03768     (window->button_experiment, window->id_experiment_name);
03769   for (j = 0; j < input->ninputs; ++j)
03770     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03771   window_update ();
03772 #if DEBUG
03773   fprintf (stderr, "window_add_experiment: end\n");
03774 #endif
03775 }
03776
03781 void
03782 window_name_experiment ()
03783 {
03784   unsigned int i;
03785   char *buffer;
03786   GFile *file1, *file2;
03787 #if DEBUG
03788   fprintf (stderr, "window_name_experiment: start\n");
03789 #endif
03790   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03791   file1
03792     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03793   file2 = g_file_new_for_path (input->directory);
03794   buffer = g_file_get_relative_path (file2, file1);
03795   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03796   gtk_combo_box_text_remove (window->combo_experiment, i);
03797   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03798   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03799   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03800   g_free (buffer);
03801   g_object_unref (file2);
03802   g_object_unref (file1);
03803 #if DEBUG
03804   fprintf (stderr, "window_name_experiment: end\n");
03805 #endif
03806 }
03807
03812 void
03813 window_weight_experiment ()
03814 {
03815   unsigned int i;
03816 #if DEBUG
03817   fprintf (stderr, "window_weight_experiment: start\n");
03818 #endif
03819   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03820   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03821 #if DEBUG
03822   fprintf (stderr, "window_weight_experiment: end\n");
03823 #endif
03824 }
03825
03831 void
03832 window_inputs_experiment ()
03833 {
03834   unsigned int j;
03835 #if DEBUG
03836   fprintf (stderr, "window_inputs_experiment: start\n");
03837 #endif
03838   j = input->ninputs - 1;
```

```
03839   if (j
03840       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03841                                         (window->check_template[j])))
03842     --input->ninputs;
03843   if (input->ninputs < MAX_NINPUTS
03844       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03845                                        (window->check_template[j])))
03846     {
03847       ++input->ninputs;
03848       for (j = 0; j < input->ninputs; ++j)
03849         {
03850           input->template[j] = (char **)
03851             g_realloc (input->template[j], input->nvariables * sizeof (char *));
03852         }
03853     }
03854   window_update ();
03855 #if DEBUG
03856   fprintf (stderr, "window_inputs_experiment: end\n");
03857 #endif
03858 }
03859
03867 void
03868 window_template_experiment (void *data)
03869 {
03870   unsigned int i, j;
03871   char *buffer;
03872   GFile *file1, *file2;
03873 #if DEBUG
03874   fprintf (stderr, "window_template_experiment: start\n");
03875 #endif
03876   i = (size_t) data;
03877   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03878   file1
03879     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03880   file2 = g_file_new_for_path (input->directory);
03881   buffer = g_file_get_relative_path (file2, file1);
03882   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03883   g_free (buffer);
03884   g_object_unref (file2);
03885   g_object_unref (file1);
03886 #if DEBUG
03887   fprintf (stderr, "window_template_experiment: end\n");
03888 #endif
03889 }
03890
03895 void
03896 window_set_variable ()
03897 {
03898   unsigned int i;
03899 #if DEBUG
03900   fprintf (stderr, "window_set_variable: start\n");
03901 #endif
03902   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03903   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
03904   gtk_entry_set_text (window->entry_variable, input->label[i]);
03905   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
03906   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03907   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03908   if (input->rangeminabs[i] != -G_MAXDOUBLE)
03909     {
03910       gtk_spin_button_set_value (window->spin_minabs, input->
    rangeminabs[i]);
03911       gtk_toggle_button_set_active
03912         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03913     }
03914   else
03915     {
03916       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03917       gtk_toggle_button_set_active
03918         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03919     }
03920   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03921     {
03922       gtk_spin_button_set_value (window->spin_maxabs, input->
    rangemaxabs[i]);
03923       gtk_toggle_button_set_active
03924         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03925     }
03926   else
03927     {
03928       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03929       gtk_toggle_button_set_active
03930         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03931     }
03932   gtk_spin_button_set_value (window->spin_precision, input->
```

```
      precision[i]);
03933   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
03934   if (input->nsteps)
03935     gtk_spin_button_set_value (window->spin_step, input->step[i]);
03936 #if DEBUG
03937   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03938            input->precision[i]);
03939 #endif
03940   switch (window_get_algorithm ())
03941     {
03942     case ALGORITHM_SWEEP:
03943       gtk_spin_button_set_value (window->spin_sweeps,
03944                                  (gdouble) input->nsweeps[i]);
03945 #if DEBUG
03946       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03947                input->nsweeps[i]);
03948 #endif
03949       break;
03950     case ALGORITHM_GENETIC:
03951       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03952 #if DEBUG
03953       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03954                input->nbits[i]);
03955 #endif
03956       break;
03957     }
03958   window_update ();
03959 #if DEBUG
03960   fprintf (stderr, "window_set_variable: end\n");
03961 #endif
03962 }
03963
03968 void
03969 window_remove_variable ()
03970 {
03971   unsigned int i, j;
03972 #if DEBUG
03973   fprintf (stderr, "window_remove_variable: start\n");
03974 #endif
03975   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03976   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03977   gtk_combo_box_text_remove (window->combo_variable, i);
03978   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03979   xmlFree (input->label[i]);
03980   --input->nvariables;
03981   for (j = i; j < input->nvariables; ++j)
03982     {
03983       input->label[j] = input->label[j + 1];
03984       input->rangemin[j] = input->rangemin[j + 1];
03985       input->rangemax[j] = input->rangemax[j + 1];
03986       input->rangeminabs[j] = input->rangeminabs[j + 1];
03987       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03988       input->precision[j] = input->precision[j + 1];
03989       input->step[j] = input->step[j + 1];
03990       switch (window_get_algorithm ())
03991         {
03992         case ALGORITHM_SWEEP:
03993           input->nsweeps[j] = input->nsweeps[j + 1];
03994           break;
03995         case ALGORITHM_GENETIC:
03996           input->nbits[j] = input->nbits[j + 1];
03997         }
03998     }
03999   j = input->nvariables - 1;
04000   if (i > j)
04001     i = j;
04002   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04003   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
04004   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04005   window_update ();
04006 #if DEBUG
04007   fprintf (stderr, "window_remove_variable: end\n");
04008 #endif
04009 }
04010
04015 void
04016 window_add_variable ()
04017 {
04018   unsigned int i, j;
04019 #if DEBUG
04020   fprintf (stderr, "window_add_variable: start\n");
```

```
04021 #endif
04022   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04023   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04024   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
04025   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04026   input->label = (char **) g_realloc
04027     (input->label, (input->nvariables + 1) * sizeof (char *));
04028   input->rangemin = (double *) g_realloc
04029     (input->rangemin, (input->nvariables + 1) * sizeof (double));
04030   input->rangemax = (double *) g_realloc
04031     (input->rangemax, (input->nvariables + 1) * sizeof (double));
04032   input->rangeminabs = (double *) g_realloc
04033     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
04034   input->rangemaxabs = (double *) g_realloc
04035     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
04036   input->precision = (unsigned int *) g_realloc
04037     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
04038   input->step = (double *) g_realloc
04039     (input->step, (input->nvariables + 1) * sizeof (double));
04040   for (j = input->nvariables - 1; j > i; --j)
04041     {
04042       input->label[j + 1] = input->label[j];
04043       input->rangemin[j + 1] = input->rangemin[j];
04044       input->rangemax[j + 1] = input->rangemax[j];
04045       input->rangeminabs[j + 1] = input->rangeminabs[j];
04046       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
04047       input->precision[j + 1] = input->precision[j];
04048       input->step[j + 1] = input->step[j];
04049     }
04050   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
04051   input->rangemin[j + 1] = input->rangemin[j];
04052   input->rangemax[j + 1] = input->rangemax[j];
04053   input->rangeminabs[j + 1] = input->rangeminabs[j];
04054   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
04055   input->precision[j + 1] = input->precision[j];
04056   input->step[j + 1] = input->step[j];
04057   switch (window_get_algorithm ())
04058     {
04059     case ALGORITHM_SWEEP:
04060       input->nsweeps = (unsigned int *) g_realloc
04061         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
04062       for (j = input->nvariables - 1; j > i; --j)
04063         input->nsweeps[j + 1] = input->nsweeps[j];
04064       input->nsweeps[j + 1] = input->nsweeps[j];
04065       break;
04066     case ALGORITHM_GENETIC:
04067       input->nbits = (unsigned int *) g_realloc
04068         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
04069       for (j = input->nvariables - 1; j > i; --j)
04070         input->nbits[j + 1] = input->nbits[j];
04071       input->nbits[j + 1] = input->nbits[j];
04072     }
04073   ++input->nvariables;
04074   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04075   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
04076   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04077   window_update ();
04078 #if DEBUG
04079   fprintf (stderr, "window_add_variable: end\n");
04080 #endif
04081 }
04082
04087 void
04088 window_label_variable ()
04089 {
04090   unsigned int i;
04091   const char *buffer;
04092 #if DEBUG
04093   fprintf (stderr, "window_label_variable: start\n");
04094 #endif
04095   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04096   buffer = gtk_entry_get_text (window->entry_variable);
04097   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04098   gtk_combo_box_text_remove (window->combo_variable, i);
04099   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
04100   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
04101   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04102 #if DEBUG
04103   fprintf (stderr, "window_label_variable: end\n");
04104 #endif
```

```
04105 }
04106
04111 void
04112 window_precision_variable ()
04113 {
04114   unsigned int i;
04115 #if DEBUG
04116   fprintf (stderr, "window_precision_variable: start\n");
04117 #endif
04118   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04119   input->precision[i]
04120     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
04121   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
04122   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
04123   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
04124   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
04125 #if DEBUG
04126   fprintf (stderr, "window_precision_variable: end\n");
04127 #endif
04128 }
04129
04134 void
04135 window_rangemin_variable ()
04136 {
04137   unsigned int i;
04138 #if DEBUG
04139   fprintf (stderr, "window_rangemin_variable: start\n");
04140 #endif
04141   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04142   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
04143 #if DEBUG
04144   fprintf (stderr, "window_rangemin_variable: end\n");
04145 #endif
04146 }
04147
04152 void
04153 window_rangemax_variable ()
04154 {
04155   unsigned int i;
04156 #if DEBUG
04157   fprintf (stderr, "window_rangemax_variable: start\n");
04158 #endif
04159   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04160   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
04161 #if DEBUG
04162   fprintf (stderr, "window_rangemax_variable: end\n");
04163 #endif
04164 }
04165
04170 void
04171 window_rangeminabs_variable ()
04172 {
04173   unsigned int i;
04174 #if DEBUG
04175   fprintf (stderr, "window_rangeminabs_variable: start\n");
04176 #endif
04177   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04178   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
04179 #if DEBUG
04180   fprintf (stderr, "window_rangeminabs_variable: end\n");
04181 #endif
04182 }
04183
04188 void
04189 window_rangemaxabs_variable ()
04190 {
04191   unsigned int i;
04192 #if DEBUG
04193   fprintf (stderr, "window_rangemaxabs_variable: start\n");
04194 #endif
04195   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04196   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
04197 #if DEBUG
04198   fprintf (stderr, "window_rangemaxabs_variable: end\n");
04199 #endif
04200 }
04201
04206 void
04207 window_step_variable ()
04208 {
04209   unsigned int i;
04210 #if DEBUG
04211   fprintf (stderr, "window_step_variable: start\n");
04212 #endif
04213   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
```

```
04214   input->step[i] = gtk_spin_button_get_value (window->spin_step);
04215 #if DEBUG
04216   fprintf (stderr, "window_step_variable: end\n");
04217 #endif
04218 }
04219
04224 void
04225 window_update_variable ()
04226 {
04227   int i;
04228 #if DEBUG
04229   fprintf (stderr, "window_update_variable: start\n");
04230 #endif
04231   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
04232   if (i < 0)
04233     i = 0;
04234   switch (window_get_algorithm ())
04235     {
04236     case ALGORITHM_SWEEP:
04237       input->nsweeps[i]
04238         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
04239 #if DEBUG
04240       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
04241                input->nsweeps[i]);
04242 #endif
04243       break;
04244     case ALGORITHM_GENETIC:
04245       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
04246 #if DEBUG
04247       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
04248                input->nbits[i]);
04249 #endif
04250     }
04251 #if DEBUG
04252   fprintf (stderr, "window_update_variable: end\n");
04253 #endif
04254 }
04255
04263 int
04264 window_read (char *filename)
04265 {
04266   unsigned int i;
04267   char *buffer;
04268 #if DEBUG
04269   fprintf (stderr, "window_read: start\n");
04270 #endif
04271
04272   // Reading new input file
04273   input_free ();
04274   if (!input_open (filename))
04275     return 0;
04276
04277   // Setting GTK+ widgets data
04278   gtk_entry_set_text (window->entry_result, input->result);
04279   gtk_entry_set_text (window->entry_variables, input->variables);
04280   buffer = g_build_filename (input->directory, input->simulator, NULL);
04281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04282                                  (window->button_simulator), buffer);
04283   g_free (buffer);
04284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04285                                 (size_t) input->evaluator);
04286   if (input->evaluator)
04287     {
04288       buffer = g_build_filename (input->directory, input->evaluator, NULL);
04289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04290                                      (window->button_evaluator), buffer);
04291       g_free (buffer);
04292     }
04293   gtk_toggle_button_set_active
04294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04295   switch (input->algorithm)
04296     {
04297     case ALGORITHM_MONTE_CARLO:
04298       gtk_spin_button_set_value (window->spin_simulations,
04299                                  (gdouble) input->nsimulations);
04300     case ALGORITHM_SWEEP:
04301       gtk_spin_button_set_value (window->spin_iterations,
04302                                  (gdouble) input->niterations);
04303       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
04304       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
04305       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
04306                                     input->nsteps);
04307       if (input->nsteps)
04308         {
```

```
04309           gtk_toggle_button_set_active
04310             (GTK_TOGGLE_BUTTON (window->button_gradient
04311                                 [input->gradient_method]), TRUE);
04312           gtk_spin_button_set_value (window->spin_steps,
04313                                      (gdouble) input->nsteps);
04314           gtk_spin_button_set_value (window->spin_relaxation,
04315                                      (gdouble) input->relaxation);
04316           switch (input->gradient_method)
04317             {
04318             case GRADIENT_METHOD_RANDOM:
04319               gtk_spin_button_set_value (window->spin_estimates,
04320                                          (gdouble) input->nestimates);
04321             }
04322         }
04323       break;
04324     default:
04325       gtk_spin_button_set_value (window->spin_population,
04326                                  (gdouble) input->nsimulations);
04327       gtk_spin_button_set_value (window->spin_generations,
04328                                  (gdouble) input->niterations);
04329       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04330       gtk_spin_button_set_value (window->spin_reproduction,
04331                                  input->reproduction_ratio);
04332       gtk_spin_button_set_value (window->spin_adaptation,
04333                                  input->adaptation_ratio);
04334     }
04335   gtk_toggle_button_set_active
04336     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
04337   gtk_spin_button_set_value (window->spin_p, input->p);
04338   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04339   g_signal_handler_block (window->button_experiment,
04340                           window->id_experiment_name);
04341   gtk_combo_box_text_remove_all (window->combo_experiment);
04342   for (i = 0; i < input->nexperiments; ++i)
04343     gtk_combo_box_text_append_text (window->combo_experiment,
04344                                     input->experiment[i]);
04345   g_signal_handler_unblock
04346     (window->button_experiment, window->id_experiment_name);
04347   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
04348   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04349   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04350   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04351   gtk_combo_box_text_remove_all (window->combo_variable);
04352   for (i = 0; i < input->nvariables; ++i)
04353     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
04354   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04355   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04356   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04357   window_set_variable ();
04358   window_update ();
04359
04360 #if DEBUG
04361   fprintf (stderr, "window_read: end\n");
04362 #endif
04363   return 1;
04364 }
04365
04370 void
04371 window_open ()
04372 {
04373   GtkFileChooserDialog *dlg;
04374   GtkFileFilter *filter;
04375   char *buffer, *directory, *name;
04376
04377 #if DEBUG
04378   fprintf (stderr, "window_open: start\n");
04379 #endif
04380
04381   // Saving a backup of the current input file
04382   directory = g_strdup (input->directory);
04383   name = g_strdup (input->name);
04384
04385   // Opening dialog
04386   dlg = (GtkFileChooserDialog *)
04387     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04388                                  window->window,
04389                                  GTK_FILE_CHOOSER_ACTION_OPEN,
04390                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04391                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
```

```
04392
04393   // Adding XML filter
04394   filter = (GtkFileFilter *) gtk_file_filter_new ();
04395   gtk_file_filter_set_name (filter, "XML");
04396   gtk_file_filter_add_pattern (filter, "*.xml");
04397   gtk_file_filter_add_pattern (filter, "*.XML");
04398   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
04399
04400   // If OK saving
04401   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04402     {
04403
04404       // Traying to open the input file
04405       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04406       if (!window_read (buffer))
04407         {
04408 #if DEBUG
04409           fprintf (stderr, "window_open: error reading input file\n");
04410 #endif
04411           g_free (buffer);
04412
04413           // Reading backup file on error
04414           buffer = g_build_filename (directory, name, NULL);
04415           if (!input_open (buffer))
04416             {
04417
04418               // Closing on backup file reading error
04419 #if DEBUG
04420               fprintf (stderr, "window_read: error reading backup file\n");
04421 #endif
04422               g_free (buffer);
04423               break;
04424             }
04425           g_free (buffer);
04426         }
04427       else
04428         {
04429           g_free (buffer);
04430           break;
04431         }
04432     }
04433
04434   // Freeing and closing
04435   g_free (name);
04436   g_free (directory);
04437   gtk_widget_destroy (GTK_WIDGET (dlg));
04438 #if DEBUG
04439   fprintf (stderr, "window_open: end\n");
04440 #endif
04441 }
04442
04447 void
04448 window_new ()
04449 {
04450   unsigned int i;
04451   char *buffer, *buffer2, buffer3[64];
04452   char *label_algorithm[NALGORITHMS] = {
04453     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04454   };
04455   char *tip_algorithm[NALGORITHMS] = {
04456     gettext ("Monte-Carlo brute force algorithm"),
04457     gettext ("Sweep brute force algorithm"),
04458     gettext ("Genetic algorithm")
04459   };
04460   char *label_gradient[NGRADIENTS] = {
04461     gettext ("_Coordinates descent"), gettext ("_Random")
04462   };
04463   char *tip_gradient[NGRADIENTS] = {
04464     gettext ("Coordinates descent gradient estimate method"),
04465     gettext ("Random gradient estimate method")
04466   };
04467   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
04468   char *tip_norm[NNORMS] = {
04469     gettext ("Euclidean error norm (L2)"),
04470     gettext ("Maximum error norm (L)"),
04471     gettext ("P error norm (Lp)"),
04472     gettext ("Taxicab error norm (L1)")
04473   };
04474
04475 #if DEBUG
04476   fprintf (stderr, "window_new: start\n");
04477 #endif
04478
04479   // Creating the window
04480   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04481
04482   // Finish when closing the window
```

```
04483   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04484
04485   // Setting the window title
04486   gtk_window_set_title (window->window, "MPCOTool");
04487
04488   // Creating the open button
04489   window->button_open = (GtkToolButton *) gtk_tool_button_new
04490     (gtk_image_new_from_icon_name ("document-open",
04491                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04492      gettext ("Open"));
04493   g_signal_connect (window->button_open, "clicked", window_open, NULL);
04494
04495   // Creating the save button
04496   window->button_save = (GtkToolButton *) gtk_tool_button_new
04497     (gtk_image_new_from_icon_name ("document-save",
04498                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04499      gettext ("Save"));
04500   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
04501                     NULL);
04502
04503   // Creating the run button
04504   window->button_run = (GtkToolButton *) gtk_tool_button_new
04505     (gtk_image_new_from_icon_name ("system-run",
04506                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04507      gettext ("Run"));
04508   g_signal_connect (window->button_run, "clicked", window_run, NULL);
04509
04510   // Creating the options button
04511   window->button_options = (GtkToolButton *) gtk_tool_button_new
04512     (gtk_image_new_from_icon_name ("preferences-system",
04513                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04514      gettext ("Options"));
04515   g_signal_connect (window->button_options, "clicked", options_new, NULL);
04516
04517   // Creating the help button
04518   window->button_help = (GtkToolButton *) gtk_tool_button_new
04519     (gtk_image_new_from_icon_name ("help-browser",
04520                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04521      gettext ("Help"));
04522   g_signal_connect (window->button_help, "clicked", window_help, NULL);
04523
04524   // Creating the about button
04525   window->button_about = (GtkToolButton *) gtk_tool_button_new
04526     (gtk_image_new_from_icon_name ("help-about",
04527                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04528      gettext ("About"));
04529   g_signal_connect (window->button_about, "clicked", window_about, NULL);
04530
04531   // Creating the exit button
04532   window->button_exit = (GtkToolButton *) gtk_tool_button_new
04533     (gtk_image_new_from_icon_name ("application-exit",
04534                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04535      gettext ("Exit"));
04536   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
04537
04538   // Creating the buttons bar
04539   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04540   gtk_toolbar_insert
04541     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
04542   gtk_toolbar_insert
04543     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
04544   gtk_toolbar_insert
04545     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
04546   gtk_toolbar_insert
04547     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
04548   gtk_toolbar_insert
04549     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
04550   gtk_toolbar_insert
04551     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
04552   gtk_toolbar_insert
04553     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
04554   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04555
04556   // Creating the simulator program label and entry
04557   window->label_simulator
04558     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04559   window->button_simulator = (GtkFileChooserButton *)
04560     gtk_file_chooser_button_new (gettext ("Simulator program"),
04561                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04562   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04563                                gettext ("Simulator program executable file"));
04564   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
04565
04566   // Creating the evaluator program label and entry
04567   window->check_evaluator = (GtkCheckButton *)
04568     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
```

```
04569   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
04570   window->button_evaluator = (GtkFileChooserButton *)
04571     gtk_file_chooser_button_new (gettext ("Evaluator program"),
04572                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04573   gtk_widget_set_tooltip_text
04574     (GTK_WIDGET (window->button_evaluator),
04575      gettext ("Optional evaluator program executable file"));
04576
04577   // Creating the results files labels and entries
04578   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
04579   window->entry_result = (GtkEntry *) gtk_entry_new ();
04580   gtk_widget_set_tooltip_text
04581     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
04582   window->label_variables
04583     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
04584   window->entry_variables = (GtkEntry *) gtk_entry_new ();
04585   gtk_widget_set_tooltip_text
04586     (GTK_WIDGET (window->entry_variables),
04587      gettext ("All simulated results file"));
04588
04589   // Creating the files grid and attaching widgets
04590   window->grid_files = (GtkGrid *) gtk_grid_new ();
04591   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
04592                    0, 0, 1, 1);
04593   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
04594                    1, 0, 1, 1);
04595   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
04596                    0, 1, 1, 1);
04597   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
04598                    1, 1, 1, 1);
04599   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
04600                    0, 2, 1, 1);
04601   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
04602                    1, 2, 1, 1);
04603   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
04604                    0, 3, 1, 1);
04605   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
04606                    1, 3, 1, 1);
04607
04608   // Creating the algorithm properties
04609   window->label_simulations = (GtkLabel *) gtk_label_new
04610     (gettext ("Simulations number"));
04611   window->spin_simulations
04612     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04613   gtk_widget_set_tooltip_text
04614     (GTK_WIDGET (window->spin_simulations),
04615      gettext ("Number of simulations to perform for each iteration"));
04616   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
04617   window->label_iterations = (GtkLabel *)
04618     gtk_label_new (gettext ("Iterations number"));
04619   window->spin_iterations
04620     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04621   gtk_widget_set_tooltip_text
04622     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04623   g_signal_connect
04624     (window->spin_iterations, "value-changed", window_update, NULL);
04625   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
04626   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04627   window->spin_tolerance
04628     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04629   gtk_widget_set_tooltip_text
04630     (GTK_WIDGET (window->spin_tolerance),
04631      gettext ("Tolerance to set the variable interval on the next iteration"));
04632   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04633   window->spin_bests
04634     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04635   gtk_widget_set_tooltip_text
04636     (GTK_WIDGET (window->spin_bests),
04637      gettext ("Number of best simulations used to set the variable interval "
04638               "on the next iteration"));
04639   window->label_population
04640     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04641   window->spin_population
04642     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04643   gtk_widget_set_tooltip_text
04644     (GTK_WIDGET (window->spin_population),
04645      gettext ("Number of population for the genetic algorithm"));
04646   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
```

```
04647   window->label_generations
04648     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04649   window->spin_generations
04650     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04651   gtk_widget_set_tooltip_text
04652     (GTK_WIDGET (window->spin_generations),
04653      gettext ("Number of generations for the genetic algorithm"));
04654   window->label_mutation
04655     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04656   window->spin_mutation
04657     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04658   gtk_widget_set_tooltip_text
04659     (GTK_WIDGET (window->spin_mutation),
04660      gettext ("Ratio of mutation for the genetic algorithm"));
04661   window->label_reproduction
04662     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04663   window->spin_reproduction
04664     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04665   gtk_widget_set_tooltip_text
04666     (GTK_WIDGET (window->spin_reproduction),
04667      gettext ("Ratio of reproduction for the genetic algorithm"));
04668   window->label_adaptation
04669     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04670   window->spin_adaptation
04671     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04672   gtk_widget_set_tooltip_text
04673     (GTK_WIDGET (window->spin_adaptation),
04674      gettext ("Ratio of adaptation for the genetic algorithm"));
04675
04676   // Creating the gradient based method properties
04677   window->check_gradient = (GtkCheckButton *)
04678     gtk_check_button_new_with_mnemonic (gettext ("_Gradient based method"));
04679   g_signal_connect (window->check_gradient, "clicked",
      window_update, NULL);
04680   window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04681   window->button_gradient[0] = (GtkRadioButton *)
04682     gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04683   gtk_grid_attach (window->grid_gradient,
04684                    GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04685   g_signal_connect (window->button_gradient[0], "clicked",
      window_update, NULL);
04686   for (i = 0; ++i < NGRADIENTS;)
04687     {
04688       window->button_gradient[i] = (GtkRadioButton *)
04689         gtk_radio_button_new_with_mnemonic
04690         (gtk_radio_button_get_group (window->button_gradient[0]),
04691          label_gradient[i]);
04692       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04693                                    tip_gradient[i]);
04694       gtk_grid_attach (window->grid_gradient,
04695                        GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04696       g_signal_connect (window->button_gradient[i], "clicked",
04697                         window_update, NULL);
04698     }
04699   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04700   window->spin_steps = (GtkSpinButton *)
04701     gtk_spin_button_new_with_range (1., 1.e12, 1.);
04702   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
04703   window->label_estimates
04704     = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04705   window->spin_estimates = (GtkSpinButton *)
04706     gtk_spin_button_new_with_range (1., 1.e3, 1.);
04707   window->label_relaxation
04708     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04709   window->spin_relaxation = (GtkSpinButton *)
04710     gtk_spin_button_new_with_range (0., 2., 0.001);
04711   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_steps),
04712                    0, NGRADIENTS, 1, 1);
04713   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_steps),
04714                    1, NGRADIENTS, 1, 1);
04715   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_estimates),
04716                    0, NGRADIENTS + 1, 1, 1);
04717   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_estimates),
04718                    1, NGRADIENTS + 1, 1, 1);
04719   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_relaxation),
04720                    0, NGRADIENTS + 2, 1, 1);
04721   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_relaxation),
04722                    1, NGRADIENTS + 2, 1, 1);
04723
04724   // Creating the array of algorithms
04725   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
```

```
04726   window->button_algorithm[0] = (GtkRadioButton *)
04727     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04728   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
04729                                tip_algorithm[0]);
04730   gtk_grid_attach (window->grid_algorithm,
04731                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
04732   g_signal_connect (window->button_algorithm[0], "clicked",
04733                     window_set_algorithm, NULL);
04734   for (i = 0; ++i < NALGORITHMS;)
04735     {
04736       window->button_algorithm[i] = (GtkRadioButton *)
04737         gtk_radio_button_new_with_mnemonic
04738         (gtk_radio_button_get_group (window->button_algorithm[0]),
04739          label_algorithm[i]);
04740       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
04741                                    tip_algorithm[i]);
04742       gtk_grid_attach (window->grid_algorithm,
04743                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
04744       g_signal_connect (window->button_algorithm[i], "clicked",
04745                         window_set_algorithm, NULL);
04746     }
04747   gtk_grid_attach (window->grid_algorithm,
04748                    GTK_WIDGET (window->label_simulations), 0,
04749                    NALGORITHMS, 1, 1);
04750   gtk_grid_attach (window->grid_algorithm,
04751                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
04752   gtk_grid_attach (window->grid_algorithm,
04753                    GTK_WIDGET (window->label_iterations), 0,
04754                    NALGORITHMS + 1, 1, 1);
04755   gtk_grid_attach (window->grid_algorithm,
04756                    GTK_WIDGET (window->spin_iterations), 1,
04757                    NALGORITHMS + 1, 1, 1);
04758   gtk_grid_attach (window->grid_algorithm,
04759                    GTK_WIDGET (window->label_tolerance), 0,
04760                    NALGORITHMS + 2, 1, 1);
04761   gtk_grid_attach (window->grid_algorithm,
04762                    GTK_WIDGET (window->spin_tolerance), 1,
04763                    NALGORITHMS + 2, 1, 1);
04764   gtk_grid_attach (window->grid_algorithm,
04765                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
04766   gtk_grid_attach (window->grid_algorithm,
04767                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
04768   gtk_grid_attach (window->grid_algorithm,
04769                    GTK_WIDGET (window->label_population), 0,
04770                    NALGORITHMS + 4, 1, 1);
04771   gtk_grid_attach (window->grid_algorithm,
04772                    GTK_WIDGET (window->spin_population), 1,
04773                    NALGORITHMS + 4, 1, 1);
04774   gtk_grid_attach (window->grid_algorithm,
04775                    GTK_WIDGET (window->label_generations), 0,
04776                    NALGORITHMS + 5, 1, 1);
04777   gtk_grid_attach (window->grid_algorithm,
04778                    GTK_WIDGET (window->spin_generations), 1,
04779                    NALGORITHMS + 5, 1, 1);
04780   gtk_grid_attach (window->grid_algorithm,
04781                    GTK_WIDGET (window->label_mutation), 0,
04782                    NALGORITHMS + 6, 1, 1);
04783   gtk_grid_attach (window->grid_algorithm,
04784                    GTK_WIDGET (window->spin_mutation), 1,
04785                    NALGORITHMS + 6, 1, 1);
04786   gtk_grid_attach (window->grid_algorithm,
04787                    GTK_WIDGET (window->label_reproduction), 0,
04788                    NALGORITHMS + 7, 1, 1);
04789   gtk_grid_attach (window->grid_algorithm,
04790                    GTK_WIDGET (window->spin_reproduction), 1,
04791                    NALGORITHMS + 7, 1, 1);
04792   gtk_grid_attach (window->grid_algorithm,
04793                    GTK_WIDGET (window->label_adaptation), 0,
04794                    NALGORITHMS + 8, 1, 1);
04795   gtk_grid_attach (window->grid_algorithm,
04796                    GTK_WIDGET (window->spin_adaptation), 1,
04797                    NALGORITHMS + 8, 1, 1);
04798   gtk_grid_attach (window->grid_algorithm,
04799                    GTK_WIDGET (window->check_gradient), 0,
04800                    NALGORITHMS + 9, 2, 1);
04801   gtk_grid_attach (window->grid_algorithm,
04802                    GTK_WIDGET (window->grid_gradient), 0,
04803                    NALGORITHMS + 10, 2, 1);
04804   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
04805   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04806                      GTK_WIDGET (window->grid_algorithm));
04807
04808   // Creating the variable widgets
04809   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04810   gtk_widget_set_tooltip_text
04811     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04812   window->id_variable = g_signal_connect
```

```
04813     (window->combo_variable, "changed", window_set_variable, NULL);
04814   window->button_add_variable
04815     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04816                                                    GTK_ICON_SIZE_BUTTON);
04817   g_signal_connect
04818     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
04819   gtk_widget_set_tooltip_text
04820     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
04821   window->button_remove_variable
04822     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04823                                                    GTK_ICON_SIZE_BUTTON);
04824   g_signal_connect
04825     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
04826   gtk_widget_set_tooltip_text
04827     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04828   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04829   window->entry_variable = (GtkEntry *) gtk_entry_new ();
04830   gtk_widget_set_tooltip_text
04831     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04832   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
04833   window->id_variable_label = g_signal_connect
04834     (window->entry_variable, "changed", window_label_variable, NULL);
04835   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04836   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04837     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04838   gtk_widget_set_tooltip_text
04839     (GTK_WIDGET (window->spin_min),
04840      gettext ("Minimum initial value of the variable"));
04841   window->scrolled_min
04842     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04843   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04844                      GTK_WIDGET (window->spin_min));
04845   g_signal_connect (window->spin_min, "value-changed",
04846                     window_rangemin_variable, NULL);
04847   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04848   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04849     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04850   gtk_widget_set_tooltip_text
04851     (GTK_WIDGET (window->spin_max),
04852      gettext ("Maximum initial value of the variable"));
04853   window->scrolled_max
04854     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04855   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04856                      GTK_WIDGET (window->spin_max));
04857   g_signal_connect (window->spin_max, "value-changed",
04858                     window_rangemax_variable, NULL);
04859   window->check_minabs = (GtkCheckButton *)
04860     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
04861   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04862   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04863     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04864   gtk_widget_set_tooltip_text
04865     (GTK_WIDGET (window->spin_minabs),
04866      gettext ("Minimum allowed value of the variable"));
04867   window->scrolled_minabs
04868     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04869   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04870                      GTK_WIDGET (window->spin_minabs));
04871   g_signal_connect (window->spin_minabs, "value-changed",
04872                     window_rangeminabs_variable, NULL);
04873   window->check_maxabs = (GtkCheckButton *)
04874     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
04875   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04876   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04877     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04878   gtk_widget_set_tooltip_text
04879     (GTK_WIDGET (window->spin_maxabs),
04880      gettext ("Maximum allowed value of the variable"));
04881   window->scrolled_maxabs
04882     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04883   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04884                      GTK_WIDGET (window->spin_maxabs));
04885   g_signal_connect (window->spin_maxabs, "value-changed",
04886                     window_rangemaxabs_variable, NULL);
04887   window->label_precision
04888     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04889   window->spin_precision = (GtkSpinButton *)
04890     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04891   gtk_widget_set_tooltip_text
04892     (GTK_WIDGET (window->spin_precision),
04893      gettext ("Number of precision floating point digits\n"
04894               "0 is for integer numbers"));
04895   g_signal_connect (window->spin_precision, "value-changed",
04896                     window_precision_variable, NULL);
04897   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
```

```
04898   window->spin_sweeps
04899     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04900   gtk_widget_set_tooltip_text
04901     (GTK_WIDGET (window->spin_sweeps),
04902      gettext ("Number of steps sweeping the variable"));
04903   g_signal_connect
04904     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
04905   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
04906   window->spin_bits
04907     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
04908   gtk_widget_set_tooltip_text
04909     (GTK_WIDGET (window->spin_bits),
04910      gettext ("Number of bits to encode the variable"));
04911   g_signal_connect
04912     (window->spin_bits, "value-changed", window_update_variable, NULL);
04913   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
04914   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
04915     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04916   gtk_widget_set_tooltip_text
04917     (GTK_WIDGET (window->spin_step),
04918      gettext ("Initial step size for the gradient based method"));
04919   window->scrolled_step
04920     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04921   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
04922                      GTK_WIDGET (window->spin_step));
04923   g_signal_connect
04924     (window->spin_step, "value-changed", window_step_variable, NULL);
04925   window->grid_variable = (GtkGrid *) gtk_grid_new ();
04926   gtk_grid_attach (window->grid_variable,
04927                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
04928   gtk_grid_attach (window->grid_variable,
04929                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
04930   gtk_grid_attach (window->grid_variable,
04931                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
04932   gtk_grid_attach (window->grid_variable,
04933                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
04934   gtk_grid_attach (window->grid_variable,
04935                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
04936   gtk_grid_attach (window->grid_variable,
04937                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
04938   gtk_grid_attach (window->grid_variable,
04939                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04940   gtk_grid_attach (window->grid_variable,
04941                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04942   gtk_grid_attach (window->grid_variable,
04943                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04944   gtk_grid_attach (window->grid_variable,
04945                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
04946   gtk_grid_attach (window->grid_variable,
04947                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
04948   gtk_grid_attach (window->grid_variable,
04949                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04950   gtk_grid_attach (window->grid_variable,
04951                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
04952   gtk_grid_attach (window->grid_variable,
04953                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
04954   gtk_grid_attach (window->grid_variable,
04955                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
04956   gtk_grid_attach (window->grid_variable,
04957                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04958   gtk_grid_attach (window->grid_variable,
04959                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04960   gtk_grid_attach (window->grid_variable,
04961                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04962   gtk_grid_attach (window->grid_variable,
04963                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04964   gtk_grid_attach (window->grid_variable,
04965                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
04966   gtk_grid_attach (window->grid_variable,
04967                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
04968   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
04969   gtk_container_add (GTK_CONTAINER (window->frame_variable),
04970                      GTK_WIDGET (window->grid_variable));
04971
04972   // Creating the experiment widgets
04973   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
04974   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
04975                                gettext ("Experiment selector"));
04976   window->id_experiment = g_signal_connect
04977     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
04978   window->button_add_experiment
04979     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04980                                                    GTK_ICON_SIZE_BUTTON);
04981   g_signal_connect
04982     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
```

```
04983   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04984                                gettext ("Add experiment"));
04985   window->button_remove_experiment
04986     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04987                                                    GTK_ICON_SIZE_BUTTON);
04988   g_signal_connect (window->button_remove_experiment, "clicked",
04989                     window_remove_experiment, NULL);
04990   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04991                                gettext ("Remove experiment"));
04992   window->label_experiment
04993     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04994   window->button_experiment = (GtkFileChooserButton *)
04995     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04996                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04997   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04998                                gettext ("Experimental data file"));
04999   window->id_experiment_name
05000     = g_signal_connect (window->button_experiment, "selection-changed",
05001                         window_name_experiment, NULL);
05002   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
05003   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
05004   window->spin_weight
05005     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
05006   gtk_widget_set_tooltip_text
05007     (GTK_WIDGET (window->spin_weight),
05008      gettext ("Weight factor to build the objective function"));
05009   g_signal_connect
05010     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
05011   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
05012   gtk_grid_attach (window->grid_experiment,
05013                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
05014   gtk_grid_attach (window->grid_experiment,
05015                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
05016   gtk_grid_attach (window->grid_experiment,
05017                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
05018   gtk_grid_attach (window->grid_experiment,
05019                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
05020   gtk_grid_attach (window->grid_experiment,
05021                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
05022   gtk_grid_attach (window->grid_experiment,
05023                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
05024   gtk_grid_attach (window->grid_experiment,
05025                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
05026   for (i = 0; i < MAX_NINPUTS; ++i)
05027     {
05028       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
05029       window->check_template[i] = (GtkCheckButton *)
05030         gtk_check_button_new_with_label (buffer3);
05031       window->id_template[i]
05032         = g_signal_connect (window->check_template[i], "toggled",
05033                             window_inputs_experiment, NULL);
05034       gtk_grid_attach (window->grid_experiment,
05035                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
05036       window->button_template[i] = (GtkFileChooserButton *)
05037         gtk_file_chooser_button_new (gettext ("Input template"),
05038                                      GTK_FILE_CHOOSER_ACTION_OPEN);
05039       gtk_widget_set_tooltip_text
05040         (GTK_WIDGET (window->button_template[i]),
05041          gettext ("Experimental input template file"));
05042       window->id_input[i]
05043         = g_signal_connect_swapped (window->button_template[i],
05044                                     "selection-changed",
05045                                     (void (*)) window_template_experiment,
05046                                     (void *) (size_t) i);
05047       gtk_grid_attach (window->grid_experiment,
05048                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
05049     }
05050   window->frame_experiment
05051     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
05052   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
05053                      GTK_WIDGET (window->grid_experiment));
05054
05055   // Creating the error norm widgets
05056   window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
05057   window->grid_norm = (GtkGrid *) gtk_grid_new ();
05058   gtk_container_add (GTK_CONTAINER (window->frame_norm),
05059                      GTK_WIDGET (window->grid_norm));
05060   window->button_norm[0] = (GtkRadioButton *)
05061     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
05062   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
05063                                tip_norm[0]);
05064   gtk_grid_attach (window->grid_norm,
05065                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
05066   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
05067   for (i = 0; ++i < NNORMS;)
05068     {
```

```
05069       window->button_norm[i] = (GtkRadioButton *)
05070         gtk_radio_button_new_with_mnemonic
05071         (gtk_radio_button_get_group (window->button_norm[0]),
05072          label_norm[i]);
05073       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
05074                                    tip_norm[i]);
05075       gtk_grid_attach (window->grid_norm,
05076                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
05077       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
05078     }
05079   window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
05080   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p),
05081                    1, 0, 1, 2);
05082   window->spin_p = (GtkSpinButton *)
05083     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
05084   gtk_widget_set_tooltip_text
05085     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
05086   window->scrolled_p
05087     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
05088   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
05089                      GTK_WIDGET (window->spin_p));
05090   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
05091   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
05092   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
05093                    2, 0, 1, 2);
05094
05095   // Creating the grid and attaching the widgets to the grid
05096   window->grid = (GtkGrid *) gtk_grid_new ();
05097   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
05098   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
05099   gtk_grid_attach (window->grid,
05100                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
05101   gtk_grid_attach (window->grid,
05102                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
05103   gtk_grid_attach (window->grid,
05104                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
05105   gtk_grid_attach (window->grid,
05106                    GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
05107   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
05108
05109   // Setting the window logo
05110   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
05111   gtk_window_set_icon (window->window, window->logo);
05112
05113   // Showing the window
05114   gtk_widget_show_all (GTK_WIDGET (window->window));
05115
05116   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
05117 #if GTK_MINOR_VERSION >= 16
05118   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
05119   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
05120   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
05121   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
05122   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
05123   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
05124 #endif
05125
05126   // Reading initial example
05127   input_new ();
05128   buffer2 = g_get_current_dir ();
05129   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
05130   g_free (buffer2);
05131   window_read (buffer);
05132   g_free (buffer);
05133
05134 #if DEBUG
05135   fprintf (stderr, "window_new: start\n");
05136 #endif
05137 }
05138
05139 #endif
05140
05146 int
05147 cores_number ()
05148 {
05149 #ifdef G_OS_WIN32
05150   SYSTEM_INFO sysinfo;
05151   GetSystemInfo (&sysinfo);
05152   return sysinfo.dwNumberOfProcessors;
05153 #else
05154   return (int) sysconf (_SC_NPROCESSORS_ONLN);
05155 #endif
05156 }
05157
05167 int
```

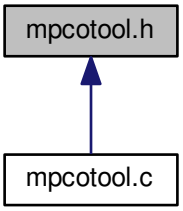
```
05168 main (int argn, char **argc)
05169 {
05170 #if HAVE_GTK
05171   char *buffer;
05172 #endif
05173
05174   // Starting pseudo-random numbers generator
05175   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
05176   calibrate->seed = DEFAULT_RANDOM_SEED;
05177
05178   // Allowing spaces in the XML data file
05179   xmlKeepBlanksDefault (0);
05180
05181   // Starting MPI
05182 #if HAVE_MPI
05183   MPI_Init (&argn, &argc);
05184   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
05185   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
05186   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
05187 #else
05188   ntasks = 1;
05189 #endif
05190
05191 #if HAVE_GTK
05192
05193   // Getting threads number
05194   nthreads_gradient = nthreads = cores_number ();
05195
05196   // Setting local language and international floating point numbers notation
05197   setlocale (LC_ALL, "");
05198   setlocale (LC_NUMERIC, "C");
05199   window->application_directory = g_get_current_dir ();
05200   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
05201   bindtextdomain (PROGRAM_INTERFACE, buffer);
05202   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
05203   textdomain (PROGRAM_INTERFACE);
05204
05205   // Initing GTK+
05206   gtk_disable_setlocale ();
05207   gtk_init (&argn, &argc);
05208
05209   // Opening the main window
05210   window_new ();
05211   gtk_main ();
05212
05213   // Freeing memory
05214   input_free ();
05215   g_free (buffer);
05216   gtk_widget_destroy (GTK_WIDGET (window->window));
05217   g_free (window->application_directory);
05218
05219 #else
05220
05221   // Checking syntax
05222   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
05223     {
05224       printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
05225       return 1;
05226     }
05227
05228   // Getting threads number
05229   if (argn == 2)
05230     nthreads_gradient = nthreads = cores_number ();
05231   else
05232     {
05233       nthreads_gradient = nthreads = atoi (argc[2]);
05234       if (!nthreads)
05235         {
05236           printf ("Bad threads number\n");
05237           return 2;
05238         }
05239     }
05240   printf ("nthreads=%u\n", nthreads);
05241
05242   // Making calibration
05243   if (input_open (argc[argn - 1]))
05244     calibrate_open ();
05245
05246   // Freeing memory
05247   calibrate_free ();
05248
05249 #endif
05250
05251   // Closing MPI
05252 #if HAVE_MPI
05253   MPI_Finalize ();
```

```
05254 #endif
05255
05256   // Freeing memory
05257   gsl_rng_free (calibrate->rng);
05258
05259   // Closing
05260   return 0;
05261 }
```

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.
- enum ErrorNorm { ERROR_NORM_EUCLIDIAN = 0, ERROR_NORM_MAXIMUM = 1, ERROR_NORM_P = 2, ERROR_NORM_TAXICAB = 3 }

  Enum to define the error norm.

Functions

- void 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

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 ErrorNorm

Enum to define the error norm.

Enumerator

ERROR_NORM_EUCLIDIAN Euclidian norm: $\sqrt{\sum_i (w_i\, x_i)^2}$.

ERROR_NORM_MAXIMUM Maximum norm: $\max_i |w_i\, x_i|$.

ERROR_NORM_P P-norm $\sqrt[p]{\sum_i |w_i\, x_i|^p}$.

ERROR_NORM_TAXICAB Taxicab norm $\sum_i |w_i\, x_i|$.

Definition at line 64 of file mpcotool.h.

```
00065 {
00066   ERROR_NORM_EUCLIDIAN = 0,
00068   ERROR_NORM_MAXIMUM = 1,
00070   ERROR_NORM_P = 2,
00072   ERROR_NORM_TAXICAB = 3
00074 };
```

5.7.2.3 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 1610 of file mpcotool.c.

```
01611 {
01612   unsigned int i, j;
01613   double e;
01614 #if DEBUG
01615   fprintf (stderr, "calibrate_best: start\n");
01616   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01617            calibrate->nsaveds, calibrate->nbest);
01618 #endif
01619   if (calibrate->nsaveds < calibrate->nbest
01620       || value < calibrate->error_best[calibrate->nsaveds - 1])
01621     {
01622       if (calibrate->nsaveds < calibrate->nbest)
01623         ++calibrate->nsaveds;
01624       calibrate->error_best[calibrate->nsaveds - 1] = value;
01625       calibrate->simulation_best[calibrate->
```

```
      nsaveds - 1] = simulation;
01626       for (i = calibrate->nsaveds; --i;)
01627         {
01628           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01629             {
01630               j = calibrate->simulation_best[i];
01631               e = calibrate->error_best[i];
01632               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01633               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01634               calibrate->simulation_best[i - 1] = j;
01635               calibrate->error_best[i - 1] = e;
01636             }
01637           else
01638             break;
01639         }
01640     }
01641 #if DEBUG
01642   fprintf (stderr, "calibrate_best: end\n");
01643 #endif
01644 }
```

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 1919 of file mpcotool.c.

```
01920 {
01921 #if DEBUG
01922   fprintf (stderr, "calibrate_best_gradient: start\n");
01923   fprintf (stderr,
01924            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01925            simulation, value, calibrate->error_best[0]);
01926 #endif
01927   if (value < calibrate->error_best[0])
01928     {
01929       calibrate->error_best[0] = value;
01930       calibrate->simulation_best[0] = simulation;
01931 #if DEBUG
01932       fprintf (stderr,
01933                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01934                simulation, value);
01935 #endif
01936     }
01937 #if DEBUG
01938   fprintf (stderr, "calibrate_best_gradient: end\n");
01939 #endif
01940 }
```

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 2218 of file mpcotool.c.

```
02219 {
02220   unsigned int j;
02221   double objective;
02222   char buffer[64];
02223 #if DEBUG
02224   fprintf (stderr, "calibrate_genetic_objective: start\n");
02225 #endif
02226   for (j = 0; j < calibrate->nvariables; ++j)
02227     {
02228       calibrate->value[entity->id * calibrate->nvariables + j]
02229         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02230     }
02231   objective = calibrate_norm (entity->id);
02232   g_mutex_lock (mutex);
02233   for (j = 0; j < calibrate->nvariables; ++j)
02234     {
02235       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02236       fprintf (calibrate->file_variables, buffer,
02237                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
02238     }
02239   fprintf (calibrate->file_variables, "%.14le\n", objective);
02240   g_mutex_unlock (mutex);
02241 #if DEBUG
02242   fprintf (stderr, "calibrate_genetic_objective: end\n");
02243 #endif
02244   return objective;
02245 }
```

5.7.3.4 void* calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

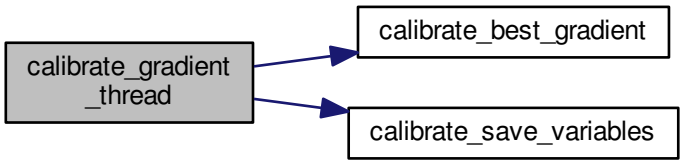
Returns

NULL

Definition at line 1982 of file mpcotool.c.

```
01983 {
01984   unsigned int i, thread;
01985   double e;
01986 #if DEBUG
01987   fprintf (stderr, "calibrate_gradient_thread: start\n");
01988 #endif
01989   thread = data->thread;
01990 #if DEBUG
01991   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01992            thread,
01993            calibrate->thread_gradient[thread],
01994            calibrate->thread_gradient[thread + 1]);
01995 #endif
01996   for (i = calibrate->thread_gradient[thread];
01997        i < calibrate->thread_gradient[thread + 1]; ++i)
01998     {
01999       e = calibrate_norm (i);
02000       g_mutex_lock (mutex);
02001       calibrate_best_gradient (i, e);
02002       calibrate_save_variables (i, e);
02003       g_mutex_unlock (mutex);
02004 #if DEBUG
02005       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
02006 #endif
02007     }
02008 #if DEBUG
02009   fprintf (stderr, "calibrate_gradient_thread: end\n");
02010 #endif
02011   g_thread_exit (NULL);
02012   return NULL;
02013 }
```

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 1252 of file mpcotool.c.

```
01253 {
01254   unsigned int i;
01255   char buffer[32], value[32], *buffer2, *buffer3, *content;
01256   FILE *file;
01257   gsize length;
01258   GRegex *regex;
01259
01260 #if DEBUG
01261   fprintf (stderr, "calibrate_input: start\n");
01262 #endif
01263
01264   // Checking the file
01265   if (!template)
01266     goto calibrate_input_end;
01267
01268   // Opening template
01269   content = g_mapped_file_get_contents (template);
01270   length = g_mapped_file_get_length (template);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01273            content);
01274 #endif
01275   file = g_fopen (input, "w");
01276
01277   // Parsing template
01278   for (i = 0; i < calibrate->nvariables; ++i)
01279     {
01280 #if DEBUG
01281       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01282 #endif
01283       snprintf (buffer, 32, "@variable%u@", i + 1);
01284       regex = g_regex_new (buffer, 0, 0, NULL);
01285       if (i == 0)
01286         {
01287           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01288                                              calibrate->label[i], 0, NULL);
01289 #if DEBUG
01290           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01291 #endif
01292         }
01293       else
01294         {
01295           length = strlen (buffer3);
01296           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01297                                              calibrate->label[i], 0, NULL);
```

```
01298           g_free (buffer3);
01299         }
01300       g_regex_unref (regex);
01301       length = strlen (buffer2);
01302       snprintf (buffer, 32, "@value%u@", i + 1);
01303       regex = g_regex_new (buffer, 0, 0, NULL);
01304       snprintf (value, 32, format[calibrate->precision[i]],
01305                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01306 
01307 #if DEBUG
01308       fprintf (stderr, "calibrate_input: value=%s\n", value);
01309 #endif
01310       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01311                                          0, NULL);
01312       g_free (buffer2);
01313       g_regex_unref (regex);
01314     }
01315 
01316   // Saving input file
01317   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01318   g_free (buffer3);
01319   fclose (file);
01320 
01321 calibrate_input_end:
01322 #if DEBUG
01323   fprintf (stderr, "calibrate_input: end\n");
01324 #endif
01325   return;
01326 }
```

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 1724 of file mpcotool.c.

```
01726 {
01727   unsigned int i, j, k, s[calibrate->nbest];
01728   double e[calibrate->nbest];
01729 #if DEBUG
01730   fprintf (stderr, "calibrate_merge: start\n");
01731 #endif
01732   i = j = k = 0;
01733   do
01734     {
01735       if (i == calibrate->nsaveds)
01736         {
01737           s[k] = simulation_best[j];
01738           e[k] = error_best[j];
01739           ++j;
01740           ++k;
01741           if (j == nsaveds)
01742             break;
01743         }
01744       else if (j == nsaveds)
01745         {
01746           s[k] = calibrate->simulation_best[i];
01747           e[k] = calibrate->error_best[i];
01748           ++i;
01749           ++k;
01750           if (i == calibrate->nsaveds)
01751             break;
01752         }
01753       else if (calibrate->error_best[i] > error_best[j])
01754         {
01755           s[k] = simulation_best[j];
01756           e[k] = error_best[j];
01757           ++j;
01758           ++k;
01759         }
01760       else
01761         {
01762           s[k] = calibrate->simulation_best[i];
```

```
01763           e[k] = calibrate->error_best[i];
01764           ++i;
01765           ++k;
01766         }
01767     }
01768   while (k < calibrate->nbest);
01769   calibrate->nsaveds = k;
01770   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01771   memcpy (calibrate->error_best, e, k * sizeof (double));
01772 #if DEBUG
01773   fprintf (stderr, "calibrate_merge: end\n");
01774 #endif
01775 }
```

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 1339 of file mpcotool.c.

```
01340 {
01341   unsigned int i;
01342   double e;
01343   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01344     *buffer3, *buffer4;
01345   FILE *file_result;
01346
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_parse: start\n");
01349   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01350            experiment);
01351 #endif
01352
01353   // Opening input files
01354   for (i = 0; i < calibrate->ninputs; ++i)
01355     {
01356       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01357 #if DEBUG
01358       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01359 #endif
01360       calibrate_input (simulation, &input[i][0],
01361                        calibrate->file[i][experiment]);
01362     }
01363   for (; i < MAX_NINPUTS; ++i)
01364     strcpy (&input[i][0], "");
01365 #if DEBUG
01366   fprintf (stderr, "calibrate_parse: parsing end\n");
01367 #endif
01368
01369   // Performing the simulation
01370   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01371   buffer2 = g_path_get_dirname (calibrate->simulator);
01372   buffer3 = g_path_get_basename (calibrate->simulator);
01373   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01374   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01375             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01376             input[6], input[7], output);
01377   g_free (buffer4);
01378   g_free (buffer3);
01379   g_free (buffer2);
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01382 #endif
01383   system (buffer);
01384
01385   // Checking the objective value function
01386   if (calibrate->evaluator)
01387     {
01388       snprintf (result, 32, "result-%u-%u", simulation, experiment);
```

```
01389       buffer2 = g_path_get_dirname (calibrate->evaluator);
01390       buffer3 = g_path_get_basename (calibrate->evaluator);
01391       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01392       snprintf (buffer, 512, "\"%s\" %s %s %s",
01393                 buffer4, output, calibrate->experiment[experiment], result);
01394       g_free (buffer4);
01395       g_free (buffer3);
01396       g_free (buffer2);
01397 #if DEBUG
01398       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01399 #endif
01400       system (buffer);
01401       file_result = g_fopen (result, "r");
01402       e = atof (fgets (buffer, 512, file_result));
01403       fclose (file_result);
01404     }
01405   else
01406     {
01407       strcpy (result, "");
01408       file_result = g_fopen (output, "r");
01409       e = atof (fgets (buffer, 512, file_result));
01410       fclose (file_result);
01411     }
01412
01413   // Removing files
01414 #if !DEBUG
01415   for (i = 0; i < calibrate->ninputs; ++i)
01416     {
01417       if (calibrate->file[i][0])
01418         {
01419           snprintf (buffer, 512, RM " %s", &input[i][0]);
01420           system (buffer);
01421         }
01422     }
01423   snprintf (buffer, 512, RM " %s %s", output, result);
01424   system (buffer);
01425 #endif
01426
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_parse: end\n");
01429 #endif
01430
01431   // Returning the objective function
01432   return e * calibrate->weight[experiment];
01433 }
```

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 1582 of file mpcotool.c.

```
01583 {
01584   unsigned int i;
01585   char buffer[64];
01586 #if DEBUG
```

```
01587   fprintf (stderr, "calibrate_save_variables: start\n");
01588 #endif
01589   for (i = 0; i < calibrate->nvariables; ++i)
01590     {
01591       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01592       fprintf (calibrate->file_variables, buffer,
01593                calibrate->value[simulation * calibrate->
      nvariables + i]);
01594     }
01595   fprintf (calibrate->file_variables, "%.14le\n", error);
01596 #if DEBUG
01597   fprintf (stderr, "calibrate_save_variables: end\n");
01598 #endif
01599 }
```

5.7.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

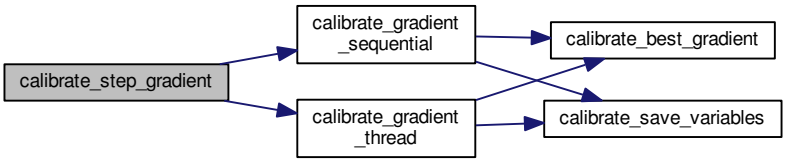
simulation	Simulation number.

Definition at line 2082 of file mpcotool.c.

```
02083 {
02084   GThread *thread[nthreads_gradient];
02085   ParallelData data[nthreads_gradient];
02086   unsigned int i, j, k, b;
02087 #if DEBUG
02088   fprintf (stderr, "calibrate_step_gradient: start\n");
02089 #endif
02090   for (i = 0; i < calibrate->nestimates; ++i)
02091     {
02092       k = (simulation + i) * calibrate->nvariables;
02093       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
02094 #if DEBUG
02095       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
02096                simulation + i, calibrate->simulation_best[0]);
02097 #endif
02098       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
02099         {
02100 #if DEBUG
02101           fprintf (stderr,
02102                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
02103                    i, j, calibrate->value[b]);
02104 #endif
02105           calibrate->value[k]
02106             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
02107           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
02108                                             calibrate->rangeminabs[j]),
02109                                       calibrate->rangemaxabs[j]);
02110 #if DEBUG
02111           fprintf (stderr,
02112                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
02113                    i, j, calibrate->value[k]);
02114 #endif
02115         }
02116     }
02117   if (nthreads_gradient == 1)
02118     calibrate_gradient_sequential (simulation);
02119   else
02120     {
02121       for (i = 0; i <= nthreads_gradient; ++i)
02122         {
02123           calibrate->thread_gradient[i]
02124             = simulation + calibrate->nstart_gradient
02125             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
02126             / nthreads_gradient;
02127 #if DEBUG
02128           fprintf (stderr,
02129                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
02130                    i, calibrate->thread_gradient[i]);
02131 #endif
02132         }
02133       for (i = 0; i < nthreads_gradient; ++i)
02134         {
```

```
02135           data[i].thread = i;
02136           thread[i] = g_thread_new
02137             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
02138         }
02139       for (i = 0; i < nthreads_gradient; ++i)
02140         g_thread_join (thread[i]);
02141     }
02142 #if DEBUG
02143   fprintf (stderr, "calibrate_step_gradient: end\n");
02144 #endif
02145 }
```

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.

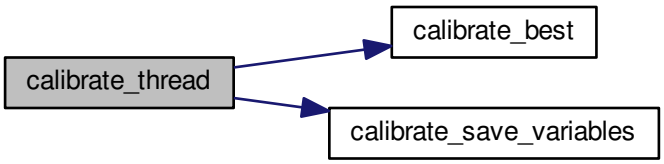
Returns

NULL

Definition at line 1682 of file mpcotool.c.

```
01683 {
01684   unsigned int i, thread;
01685   double e;
01686 #if DEBUG
01687   fprintf (stderr, "calibrate_thread: start\n");
01688 #endif
01689   thread = data->thread;
01690 #if DEBUG
01691   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01692            calibrate->thread[thread], calibrate->thread[thread + 1]);
01693 #endif
01694   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01695     {
01696       e = calibrate_norm (i);
01697       g_mutex_lock (mutex);
01698       calibrate_best (i, e);
01699       calibrate_save_variables (i, e);
01700       g_mutex_unlock (mutex);
01701 #if DEBUG
01702       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01703 #endif
01704     }
01705 #if DEBUG
01706   fprintf (stderr, "calibrate_thread: end\n");
01707 #endif
01708   g_thread_exit (NULL);
01709   return NULL;
01710 }
```

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 574 of file mpcotool.c.

```
00575 {
00576   char buffer2[64];
00577   char *buffert[MAX_NINPUTS] =
00578     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00579   xmlDoc *doc;
00580   xmlNode *node, *child;
00581   xmlChar *buffer;
00582   char *msg;
00583   int error_code;
00584   unsigned int i;
00585
00586 #if DEBUG
00587   fprintf (stderr, "input_open: start\n");
00588 #endif
00589
00590   // Resetting input data
00591   buffer = NULL;
00592   input_new ();
00593
00594   // Parsing the input file
00595 #if DEBUG
00596   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00597 #endif
00598   doc = xmlParseFile (filename);
00599   if (!doc)
00600     {
00601       msg = gettext ("Unable to parse the input file");
00602       goto exit_on_error;
00603     }
00604
00605   // Getting the root node
00606 #if DEBUG
00607   fprintf (stderr, "input_open: getting the root node\n");
00608 #endif
00609   node = xmlDocGetRootElement (doc);
00610   if (xmlStrcmp (node->name, XML_CALIBRATE))
00611     {
00612       msg = gettext ("Bad root XML node");
00613       goto exit_on_error;
00614     }
00615
```

```
00616   // Getting results file names
00617   input->result = (char *) xmlGetProp (node, XML_RESULT);
00618   if (!input->result)
00619     input->result = (char *) xmlStrdup (result_name);
00620   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00621   if (!input->variables)
00622     input->variables = (char *) xmlStrdup (variables_name);
00623
00624   // Opening simulator program name
00625   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00626   if (!input->simulator)
00627     {
00628       msg = gettext ("Bad simulator program");
00629       goto exit_on_error;
00630     }
00631
00632   // Opening evaluator program name
00633   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00634
00635   // Obtaining pseudo-random numbers generator seed
00636   input->seed
00637     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00638                                       &error_code);
00639   if (error_code)
00640     {
00641       msg = gettext ("Bad pseudo-random numbers generator seed");
00642       goto exit_on_error;
00643     }
00644
00645   // Opening algorithm
00646   buffer = xmlGetProp (node, XML_ALGORITHM);
00647   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00648     {
00649       input->algorithm = ALGORITHM_MONTE_CARLO;
00650
00651       // Obtaining simulations number
00652       input->nsimulations
00653         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00654       if (error_code)
00655         {
00656           msg = gettext ("Bad simulations number");
00657           goto exit_on_error;
00658         }
00659     }
00660   else if (!xmlStrcmp (buffer, XML_SWEEP))
00661     input->algorithm = ALGORITHM_SWEEP;
00662   else if (!xmlStrcmp (buffer, XML_GENETIC))
00663     {
00664       input->algorithm = ALGORITHM_GENETIC;
00665
00666       // Obtaining population
00667       if (xmlHasProp (node, XML_NPOPULATION))
00668         {
00669           input->nsimulations
00670             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00671           if (error_code || input->nsimulations < 3)
00672             {
00673               msg = gettext ("Invalid population number");
00674               goto exit_on_error;
00675             }
00676         }
00677       else
00678         {
00679           msg = gettext ("No population number");
00680           goto exit_on_error;
00681         }
00682
00683       // Obtaining generations
00684       if (xmlHasProp (node, XML_NGENERATIONS))
00685         {
00686           input->niterations
00687             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00688           if (error_code || !input->niterations)
00689             {
00690               msg = gettext ("Invalid generations number");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         {
00696           msg = gettext ("No generations number");
00697           goto exit_on_error;
00698         }
00699
00700       // Obtaining mutation probability
00701       if (xmlHasProp (node, XML_MUTATION))
```

```
00702         {
00703           input->mutation_ratio
00704             = xml_node_get_float (node, XML_MUTATION, &error_code);
00705           if (error_code || input->mutation_ratio < 0.
00706               || input->mutation_ratio >= 1.)
00707             {
00708               msg = gettext ("Invalid mutation probability");
00709               goto exit_on_error;
00710             }
00711         }
00712       else
00713         {
00714           msg = gettext ("No mutation probability");
00715           goto exit_on_error;
00716         }
00717
00718       // Obtaining reproduction probability
00719       if (xmlHasProp (node, XML_REPRODUCTION))
00720         {
00721           input->reproduction_ratio
00722             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00723           if (error_code || input->reproduction_ratio < 0.
00724               || input->reproduction_ratio >= 1.0)
00725             {
00726               msg = gettext ("Invalid reproduction probability");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         {
00732           msg = gettext ("No reproduction probability");
00733           goto exit_on_error;
00734         }
00735
00736       // Obtaining adaptation probability
00737       if (xmlHasProp (node, XML_ADAPTATION))
00738         {
00739           input->adaptation_ratio
00740             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00741           if (error_code || input->adaptation_ratio < 0.
00742               || input->adaptation_ratio >= 1.)
00743             {
00744               msg = gettext ("Invalid adaptation probability");
00745               goto exit_on_error;
00746             }
00747         }
00748       else
00749         {
00750           msg = gettext ("No adaptation probability");
00751           goto exit_on_error;
00752         }
00753
00754       // Checking survivals
00755       i = input->mutation_ratio * input->nsimulations;
00756       i += input->reproduction_ratio * input->
      nsimulations;
00757       i += input->adaptation_ratio * input->
      nsimulations;
00758       if (i > input->nsimulations - 2)
00759         {
00760           msg = gettext
00761             ("No enough survival entities to reproduce the population");
00762           goto exit_on_error;
00763         }
00764     }
00765   else
00766     {
00767       msg = gettext ("Unknown algorithm");
00768       goto exit_on_error;
00769     }
00770   xmlFree (buffer);
00771   buffer = NULL;
00772
00773   if (input->algorithm == ALGORITHM_MONTE_CARLO
00774       || input->algorithm == ALGORITHM_SWEEP)
00775     {
00776
00777       // Obtaining iterations number
00778       input->niterations
00779         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00780       if (error_code == 1)
00781         input->niterations = 1;
00782       else if (error_code)
00783         {
00784           msg = gettext ("Bad iterations number");
00785           goto exit_on_error;
00786         }
```

```
00787
00788       // Obtaining best number
00789       input->nbest
00790         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00791       if (error_code || !input->nbest)
00792         {
00793           msg = gettext ("Invalid best number");
00794           goto exit_on_error;
00795         }
00796
00797       // Obtaining tolerance
00798       input->tolerance
00799         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00800                                          &error_code);
00801       if (error_code || input->tolerance < 0.)
00802         {
00803           msg = gettext ("Invalid tolerance");
00804           goto exit_on_error;
00805         }
00806
00807       // Getting gradient method parameters
00808       if (xmlHasProp (node, XML_NSTEPS))
00809         {
00810           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00811           if (error_code || !input->nsteps)
00812             {
00813               msg = gettext ("Invalid steps number");
00814               goto exit_on_error;
00815             }
00816           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00817           if (!xmlStrcmp (buffer, XML_COORDINATES))
00818             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00819           else if (!xmlStrcmp (buffer, XML_RANDOM))
00820             {
00821               input->gradient_method =
      GRADIENT_METHOD_RANDOM;
00822               input->nestimates
00823                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00824               if (error_code || !input->nestimates)
00825                 {
00826                   msg = gettext ("Invalid estimates number");
00827                   goto exit_on_error;
00828                 }
00829             }
00830           else
00831             {
00832               msg = gettext ("Unknown method to estimate the gradient");
00833               goto exit_on_error;
00834             }
00835           xmlFree (buffer);
00836           buffer = NULL;
00837           input->relaxation
00838             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00839                                                DEFAULT_RELAXATION, &error_code);
00840           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00841             {
00842               msg = gettext ("Invalid relaxation parameter");
00843               goto exit_on_error;
00844             }
00845         }
00846       else
00847         input->nsteps = 0;
00848     }
00849
00850   // Reading the experimental data
00851   for (child = node->children; child; child = child->next)
00852     {
00853       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00854         break;
00855 #if DEBUG
00856       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00857 #endif
00858       if (xmlHasProp (child, XML_NAME))
00859         buffer = xmlGetProp (child, XML_NAME);
00860       else
00861         {
00862           snprintf (buffer2, 64, "%s %u: %s",
00863                     gettext ("Experiment"),
00864                     input->nexperiments + 1, gettext ("no data file name"));
00865           msg = buffer2;
00866           goto exit_on_error;
```

```
00867         }
00868 #if DEBUG
00869       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00870 #endif
00871       input->weight = g_realloc (input->weight,
00872                                  (1 + input->nexperiments) * sizeof (double));
00873       input->weight[input->nexperiments]
00874         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00875       if (error_code)
00876         {
00877           snprintf (buffer2, 64, "%s %s: %s",
00878                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00879           msg = buffer2;
00880           goto exit_on_error;
00881         }
00882 #if DEBUG
00883       fprintf (stderr, "input_open: weight=%lg\n",
00884                input->weight[input->nexperiments]);
00885 #endif
00886       if (!input->nexperiments)
00887         input->ninputs = 0;
00888 #if DEBUG
00889       fprintf (stderr, "input_open: template[0]\n");
00890 #endif
00891       if (xmlHasProp (child, XML_TEMPLATE1))
00892         {
00893           input->template[0]
00894             = (char **) g_realloc (input->template[0],
00895                                    (1 + input->nexperiments) * sizeof (char *));
00896           buffert[0] = (char *) xmlGetProp (child, template[0]);
00897 #if DEBUG
00898           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00899                    input->nexperiments, buffert[0]);
00900 #endif
00901           if (!input->nexperiments)
00902             ++input->ninputs;
00903 #if DEBUG
00904           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00905 #endif
00906         }
00907       else
00908         {
00909           snprintf (buffer2, 64, "%s %s: %s",
00910                     gettext ("Experiment"), buffer, gettext ("no template"));
00911           msg = buffer2;
00912           goto exit_on_error;
00913         }
00914       for (i = 1; i < MAX_NINPUTS; ++i)
00915         {
00916 #if DEBUG
00917           fprintf (stderr, "input_open: template%u\n", i + 1);
00918 #endif
00919           if (xmlHasProp (child, template[i]))
00920             {
00921               if (input->nexperiments && input->ninputs <= i)
00922                 {
00923                   snprintf (buffer2, 64, "%s %s: %s",
00924                             gettext ("Experiment"),
00925                             buffer, gettext ("bad templates number"));
00926                   msg = buffer2;
00927                   while (i-- > 0)
00928                     xmlFree (buffert[i]);
00929                   goto exit_on_error;
00930                 }
00931               input->template[i] = (char **)
00932                 g_realloc (input->template[i],
00933                            (1 + input->nexperiments) * sizeof (char *));
00934               buffert[i] = (char *) xmlGetProp (child, template[i]);
00935 #if DEBUG
00936               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00937                        input->nexperiments, i + 1,
00938                        input->template[i][input->nexperiments]);
00939 #endif
00940               if (!input->nexperiments)
00941                 ++input->ninputs;
00942 #if DEBUG
00943               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00944 #endif
00945             }
00946           else if (input->nexperiments && input->ninputs > i)
00947             {
00948               snprintf (buffer2, 64, "%s %s: %s%u",
00949                         gettext ("Experiment"),
00950                         buffer, gettext ("no template"), i + 1);
00951               msg = buffer2;
00952               while (i-- > 0)
```

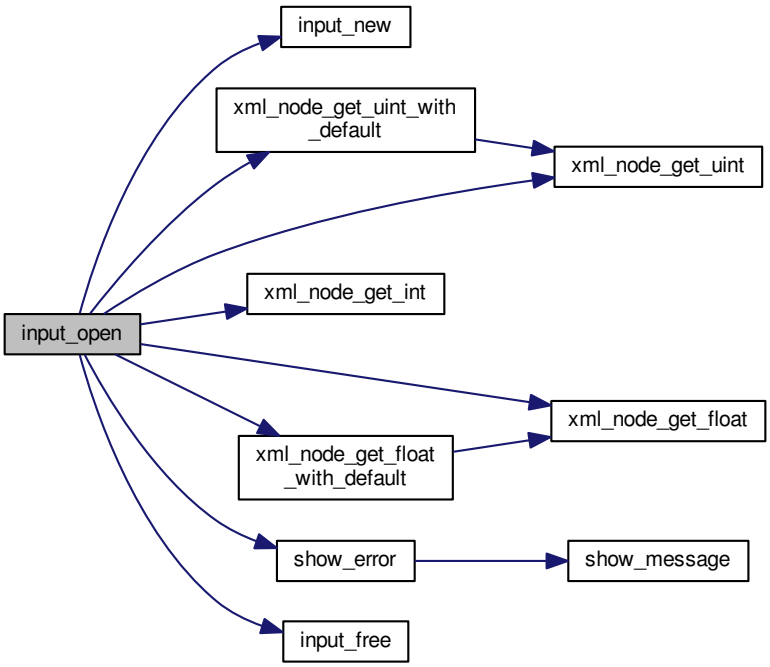
```
00953                 xmlFree (buffert[i]);
00954               goto exit_on_error;
00955             }
00956           else
00957             break;
00958         }
00959       input->experiment
00960         = g_realloc (input->experiment,
00961                      (1 + input->nexperiments) * sizeof (char *));
00962       input->experiment[input->nexperiments] = (char *) buffer;
00963       for (i = 0; i < input->ninputs; ++i)
00964         input->template[i][input->nexperiments] = buffert[i];
00965       ++input->nexperiments;
00966 #if DEBUG
00967       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00968 #endif
00969     }
00970   if (!input->nexperiments)
00971     {
00972       msg = gettext ("No calibration experiments");
00973       goto exit_on_error;
00974     }
00975   buffer = NULL;
00976
00977   // Reading the variables data
00978   for (; child; child = child->next)
00979     {
00980       if (xmlStrcmp (child->name, XML_VARIABLE))
00981         {
00982           snprintf (buffer2, 64, "%s %u: %s",
00983                     gettext ("Variable"),
00984                     input->nvariables + 1, gettext ("bad XML node"));
00985           msg = buffer2;
00986           goto exit_on_error;
00987         }
00988       if (xmlHasProp (child, XML_NAME))
00989         buffer = xmlGetProp (child, XML_NAME);
00990       else
00991         {
00992           snprintf (buffer2, 64, "%s %u: %s",
00993                     gettext ("Variable"),
00994                     input->nvariables + 1, gettext ("no name"));
00995           msg = buffer2;
00996           goto exit_on_error;
00997         }
00998       if (xmlHasProp (child, XML_MINIMUM))
00999         {
01000           input->rangemin = g_realloc
01001             (input->rangemin, (1 + input->nvariables) * sizeof (double));
01002           input->rangeminabs = g_realloc
01003             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
01004           input->rangemin[input->nvariables]
01005             = xml_node_get_float (child, XML_MINIMUM, &error_code);
01006           if (error_code)
01007             {
01008               snprintf (buffer2, 64, "%s %s: %s",
01009                         gettext ("Variable"), buffer, gettext ("bad minimum"));
01010               msg = buffer2;
01011               goto exit_on_error;
01012             }
01013           input->rangeminabs[input->nvariables]
01014             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MINIMUM,
01015                                                -G_MAXDOUBLE, &error_code);
01016           if (error_code)
01017             {
01018               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01019                         gettext ("bad absolute minimum"));
01020               msg = buffer2;
01021               goto exit_on_error;
01022             }
01023           if (input->rangemin[input->nvariables]
01024               < input->rangeminabs[input->nvariables])
01025             {
01026               snprintf (buffer2, 64, "%s %s: %s",
01027                         gettext ("Variable"),
01028                         buffer, gettext ("minimum range not allowed"));
01029               msg = buffer2;
01030               goto exit_on_error;
01031             }
01032         }
01033       else
01034         {
01035           snprintf (buffer2, 64, "%s %s: %s",
01036                     gettext ("Variable"), buffer, gettext ("no minimum range"));
01037           msg = buffer2;
01038           goto exit_on_error;
```

```
01039         }
01040       if (xmlHasProp (child, XML_MAXIMUM))
01041         {
01042           input->rangemax = g_realloc
01043             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01044           input->rangemaxabs = g_realloc
01045             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01046           input->rangemax[input->nvariables]
01047             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01048           if (error_code)
01049             {
01050               snprintf (buffer2, 64, "%s %s: %s",
01051                         gettext ("Variable"), buffer, gettext ("bad maximum"));
01052               msg = buffer2;
01053               goto exit_on_error;
01054             }
01055           input->rangemaxabs[input->nvariables]
01056             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MAXIMUM,
01057                                                G_MAXDOUBLE, &error_code);
01058           if (error_code)
01059             {
01060               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01061                         gettext ("bad absolute maximum"));
01062               msg = buffer2;
01063               goto exit_on_error;
01064             }
01065           if (input->rangemax[input->nvariables]
01066               > input->rangemaxabs[input->nvariables])
01067             {
01068               snprintf (buffer2, 64, "%s %s: %s",
01069                         gettext ("Variable"),
01070                         buffer, gettext ("maximum range not allowed"));
01071               msg = buffer2;
01072               goto exit_on_error;
01073             }
01074         }
01075       else
01076         {
01077           snprintf (buffer2, 64, "%s %s: %s",
01078                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01079           msg = buffer2;
01080           goto exit_on_error;
01081         }
01082       if (input->rangemax[input->nvariables]
01083           < input->rangemin[input->nvariables])
01084         {
01085           snprintf (buffer2, 64, "%s %s: %s",
01086                     gettext ("Variable"), buffer, gettext ("bad range"));
01087           msg = buffer2;
01088           goto exit_on_error;
01089         }
01090       input->precision = g_realloc
01091         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01092       input->precision[input->nvariables]
01093         = xml_node_get_uint_with_default (child,
    XML_PRECISION,
01094                                           DEFAULT_PRECISION, &error_code);
01095       if (error_code || input->precision[input->nvariables] >=
    NPRECISIONS)
01096         {
01097           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                     gettext ("bad precision"));
01099           msg = buffer2;
01100           goto exit_on_error;
01101         }
01102       if (input->algorithm == ALGORITHM_SWEEP)
01103         {
01104           if (xmlHasProp (child, XML_NSWEEPS))
01105             {
01106               input->nsweeps = (unsigned int *)
01107                 g_realloc (input->nsweeps,
01108                            (1 + input->nvariables) * sizeof (unsigned int));
01109               input->nsweeps[input->nvariables]
01110                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01111               if (error_code || !input->nsweeps[input->
    nvariables])
01112                 {
01113                   snprintf (buffer2, 64, "%s %s: %s",
01114                             gettext ("Variable"),
01115                             buffer, gettext ("bad sweeps"));
01116                   msg = buffer2;
01117                   goto exit_on_error;
01118                 }
01119             }
01120           else
01121             {
```

```
01122               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01123                         gettext ("no sweeps number"));
01124               msg = buffer2;
01125               goto exit_on_error;
01126             }
01127 #if DEBUG
01128           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01129                    input->nsweeps[input->nvariables],
      input->nsimulations);
01130 #endif
01131         }
01132       if (input->algorithm == ALGORITHM_GENETIC)
01133         {
01134           // Obtaining bits representing each variable
01135           if (xmlHasProp (child, XML_NBITS))
01136             {
01137               input->nbits = (unsigned int *)
01138                 g_realloc (input->nbits,
01139                            (1 + input->nvariables) * sizeof (unsigned int));
01140               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01141               if (error_code || !i)
01142                 {
01143                   snprintf (buffer2, 64, "%s %s: %s",
01144                             gettext ("Variable"),
01145                             buffer, gettext ("invalid bits number"));
01146                   msg = buffer2;
01147                   goto exit_on_error;
01148                 }
01149               input->nbits[input->nvariables] = i;
01150             }
01151           else
01152             {
01153               snprintf (buffer2, 64, "%s %s: %s",
01154                         gettext ("Variable"),
01155                         buffer, gettext ("no bits number"));
01156               msg = buffer2;
01157               goto exit_on_error;
01158             }
01159         }
01160       else if (input->nsteps)
01161         {
01162           input->step = (double *)
01163             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01164           input->step[input->nvariables]
01165             = xml_node_get_float (child, XML_STEP, &error_code);
01166           if (error_code || input->step[input->nvariables] < 0.)
01167             {
01168               snprintf (buffer2, 64, "%s %s: %s",
01169                         gettext ("Variable"),
01170                         buffer, gettext ("bad step size"));
01171               msg = buffer2;
01172               goto exit_on_error;
01173             }
01174         }
01175       input->label = g_realloc
01176         (input->label, (1 + input->nvariables) * sizeof (char *));
01177       input->label[input->nvariables] = (char *) buffer;
01178       ++input->nvariables;
01179     }
01180   if (!input->nvariables)
01181     {
01182       msg = gettext ("No calibration variables");
01183       goto exit_on_error;
01184     }
01185   buffer = NULL;
01186
01187   // Obtaining the error norm
01188   if (xmlHasProp (node, XML_NORM))
01189     {
01190       buffer = xmlGetProp (node, XML_NORM);
01191       if (!xmlStrcmp (buffer, XML_EUCLIDIAN))
01192         input->norm = ERROR_NORM_EUCLIDIAN;
01193       else if (!xmlStrcmp (buffer, XML_MAXIMUM))
01194         input->norm = ERROR_NORM_MAXIMUM;
01195       else if (!xmlStrcmp (buffer, XML_P))
01196         {
01197           input->norm = ERROR_NORM_P;
01198           input->p = xml_node_get_float (node, XML_P, &error_code);
01199           if (!error_code)
01200             {
01201               msg = gettext ("Bad P parameter");
01202               goto exit_on_error;
01203             }
01204         }
01205       else if (!xmlStrcmp (buffer, XML_TAXICAB))
01206         input->norm = ERROR_NORM_TAXICAB;
01207       else
```

```
01208         {
01209           msg = gettext ("Unknown error norm");
01210           goto exit_on_error;
01211         }
01212       xmlFree (buffer);
01213     }
01214   else
01215     input->norm = ERROR_NORM_EUCLIDIAN;
01216
01217   // Getting the working directory
01218   input->directory = g_path_get_dirname (filename);
01219   input->name = g_path_get_basename (filename);
01220
01221   // Closing the XML document
01222   xmlFreeDoc (doc);
01223
01224 #if DEBUG
01225   fprintf (stderr, "input_open: end\n");
01226 #endif
01227   return 1;
01228
01229 exit_on_error:
01230   xmlFree (buffer);
01231   xmlFreeDoc (doc);
01232   show_error (msg);
01233   input_free ();
01234 #if DEBUG
01235   fprintf (stderr, "input_open: end\n");
01236 #endif
01237   return 0;
01238 }
```

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.

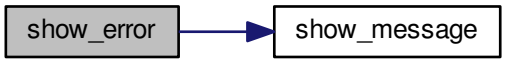
Parameters

msg	Error message.

Definition at line 259 of file mpcotool.c.

```
00260 {
00261   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00262 }
```

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 229 of file mpcotool.c.

```
00230 {
00231 #if HAVE_GTK
00232   GtkMessageDialog *dlg;
00233
00234   // Creating the dialog
00235   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00236     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00237
00238   // Setting the dialog title
00239   gtk_window_set_title (GTK_WINDOW (dlg), title);
00240
00241   // Showing the dialog and waiting response
00242   gtk_dialog_run (GTK_DIALOG (dlg));
00243
00244   // Closing and freeing memory
00245   gtk_widget_destroy (GTK_WIDGET (dlg));
00246
00247 #else
00248   printf ("%s: %s\n", title, msg);
00249 #endif
00250 }
```

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 369 of file mpcotool.c.

```
00370 {
00371   double x = 0.;
00372   xmlChar *buffer;
00373   buffer = xmlGetProp (node, prop);
00374   if (!buffer)
00375     *error_code = 1;
00376   else
00377     {
00378       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00379         *error_code = 2;
00380       else
00381         *error_code = 0;
00382       xmlFree (buffer);
00383     }
00384   return x;
00385 }
```

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.

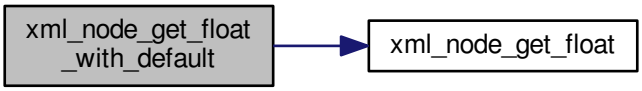
Returns

Floating point number value.

Definition at line 403 of file mpcotool.c.

```
00405 {
00406   double x;
00407   if (xmlHasProp (node, prop))
00408     x = xml_node_get_float (node, prop, error_code);
00409   else
00410     {
00411       x = default_value;
00412       *error_code = 0;
00413     }
00414   return x;
00415 }
```

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 277 of file mpcotool.c.

```
00278 {
00279   int i = 0;
00280   xmlChar *buffer;
00281   buffer = xmlGetProp (node, prop);
00282   if (!buffer)
00283     *error_code = 1;
00284   else
00285     {
00286       if (sscanf ((char *) buffer, "%d", &i) != 1)
00287         *error_code = 2;
00288       else
00289         *error_code = 0;
00290       xmlFree (buffer);
00291     }
00292   return i;
00293 }
```

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 308 of file mpcotool.c.

```
00309 {
00310   unsigned int i = 0;
00311   xmlChar *buffer;
00312   buffer = xmlGetProp (node, prop);
00313   if (!buffer)
00314     *error_code = 1;
00315   else
00316     {
00317       if (sscanf ((char *) buffer, "%u", &i) != 1)
00318         *error_code = 2;
00319       else
00320         *error_code = 0;
00321       xmlFree (buffer);
00322     }
00323   return i;
00324 }
```

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.

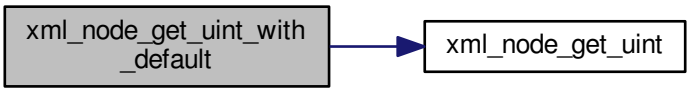
Returns

Unsigned integer number value.

Definition at line 342 of file mpcotool.c.

```
00344 {
00345   unsigned int i;
00346   if (xmlHasProp (node, prop))
00347     i = xml_node_get_uint (node, prop, error_code);
00348   else
00349     {
00350       i = default_value;
00351       *error_code = 0;
00352     }
00353   return i;
00354 }
```

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 466 of file mpcotool.c.

```
00467 {
00468   xmlChar buffer[64];
00469   snprintf ((char *) buffer, 64, "%.14lg", value);
00470   xmlSetProp (node, prop, buffer);
00471 }
```

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 428 of file mpcotool.c.

```
00429 {
00430   xmlChar buffer[64];
00431   snprintf ((char *) buffer, 64, "%d", value);
00432   xmlSetProp (node, prop, buffer);
00433 }
```

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 447 of file mpcotool.c.

```
00448 {
00449   xmlChar buffer[64];
00450   snprintf ((char *) buffer, 64, "%u", value);
00451   xmlSetProp (node, prop, buffer);
00452 }
```

5.8 mpcotool.h

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2016, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
```

```
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef MPCOTOOL__H
00037 #define MPCOTOOL__H 1
00038
00043 enum Algorithm
00044 {
00045   ALGORITHM_MONTE_CARLO = 0,
00046   ALGORITHM_SWEEP = 1,
00047   ALGORITHM_GENETIC = 2
00048 };
00049
00054 enum GradientMethod
00055 {
00056   GRADIENT_METHOD_COORDINATES = 0,
00057   GRADIENT_METHOD_RANDOM = 1,
00058 };
00059
00064 enum ErrorNorm
00065 {
00066   ERROR_NORM_EUCLIDIAN = 0,
00068   ERROR_NORM_MAXIMUM = 1,
00070   ERROR_NORM_P = 2,
00072   ERROR_NORM_TAXICAB = 3
00074 };
00075
00080 typedef struct
00081 {
00082   char **template[MAX_NINPUTS];
00083   char **experiment;
00084   char **label;
00085   char *result;
00086   char *variables;
00087   char *simulator;
00088   char *evaluator;
00090   char *directory;
00091   char *name;
00092   double *rangemin;
00093   double *rangemax;
00094   double *rangeminabs;
00095   double *rangemaxabs;
00096   double *weight;
00097   double *step;
00098   unsigned int *precision;
00099   unsigned int *nsweeps;
00100   unsigned int *nbits;
00102   double tolerance;
00103   double mutation_ratio;
00104   double reproduction_ratio;
00105   double adaptation_ratio;
00106   double relaxation;
00107   double p;
00108   unsigned long int seed;
00110   unsigned int nvariables;
00111   unsigned int nexperiments;
00112   unsigned int ninputs;
00113   unsigned int nsimulations;
00114   unsigned int algorithm;
00115   unsigned int nsteps;
00117   unsigned int gradient_method;
00118   unsigned int nestimates;
00120   unsigned int niterations;
00121   unsigned int nbest;
00122   unsigned int norm;
00123 } Input;
00124
00129 typedef struct
00130 {
00131   GMappedFile **file[MAX_NINPUTS];
00132   char **template[MAX_NINPUTS];
00133   char **experiment;
00134   char **label;
00135   gsl_rng *rng;
```

```
00136   GeneticVariable *genetic_variable;
00138   FILE *file_result;
00139   FILE *file_variables;
00140   char *result;
00141   char *variables;
00142   char *simulator;
00143   char *evaluator;
00145   double *value;
00146   double *rangemin;
00147   double *rangemax;
00148   double *rangeminabs;
00149   double *rangemaxabs;
00150   double *error_best;
00151   double *weight;
00152   double *step;
00153   double *gradient;
00154   double *value_old;
00156   double *error_old;
00158   unsigned int *precision;
00159   unsigned int *nsweeps;
00160   unsigned int *thread;
00162   unsigned int *thread_gradient;
00165   unsigned int *simulation_best;
00166   double tolerance;
00167   double mutation_ratio;
00168   double reproduction_ratio;
00169   double adaptation_ratio;
00170   double relaxation;
00171   double calculation_time;
00172   double p;
00173   unsigned long int seed;
00175   unsigned int nvariables;
00176   unsigned int nexperiments;
00177   unsigned int ninputs;
00178   unsigned int nsimulations;
00179   unsigned int gradient_method;
00180   unsigned int nsteps;
00182   unsigned int nestimates;
00184   unsigned int algorithm;
00185   unsigned int nstart;
00186   unsigned int nend;
00187   unsigned int nstart_gradient;
00189   unsigned int nend_gradient;
00191   unsigned int niterations;
00192   unsigned int nbest;
00193   unsigned int nsaveds;
00194 #if HAVE_MPI
00195   int mpi_rank;
00196 #endif
00197 } Calibrate;
00198
00203 typedef struct
00204 {
00205   unsigned int thread;
00206 } ParallelData;
00207
00208 // Public functions
00209 void show_message (char *title, char *msg, int type);
00210 void show_error (char *msg);
00211 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00212 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00213                                 int *error_code);
00214 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00215                                              const xmlChar * prop,
00216                                              unsigned int default_value,
00217                                              int *error_code);
00218 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00219                            int *error_code);
00220 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
    ,
00221                                         double default_value, int *error_code);
00222 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00223 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00224                         unsigned int value);
00225 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00226
00227 void input_new ();
00228 void input_free ();
00229 int input_open (char *filename);
00230 void calibrate_input (unsigned int simulation, char *input,
00231                       GMappedFile * template);
00232 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00233 void calibrate_print ();
00234 void calibrate_save_variables (unsigned int simulation, double error);
00235 void calibrate_best (unsigned int simulation, double value);
00236 void calibrate_sequential ();
00237 void *calibrate_thread (ParallelData * data);
```

```
00238 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00239                       double *error_best);
00240 #if HAVE_MPI
00241 void calibrate_synchronise ();
00242 #endif
00243 void calibrate_sweep ();
00244 void calibrate_MonteCarlo ();
00245 void calibrate_best_gradient (unsigned int simulation, double value);
00246 void calibrate_gradient_sequential ();
00247 void *calibrate_gradient_thread (ParallelData * data);
00248 double calibrate_variable_step_gradient (unsigned int variable);
00249 void calibrate_step_gradient (unsigned int simulation);
00250 void calibrate_gradient ();
00251 double calibrate_genetic_objective (Entity * entity);
00252 void calibrate_genetic ();
00253 void calibrate_save_old ();
00254 void calibrate_merge_old ();
00255 void calibrate_refine ();
00256 void calibrate_step ();
00257 void calibrate_iterate ();
00258 void calibrate_open ();
00259
00260 #endif
```

Index