Calibrator

1.2.5

Generated by Doxygen 1.8.2

Wed Jan 13 2016 10:04:02

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

1. Download this repository:

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

1. Link the latest genetic version to genetic:

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

1. Build doing on a terminal:

   $ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

1. 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

1. Then, in a MSYS2 terminal, follow steps 1 to 4 of the previous Debian 8 section.

1. 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

1. 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

1. 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.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int gradient_method

  Method to estimate the gradient.
- unsigned int nsteps

Number of steps for the gradient based method.

- unsigned int nestimates

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

  Algorithm type.
- unsigned int nstart

  Beginning simulation number of the task.
- unsigned int nend

  Ending simulation number of the task.
- unsigned int nstart_gradient

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

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 111 of file mpcotool.h.

4.1.2 Field Documentation

4.1.2.1 unsigned int* Calibrate::thread_gradient

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

Definition at line 145 of file mpcotool.h.

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

- mpcotool.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

- char * template [MAX_NINPUTS]

  Array of input template names.
- char * name

  File name.
- double weight

  Weight to calculate the objective function value.

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <mpcotool.h>`

Data Fields

- char ** template [MAX_NINPUTS]
  Matrix of template names of input files.
- char ** experiment
  Array of experimental data file names.
- char ** label
  Array of variable names.
- char * result
  Name of the result file.
- char * variables
  Name of the variables file.
- char * simulator
  Name of the simulator program.
- char * evaluator
  Name of the program to evaluate the objective function.
- char * directory
  Working directory.
- char * name
  Input data file name.
- double * rangemin
  Array of minimum variable values.
- double * rangemax
  Array of maximum variable values.
- double * rangeminabs
  Array of absolute minimum variable values.
- double * rangemaxabs
  Array of absolute maximum variable values.
- double * weight
  Array of the experiment weights.
- double * step
  Array of gradient based method step sizes.
- unsigned int * precision
  Array of variable precisions.
- unsigned int * nsweeps
  Array of sweeps of the sweep algorithm.

- unsigned int * nbits

  Array of bits numbers of the genetic algorithm.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int algorithm

  Algorithm type.
- unsigned int nsteps

  Number of steps to do the gradient based method.
- unsigned int gradient_method

  Method to estimate the gradient.
- unsigned int nestimates

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 64 of file mpcotool.h.

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

- mpcotool.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_seed

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

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

  Threads number GtkLabel.
- GtkSpinButton * spin_threads

  Threads number GtkSpinButton.
- GtkLabel * label_gradient

  Gradient threads number GtkLabel.
- GtkSpinButton * spin_gradient

  Gradient threads number GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 76 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <mpcotool.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 184 of file mpcotool.h.

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

- mpcotool.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkLabel * label

  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 94 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label

  Variable label.
- double rangemin

  Minimum value.
- double rangemax

  Maximum value.
- double rangeminabs

  Minimum allowed value.
- double rangemaxabs

  Maximum allowed value.
- double step

  Initial step size for the gradient based method.
- unsigned int precision

  Precision digits.
- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file interface.h.

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

- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:

Data Fields

- GtkWindow * window
  Main GtkWindow.
- GtkGrid * grid
  Main GtkGrid.
- GtkToolbar * bar_buttons
  GtkToolbar to store the main buttons.
- GtkToolButton * button_open
  Open GtkToolButton.
- GtkToolButton * button_save
  Save GtkToolButton.
- GtkToolButton * button_run
  Run GtkToolButton.
- GtkToolButton * button_options
  Options GtkToolButton.
- GtkToolButton * button_help
  Help GtkToolButton.
- GtkToolButton * button_about
  Help GtkToolButton.
- GtkToolButton * button_exit
  Exit GtkToolButton.
- GtkGrid * grid_files
  Files GtkGrid.
- GtkLabel * label_simulator
  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator
  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator
  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator
  Evaluator program GtkFileChooserButton.
- GtkLabel * label_result
  Result file GtkLabel.
- GtkEntry * entry_result

Result file GtkEntry.

- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

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

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

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton * check_gradient

  GtkCheckButton to check running the gradient based method.
- GtkGrid * grid_gradient

  GtkGrid to pack the gradient based method widgets.

- GtkRadioButton * button_gradient [NGRADIENTS]

  GtkRadioButtons array to set the gradient estimate method.
- GtkLabel * label_steps

  GtkLabel to set the steps number.
- GtkSpinButton * spin_steps

  GtkSpinButton to set the steps number.
- GtkLabel * label_estimates

  GtkLabel to set the estimates number.
- GtkSpinButton * spin_estimates

  GtkSpinButton to set the estimates number.
- GtkLabel * label_relaxation

  GtkLabel to set the relaxation parameter.
- GtkSpinButton * spin_relaxation

  GtkSpinButton to set the relaxation parameter.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.
- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

Absolute maximum GtkScrolledWindow.

- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkLabel * label_step

  GtkLabel to set the step.
- GtkSpinButton * spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow * scrolled_step

  step GtkScrolledWindow.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

  GtkFileChooserButton to set the experimental data file.
- GtkLabel * label_weight

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

  Array of GtkCheckButtons to set the input templates.
- GtkFileChooserButton * button_template [MAX_NINPUTS]

  Array of GtkFileChooserButtons to set the input templates.
- GdkPixbuf * logo

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.

- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

  Identifier of the entry_variable signal.
- gulong id_template [MAX_NINPUTS]

  Array of identifiers of the check_template signal.
- gulong id_input [MAX_NINPUTS]

  Array of identifiers of the button_template signal.
- unsigned int nexperiments

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 104 of file interface.h.

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

- interface.h

Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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

Macros

- #define MAX_NINPUTS 8

  Maximum number of input files in the simulator program.
- #define NALGORITHMS 3

  Number of stochastic algorithms.
- #define NGRADIENTS 2

  Number of gradient estimate methods.
- #define NPRECISIONS 15

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

  Default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

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

  Default relaxation parameter.
- #define LOCALE_DIR "locales"

  Locales directory.
- #define PROGRAM_INTERFACE "mpcotool"

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

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

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

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

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

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

coordinates XML label.

- #define XML_EVALUATOR (const xmlChar*)"evaluator"

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

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

  genetic XML label.
- #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"

  gradient_method XML label.
- #define XML_MINIMUM (const xmlChar*)"minimum"

  minimum XML label.
- #define XML_MAXIMUM (const xmlChar*)"maximum"

  maximum XML label.
- #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"

  Monte-Carlo XML label.
- #define XML_MUTATION (const xmlChar*)"mutation"

  mutation XML label.
- #define XML_NAME (const xmlChar*)"name"

  name XML label.
- #define XML_NBEST (const xmlChar*)"nbest"

  nbest XML label.
- #define XML_NBITS (const xmlChar*)"nbits"

  nbits XML label.
- #define XML_NESTIMATES (const xmlChar*)"nestimates"

  nestimates XML label.
- #define XML_NGENERATIONS (const xmlChar*)"ngenerations"

  ngenerations XML label.
- #define XML_NITERATIONS (const xmlChar*)"niterations"

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

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

  nsimulations XML label.
- #define XML_NSTEPS (const xmlChar*)"nsteps"

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

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

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

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

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

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

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

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

  seed XML label.

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

5.1.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file config.h.

5.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without
```

```
       modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright
       notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
       EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
       IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043
00044 #define NALGORITHMS 3
00045 #define NGRADIENTS 2
00046 #define NPRECISIONS 15
00047
00048 // Default choices
00049
00050 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00051 #define DEFAULT_RANDOM_SEED 7007
00052 #define DEFAULT_RELAXATION 1.
00053
00054 // Interface labels
00055
00056 #define LOCALE_DIR "locales"
00057 #define PROGRAM_INTERFACE "mpcotool"
00058
00059 // XML labels
00060
00061 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00062
00063 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00064
00065 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00066
00067 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00068
00069 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00070
00071 #define XML_COORDINATES (const xmlChar*)"coordinates"
00072
00073 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00074
00075 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00076
00077 #define XML_GENETIC (const xmlChar*)"genetic"
00078 #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
00079
00080 #define XML_MINIMUM (const xmlChar*)"minimum"
00081 #define XML_MAXIMUM (const xmlChar*)"maximum"
00082 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00083
00084 #define XML_MUTATION (const xmlChar*)"mutation"
00085 #define XML_NAME (const xmlChar*)"name"
00086 #define XML_NBEST (const xmlChar*)"nbest"
00087 #define XML_NBITS (const xmlChar*)"nbits"
00088 #define XML_NESTIMATES (const xmlChar*)"nestimates"
00089
00090 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00091
00092 #define XML_NITERATIONS (const xmlChar*)"niterations"
00093
00094 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00095
00096 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00097
00098 #define XML_NSTEPS (const xmlChar*)"nsteps"
```

```
00099 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00100 #define XML_PRECISION (const xmlChar*)"precision"
00101
00102 #define XML_RANDOM (const xmlChar*)"random"
00103 #define XML_RELAXATION (const xmlChar*)"relaxation"
00104
00105 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00106
00107 #define XML_RESULT (const xmlChar*)"result"
00108 #define XML_SIMULATOR (const xmlChar*)"simulator"
00109
00110 #define XML_SEED (const xmlChar*)"seed"
00111 #define XML_STEP (const xmlChar*)"step"
00112 #define XML_SWEEP (const xmlChar*)"sweep"
00113 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00114
00115 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00116
00117 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00118
00119 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00120
00121 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00122
00123 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00124
00125 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00126
00127 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00128
00129 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00130
00131 #define XML_VARIABLE (const xmlChar*)"variable"
00132 #define XML_VARIABLES (const xmlChar*)"variables"
00133
00134 #define XML_WEIGHT (const xmlChar*)"weight"
00135
00136 #endif
```

5.3 interface.h File Reference

Header file of the interface.

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

Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- void input_save (char *filename)

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

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

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

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Function to update the variable rangemin in the main window.

- void window_rangemax_variable ()

  Function to update the variable rangemax in the main window.

- void window_rangeminabs_variable ()

  Function to update the variable rangeminabs in the main window.

- void window_rangemaxabs_variable ()

  Function to update the variable rangemaxabs in the main window.

- void window_update_variable ()

  Function to update the variable data in the main window.

- int window_read (char *filename)

  Function to read the input data of a file.

- void window_open ()

  Function to open the input data.

- void window_new ()

  Function to open the main window.

- int cores_number ()

  Function to obtain the cores number.

5.3.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Copyright 2012-2015, all rights reserved.

Definition in file interface.h.

5.3.2 Function Documentation

5.3.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4875 of file mpcotool.c.

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

5.3.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 2699 of file mpcotool.c.

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

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

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

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

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

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

  // Setting the algorithm
  buffer = (char *) g_malloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO
      );
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
```

```
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
      input_save_gradient (node);
      break;
    default:
      xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio
      );
      xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio
      );
      xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_free (buffer);

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

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

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

  // Freeing memory
  xmlFreeDoc (doc);

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

Here is the call graph for this function:

5.3.2.3 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2932 of file mpcotool.c.

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

5.3.2.4 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2955 of file mpcotool.c.

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

5.3.2.5 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 4052 of file mpcotool.c.

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

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

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

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

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

Here is the call graph for this function:

5.3.2.6 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 3010 of file mpcotool.c.

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

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

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

```
gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
g_free (buffer);

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

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

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

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

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

    // Closing and freeing memory
    g_free (buffer);
```

```
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

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

Here is the call graph for this function:

5.3.2.7 void window_template_experiment (void * *data*)

Function to update the experiment i-th input template in the main window.

Parameters

data	Callback data (i-th input template).

Definition at line 3656 of file mpcotool.c.

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

5.4 interface.h

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without
       modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
       notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
```

```
       EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
       IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef INTERFACE__H
00037 #define INTERFACE__H 1
00038
00039 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00040
00041
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_algorithm;
00128   GtkGrid *grid_algorithm;
00129   GtkRadioButton *button_algorithm[NALGORITHMS];
00131   GtkLabel *label_simulations;
00132   GtkSpinButton *spin_simulations;
00134   GtkLabel *label_iterations;
00135   GtkSpinButton *spin_iterations;
00137   GtkLabel *label_tolerance;
00138   GtkSpinButton *spin_tolerance;
00139   GtkLabel *label_bests;
00140   GtkSpinButton *spin_bests;
```

```
00141   GtkLabel *label_population;
00142   GtkSpinButton *spin_population;
00144   GtkLabel *label_generations;
00145   GtkSpinButton *spin_generations;
00147   GtkLabel *label_mutation;
00148   GtkSpinButton *spin_mutation;
00149   GtkLabel *label_reproduction;
00150   GtkSpinButton *spin_reproduction;
00152   GtkLabel *label_adaptation;
00153   GtkSpinButton *spin_adaptation;
00155   GtkCheckButton *check_gradient;
00157   GtkGrid *grid_gradient;
00159   GtkRadioButton *button_gradient[NGRADIENTS];
00161   GtkLabel *label_steps;
00162   GtkSpinButton *spin_steps;
00163   GtkLabel *label_estimates;
00164   GtkSpinButton *spin_estimates;
00166   GtkLabel *label_relaxation;
00168   GtkSpinButton *spin_relaxation;
00170   GtkFrame *frame_variable;
00171   GtkGrid *grid_variable;
00172   GtkComboBoxText *combo_variable;
00174   GtkButton *button_add_variable;
00175   GtkButton *button_remove_variable;
00176   GtkLabel *label_variable;
00177   GtkEntry *entry_variable;
00178   GtkLabel *label_min;
00179   GtkSpinButton *spin_min;
00180   GtkScrolledWindow *scrolled_min;
00181   GtkLabel *label_max;
00182   GtkSpinButton *spin_max;
00183   GtkScrolledWindow *scrolled_max;
00184   GtkCheckButton *check_minabs;
00185   GtkSpinButton *spin_minabs;
00186   GtkScrolledWindow *scrolled_minabs;
00187   GtkCheckButton *check_maxabs;
00188   GtkSpinButton *spin_maxabs;
00189   GtkScrolledWindow *scrolled_maxabs;
00190   GtkLabel *label_precision;
00191   GtkSpinButton *spin_precision;
00192   GtkLabel *label_sweeps;
00193   GtkSpinButton *spin_sweeps;
00194   GtkLabel *label_bits;
00195   GtkSpinButton *spin_bits;
00196   GtkLabel *label_step;
00197   GtkSpinButton *spin_step;
00198   GtkScrolledWindow *scrolled_step;
00199   GtkFrame *frame_experiment;
00200   GtkGrid *grid_experiment;
00201   GtkComboBoxText *combo_experiment;
00202   GtkButton *button_add_experiment;
00203   GtkButton *button_remove_experiment;
00204   GtkLabel *label_experiment;
00205   GtkFileChooserButton *button_experiment;
00207   GtkLabel *label_weight;
00208   GtkSpinButton *spin_weight;
00209   GtkCheckButton *check_template[MAX_NINPUTS];
00211   GtkFileChooserButton *button_template[MAX_NINPUTS];
00213   GdkPixbuf *logo;
00214   Experiment *experiment;
00215   Variable *variable;
00216   char *application_directory;
00217   gulong id_experiment;
00218   gulong id_experiment_name;
00219   gulong id_variable;
00220   gulong id_variable_label;
00221   gulong id_template[MAX_NINPUTS];
00223   gulong id_input[MAX_NINPUTS];
00225   unsigned int nexperiments;
00226   unsigned int nvariables;
00227 } Window;
00228
00229 // Public functions
00230 void input_save (char *filename);
00231 void options_new ();
00232 void running_new ();
00233 int window_get_algorithm ();
00234 int window_get_gradient ();
00235 void window_save_gradient ();
00236 int window_save ();
00237 void window_run ();
00238 void window_help ();
00239 void window_update_gradient ();
00240 void window_update ();
00241 void window_set_algorithm ();
00242 void window_set_experiment ();
00243 void window_remove_experiment ();
```

```
00244 void window_add_experiment ();
00245 void window_name_experiment ();
00246 void window_weight_experiment ();
00247 void window_inputs_experiment ();
00248 void window_template_experiment (void *data);
00249 void window_set_variable ();
00250 void window_remove_variable ();
00251 void window_add_variable ();
00252 void window_label_variable ();
00253 void window_precision_variable ();
00254 void window_rangemin_variable ();
00255 void window_rangemax_variable ();
00256 void window_rangeminabs_variable ();
00257 void window_rangemaxabs_variable ();
00258 void window_update_variable ();
00259 int window_read (char *filename);
00260 void window_open ();
00261 void window_new ();
00262 int cores_number ();
00263
00264 #endif
```

5.5 mpcotool.c File Reference

Source file of the mpcotool.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "mpcotool.h"
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "interface.h"
```

Include dependency graph for mpcotool.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG 0

  Macro to debug.
- #define ERROR_TYPE GTK_MESSAGE_ERROR

  Macro to define the error message type.
- #define INFO_TYPE GTK_MESSAGE_INFO

  Macro to define the information message type.
- #define INPUT_FILE "test-ga.xml"

  Macro to define the initial input file.
- #define RM "rm"

  Macro to define the shell remove command.

Functions

- void show_message (char *title, char *msg, int type)

  Function to show a dialog with a message.
- void show_error (char *msg)

  Function to show a dialog with an error message.
- int xml_node_get_int (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an integer number of a XML node property.
- unsigned int xml_node_get_uint (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an unsigned integer number of a XML node property.
- unsigned int xml_node_get_uint_with_default (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)

  Function to get an unsigned integer number of a XML node property with a default value.
- double xml_node_get_float (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get a floating point number of a XML node property.
- double xml_node_get_float_with_default (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)

  Function to get a floating point number of a XML node property with a default value.
- void xml_node_set_int (xmlNode *node, const xmlChar *prop, int value)

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

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

  Function to set a floating point number in a XML node property.
- void input_new ()

  Function to create a new Input struct.
- void input_free ()

  Function to free the memory of the input file data.
- int input_open (char *filename)

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

  Function to write the simulation input file.
- double calibrate_parse (unsigned int simulation, unsigned int experiment)

  Function to parse input files, simulating and calculating the \ objective function.
- void calibrate_print ()

  Function to print the results.
- void calibrate_save_variables (unsigned int simulation, double error)

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

  Function to save the best simulations.
- void calibrate_sequential ()

  Function to calibrate sequentially.
- void * calibrate_thread (ParallelData *data)

  Function to calibrate on a thread.
- void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 calibration results.
- void calibrate_synchronise ()

  Function to synchronise the calibration results of MPI tasks.
- void calibrate_sweep ()

  Function to calibrate with the sweep algorithm.
- void calibrate_MonteCarlo ()

Function to calibrate with the Monte-Carlo algorithm.

- void calibrate_best_gradient (unsigned int simulation, double value)

  Function to save the best simulation in a gradient based method.
- void calibrate_gradient_sequential (unsigned int simulation)

  Function to estimate the gradient sequentially.
- void * calibrate_gradient_thread (ParallelData *data)

  Function to estimate the gradient on a thread.
- double calibrate_estimate_gradient_random (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the gradient vector.
- double calibrate_estimate_gradient_coordinates (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the gradient vector.
- void calibrate_step_gradient (unsigned int simulation)

  Function to do a step of the gradient based method.
- void calibrate_gradient ()

  Function to calibrate with a gradient based method.
- double calibrate_genetic_objective (Entity *entity)

  Function to calculate the objective function of an entity.
- void calibrate_genetic ()

  Function to calibrate with the genetic algorithm.
- void calibrate_save_old ()

  Function to save the best results on iterative methods.
- void calibrate_merge_old ()

  Function to merge the best results with the previous step best results on iterative methods.
- void calibrate_refine ()

  Function to refine the search ranges of the variables in iterative algorithms.
- void calibrate_step ()

  Function to do a step of the iterative algorithm.
- void calibrate_iterate ()

  Function to iterate the algorithm.
- void calibrate_free ()

  Function to free the memory used by Calibrate struct.
- void calibrate_open ()

  Function to open and perform a calibration.
- void input_save_gradient (xmlNode *node)

  Function to save the gradient based method data in a XML node.
- void input_save (char *filename)

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

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

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

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

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

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

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

  Function to run a calibration.

- void window_help ()

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

  Function to show an about dialog.
- void window_update_gradient ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to update the variable rangemin in the main window.
- void window_rangemax_variable ()

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

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

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

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

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

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

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

Function to open the main window.

- int cores_number ()

  Function to obtain the cores number.

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

  Main function.

Variables

- int ntasks

  Number of tasks.

- unsigned int nthreads

  Number of threads.

- unsigned int nthreads_gradient

  Number of threads for the gradient based method.

- GMutex mutex [1]

  Mutex struct.

- void(* calibrate_algorithm)()

  Pointer to the function to perform a calibration algorithm step.

- double(* calibrate_estimate_gradient)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the gradient.

- Input input [1]

  Input struct to define the input file to mpcotool.

- Calibrate calibrate [1]

  Calibration data.

- const xmlChar * result_name = (xmlChar *) "result"

  Name of the result file.

- const xmlChar * variables_name = (xmlChar *) "variables"

  Name of the variables file.

- const xmlChar * template [MAX_NINPUTS]

  Array of xmlChar strings with template labels.

- const char * format [NPRECISIONS]

  Array of C-strings with variable formats.

- const double precision [NPRECISIONS]

  Array of variable precisions.

- const char * logo []

  Logo pixmap.

- Options options [1]

  Options struct to define the options dialog.

- Running running [1]

  Running struct to define the running dialog.

- Window window [1]

  Window struct to define the main interface window.

5.5.1 Detailed Description

Source file of the mpcotool.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file mpcotool.c.

5.5.2 Function Documentation

5.5.2.1 void calibrate_best (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1443 of file mpcotool.c.

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

5.5.2.2 void calibrate_best_gradient (unsigned int *simulation*, double *value*)

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

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1756 of file mpcotool.c.

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

5.5.2.3 double calibrate_estimate_gradient_coordinates (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the gradient vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1893 of file mpcotool.c.

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

5.5.2.4 double calibrate_estimate_gradient_random (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the gradient vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1866 of file mpcotool.c.

```
{
  double x;
#if DEBUG
  fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
#endif
```

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

5.5.2.5 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

Returns

objective function value.

Definition at line 2059 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.6 void calibrate_gradient_sequential (unsigned int *simulation*)

Function to estimate the gradient sequentially.

Parameters

simulation	Simulation number.

Definition at line 1786 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.7 void * calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1821 of file mpcotool.c.

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

```
  g_thread_exit (NULL);
  return NULL;
}
```

Here is the call graph for this function:

5.5.2.8 void calibrate_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

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

Definition at line 1196 of file mpcotool.c.

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

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

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

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

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

#if DEBUG
      fprintf (stderr, "calibrate_input: value=%s\n", value);
```

```
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

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

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

5.5.2.9 void calibrate_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 calibration results.

Parameters

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

Definition at line 1561 of file mpcotool.c.

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

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

5.5.2.10 double calibrate_parse (unsigned int *simulation*, unsigned int *experiment*)

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1283 of file mpcotool.c.

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

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

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

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

  // Checking the objective value function
  if (calibrate->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (calibrate->evaluator);
      buffer3 = g_path_get_basename (calibrate->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, calibrate->experiment[
```

```
      experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG
      fprintf (stderr, "calibrate_parse: %s\n", buffer);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

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

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

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

Here is the call graph for this function:

5.5.2.11 void calibrate_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1415 of file mpcotool.c.

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

5.5.2.12 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

simulation	Simulation number.

Definition at line 1923 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.13 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1517 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.14 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4875 of file mpcotool.c.

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

5.5.2.15 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 548 of file mpcotool.c.

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

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

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

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

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

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

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

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

  // Obtaining pseudo-random numbers generator seed
  input->seed
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, XML_EXPERIMENT))
        break;
#if DEBUG
      fprintf (stderr, "input_open: nexperiments=%u\n", input->
      nexperiments);
#endif
      if (xmlHasProp (child, XML_NAME))
        buffer = xmlGetProp (child, XML_NAME);
      else
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    gettext ("Experiment"),
                    input->nexperiments + 1, gettext ("no data
       file name"));
          msg = buffer2;
          goto exit_on_error;
        }
#if DEBUG
      fprintf (stderr, "input_open: experiment=%s\n", buffer);
#endif
      input->weight = g_realloc (input->weight,
                                 (1 + input->nexperiments) *
      sizeof (double));
```

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

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

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

```
        msg = buffer2;
        goto exit_on_error;
      }
  }
else
  {
    snprintf (buffer2, 64, "%s %s: %s",
              gettext ("Variable"), buffer, gettext ("no minimum range"))
;
    msg = buffer2;
    goto exit_on_error;
  }
if (xmlHasProp (child, XML_MAXIMUM))
  {
    input->rangemax = g_realloc
      (input->rangemax, (1 + input->nvariables
) * sizeof (double));
    input->rangemaxabs = g_realloc
      (input->rangemaxabs, (1 + input->nvariables
) * sizeof (double));
    input->rangemax[input->nvariables]
      = xml_node_get_float (child, XML_MAXIMUM
, &error_code);
    if (error_code)
      {
        snprintf (buffer2, 64, "%s %s: %s",
                  gettext ("Variable"), buffer, gettext ("bad maximum"));
        msg = buffer2;
        goto exit_on_error;
      }
    input->rangemaxabs[input->nvariables]
      = xml_node_get_float_with_default (
child, XML_ABSOLUTE_MAXIMUM,
                                       G_MAXDOUBLE, &error_code);
    if (error_code)
      {
        snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                  gettext ("bad absolute maximum"));
        msg = buffer2;
        goto exit_on_error;
      }
    if (input->rangemax[input->nvariables]
        > input->rangemaxabs[input->nvariables
])
      {
        snprintf (buffer2, 64, "%s %s: %s",
                  gettext ("Variable"),
                  buffer, gettext ("maximum range not allowed"));
        msg = buffer2;
        goto exit_on_error;
      }
  }
else
  {
    snprintf (buffer2, 64, "%s %s: %s",
              gettext ("Variable"), buffer, gettext ("no maximum range"))
;
    msg = buffer2;
    goto exit_on_error;
  }
if (input->rangemax[input->nvariables]
    < input->rangemin[input->nvariables])
  {
    snprintf (buffer2, 64, "%s %s: %s",
              gettext ("Variable"), buffer, gettext ("bad range"));
    msg = buffer2;
    goto exit_on_error;
  }
input->precision = g_realloc
  (input->precision, (1 + input->nvariables)
 * sizeof (unsigned int));
input->precision[input->nvariables]
  = xml_node_get_uint_with_default (child,
XML_PRECISION,
                                    DEFAULT_PRECISION, &
error_code);
if (error_code || input->precision[input->nvariables
] >= NPRECISIONS)
  {
    snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
              gettext ("bad precision"));
    msg = buffer2;
    goto exit_on_error;
  }
if (input->algorithm == ALGORITHM_SWEEP)
  {
    if (xmlHasProp (child, XML_NSWEEPS))
```

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

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

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

  // Closing the XML document
  xmlFreeDoc (doc);

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

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

Here is the call graph for this function:

5.5.2.16 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 2699 of file mpcotool.c.

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

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

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

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

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

  // Adding properties to the root XML node
  if (xmlStrcmp ((const xmlChar *) input->result, result_name
      ))
    xmlSetProp (node, XML_RESULT, (xmlChar *) input->result
      );
  if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
    xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
```

```
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, XML_SEED, input->
      seed);

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

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

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, XML_VARIABLE, 0);
      xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i
      ]);
      xml_node_set_float (child, XML_MINIMUM,
      input->rangemin[i]);
      if (input->rangeminabs[i] != -G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MINIMUM
      , input->rangeminabs[i]);
      xml_node_set_float (child, XML_MAXIMUM,
      input->rangemax[i]);
      if (input->rangemaxabs[i] != G_MAXDOUBLE)
        xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM
      , input->rangemaxabs[i]);
```

```
      if (input->precision[i] != DEFAULT_PRECISION
      )
        xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
      if (input->algorithm == ALGORITHM_SWEEP)
        xml_node_set_uint (child, XML_NSWEEPS,
      input->nsweeps[i]);
      else if (input->algorithm == ALGORITHM_GENETIC
      )
        xml_node_set_uint (child, XML_NBITS, input
      ->nbits[i]);
      if (input->nsteps)
        xml_node_set_float (child, XML_STEP, input
      ->step[i]);
    }

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

  // Freeing memory
  xmlFreeDoc (doc);

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

Here is the call graph for this function:

5.5.2.17 void input_save_gradient (xmlNode * *node*)

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

Parameters

node	XML node.

Definition at line 2667 of file mpcotool.c.

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

Here is the call graph for this function:

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

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

0 on success, >0 on error.

Definition at line 4896 of file mpcotool.c.

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

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

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

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

#if HAVE_GTK

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

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

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

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

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

#else

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

  // Getting threads number
  if (argn == 2)
    nthreads_gradient = nthreads = cores_number
       ();
  else
    {
      nthreads_gradient = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
```

```
          printf ("Bad threads number\n");
          return 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);

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

  // Freeing memory
  calibrate_free ();

#endif

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

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

  // Closing
  return 0;
}
```

Here is the call graph for this function:

5.5.2.19 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 256 of file mpcotool.c.

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file mpcotool.c.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

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

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

  // Showing the dialog and waiting response
```

```
  gtk_dialog_run (GTK_DIALOG (dlg));

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

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

5.5.2.21 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2932 of file mpcotool.c.

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

5.5.2.22 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2955 of file mpcotool.c.

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

5.5.2.23 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 4052 of file mpcotool.c.

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

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

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->
      result);
  gtk_entry_set_text (window->entry_variables, input
      ->variables);
  buffer = g_build_filename (input->directory, input->
      simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator
      ), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator
      ), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input
      ->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations
      ,
                                 (gdouble) input->nsimulations
      );
    case ALGORITHM_SWEEP:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
                                    input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_gradient
                                [input->gradient_method]),
      TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation
      ,
                                     (gdouble) input->relaxation
      );
          switch (input->gradient_method)
            {
            case GRADIENT_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates
      ,
                                         (gdouble) input->nestimates
```

```
      );
          }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations
      );
      gtk_spin_button_set_value (window->spin_generations
      ,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation,
      input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction
      ,
                                 input->reproduction_ratio
      );
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  g_signal_handler_block (window->combo_experiment,
      window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment
      );
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment
      ,
                                    input->experiment[i]);
  g_signal_handler_unblock
    (window->button_experiment, window->
      id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment
      ), 0);
  g_signal_handler_block (window->combo_variable, window
      ->id_variable);
  g_signal_handler_block (window->entry_variable, window
      ->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable,
       input->label[i]);
  g_signal_handler_unblock (window->entry_variable, window
      ->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window
      ->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable
      ), 0);
  window_set_variable ();
  window_update ();

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

Here is the call graph for this function:

5.5.2.24 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 3010 of file mpcotool.c.

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

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

```
#endif

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

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

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

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

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

```
            = gtk_spin_button_get_value (window->spin_reproduction
      );
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation
      );
          break;
        }

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

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

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

Here is the call graph for this function:

5.5.2.25 void window_template_experiment (void * *data*)

Function to update the experiment i-th input template in the main window.

Parameters

data	Callback data (i-th input template).

Definition at line 3656 of file mpcotool.c.

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

5.5.2.26 double xml_node_get_float (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

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

Parameters

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

Returns

Floating point number value.

Definition at line 366 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 400 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Returns

Integer number value.

Definition at line 274 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 305 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 339 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Definition at line 463 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 425 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 444 of file mpcotool.c.

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

5.5.3 Variable Documentation

5.5.3.1 const char* format[NPRECISIONS]

Initial value:

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

Array of C-strings with variable formats.

Definition at line 117 of file mpcotool.c.

5.5.3.2 const double precision[NPRECISIONS]

Initial value:

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

Array of variable precisions.

Definition at line 122 of file mpcotool.c.

5.5.3.3 const xmlChar* template[MAX_NINPUTS]

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 111 of file mpcotool.c.

5.6 mpcotool.c

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
```

```
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without
       modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright
       notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
       EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
       IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #define _GNU_SOURCE
00037 #include "config.h"
00038 #include <stdio.h>
00039 #include <stdlib.h>
00040 #include <string.h>
00041 #include <math.h>
00042 #include <unistd.h>
00043 #include <locale.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #ifdef G_OS_WIN32
00050 #include <windows.h>
00051 #elif (!__BSD_VISIBLE)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "mpcotool.h"
00059 #if HAVE_GTK
00060 #include <gio/gio.h>
00061 #include <gtk/gtk.h>
00062 #include "interface.h"
00063 #endif
00064
00065 #define DEBUG 0
00066
00067
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00092 int ntasks;
00093 unsigned int nthreads;
00094 unsigned int nthreads_gradient;
00096 GMutex mutex[1];
00097 void (*calibrate_algorithm) ();
00099 double (*calibrate_estimate_gradient) (unsigned int
      variable,
00100                                        unsigned int estimate);
00102 Input input[1];
00104 Calibrate calibrate[1];
```

```
00105
00106 const xmlChar *result_name = (xmlChar *) "result";
00108 const xmlChar *variables_name = (xmlChar *) "variables";
00110
00111 const xmlChar *template[MAX_NINPUTS] = {
00112   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3
      , XML_TEMPLATE4,
00113   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7
      , XML_TEMPLATE8
00114 };
00116
00117 const char *format[NPRECISIONS] = {
00118   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00119   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00120 };
00121
00122 const double precision[NPRECISIONS] = {
00123   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00124   1e-13, 1e-14
00125 };
00126
00127 const char *logo[] = {
00128   "32 32 3 1",
00129   "     c None",
00130   ".    c #0000FF",
00131   "+    c #FF0000",
00132   "                                ",
00133   "                                ",
00134   "                                ",
00135   "     .      .      .      .     ",
00136   "     .      .      .      .     ",
00137   "     .      .      .      .     ",
00138   "     .      .      .      .     ",
00139   "     .      .     +++     .     ",
00140   "     .      .    +++++    .     ",
00141   "     .      .    +++++    .     ",
00142   "     .      .    +++++    .     ",
00143   "    +++     .     +++    +++    ",
00144   "   +++++    .      .    +++++   ",
00145   "   +++++    .      .    +++++   ",
00146   "   +++++    .      .    +++++   ",
00147   "    +++     .      .     +++    ",
00148   "     .      .      .      .     ",
00149   "     .     +++     .      .     ",
00150   "     .    +++++    .      .     ",
00151   "     .    +++++    .      .     ",
00152   "     .    +++++    .      .     ",
00153   "     .     +++     .      .     ",
00154   "     .      .      .      .     ",
00155   "     .      .      .      .     ",
00156   "     .      .      .      .     ",
00157   "     .      .      .      .     ",
00158   "     .      .      .      .     ",
00159   "     .      .      .      .     ",
00160   "     .      .      .      .     ",
00161   "                                ",
00162   "                                ",
00163   "                                "
00164 };
00165
00166 /*
00167 const char * logo[] = {
00168 "32 32 3 1",
00169 "     c #FFFFFFFFFFFF",
00170 ".    c #00000000FFFF",
00171 "X    c #FFFF00000000",
00172 "                                ",
00173 "                                ",
00174 "                                ",
00175 "     .      .      .      .     ",
00176 "     .      .      .      .     ",
00177 "     .      .      .      .     ",
00178 "     .      .      .      .     ",
00179 "     .      .     XXX     .     ",
00180 "     .      .    XXXXX    .     ",
00181 "     .      .    XXXXX    .     ",
00182 "     .      .    XXXXX    .     ",
00183 "    XXX     .     XXX    XXX    ",
00184 "   XXXXX    .      .    XXXXX   ",
00185 "   XXXXX    .      .    XXXXX   ",
00186 "   XXXXX    .      .    XXXXX   ",
00187 "    XXX     .      .     XXX    ",
00188 "     .      .      .      .     ",
00189 "     .     XXX     .      .     ",
00190 "     .    XXXXX    .      .     ",
00191 "     .    XXXXX    .      .     ",
00192 "     .    XXXXX    .      .     ",
```

```
00193 "     .     XXX     .       .     ",
00194 "     .       .     .       .     ",
00195 "     .       .     .       .     ",
00196 "     .       .     .       .     ",
00197 "     .       .     .       .     ",
00198 "     .       .     .       .     ",
00199 "     .       .     .       .     ",
00200 "     .       .     .       .     ",
00201 "                                 ",
00202 "                                 ",
00203 "                                 "};
00204 */
00205
00206 #if HAVE_GTK
00207 Options options[1];
00209 Running running[1];
00211 Window window[1];
00213 #endif
00214
00225 void
00226 show_message (char *title, char *msg, int type)
00227 {
00228 #if HAVE_GTK
00229   GtkMessageDialog *dlg;
00230
00231   // Creating the dialog
00232   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00233     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00234
00235   // Setting the dialog title
00236   gtk_window_set_title (GTK_WINDOW (dlg), title);
00237
00238   // Showing the dialog and waiting response
00239   gtk_dialog_run (GTK_DIALOG (dlg));
00240
00241   // Closing and freeing memory
00242   gtk_widget_destroy (GTK_WIDGET (dlg));
00243
00244 #else
00245   printf ("%s: %s\n", title, msg);
00246 #endif
00247 }
00248
00255 void
00256 show_error (char *msg)
00257 {
00258   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00259 }
00260
00273 int
00274 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *
      error_code)
00275 {
00276   int i = 0;
00277   xmlChar *buffer;
00278   buffer = xmlGetProp (node, prop);
00279   if (!buffer)
00280     *error_code = 1;
00281   else
00282     {
00283       if (sscanf ((char *) buffer, "%d", &i) != 1)
00284         *error_code = 2;
00285       else
00286         *error_code = 0;
00287       xmlFree (buffer);
00288     }
00289   return i;
00290 }
00291
00304 unsigned int
00305 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *
      error_code)
00306 {
00307   unsigned int i = 0;
00308   xmlChar *buffer;
00309   buffer = xmlGetProp (node, prop);
00310   if (!buffer)
00311     *error_code = 1;
00312   else
00313     {
00314       if (sscanf ((char *) buffer, "%u", &i) != 1)
00315         *error_code = 2;
00316       else
00317         *error_code = 0;
00318       xmlFree (buffer);
00319     }
00320   return i;
```

```
00321 }
00322
00338 unsigned int
00339 xml_node_get_uint_with_default (xmlNode * node,
      const xmlChar * prop,
00340                                 unsigned int default_value, int *error_code)
00341 {
00342   unsigned int i;
00343   if (xmlHasProp (node, prop))
00344     i = xml_node_get_uint (node, prop, error_code);
00345   else
00346     {
00347       i = default_value;
00348       *error_code = 0;
00349     }
00350   return i;
00351 }
00352
00365 double
00366 xml_node_get_float (xmlNode * node, const xmlChar * prop, int
       *error_code)
00367 {
00368   double x = 0.;
00369   xmlChar *buffer;
00370   buffer = xmlGetProp (node, prop);
00371   if (!buffer)
00372     *error_code = 1;
00373   else
00374     {
00375       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00376         *error_code = 2;
00377       else
00378         *error_code = 0;
00379       xmlFree (buffer);
00380     }
00381   return x;
00382 }
00383
00399 double
00400 xml_node_get_float_with_default (xmlNode * node,
       const xmlChar * prop,
00401                                  double default_value, int *error_code)
00402 {
00403   double x;
00404   if (xmlHasProp (node, prop))
00405     x = xml_node_get_float (node, prop, error_code);
00406   else
00407     {
00408       x = default_value;
00409       *error_code = 0;
00410     }
00411   return x;
00412 }
00413
00424 void
00425 xml_node_set_int (xmlNode * node, const xmlChar * prop, int
      value)
00426 {
00427   xmlChar buffer[64];
00428   snprintf ((char *) buffer, 64, "%d", value);
00429   xmlSetProp (node, prop, buffer);
00430 }
00431
00443 void
00444 xml_node_set_uint (xmlNode * node, const xmlChar * prop,
      unsigned int value)
00445 {
00446   xmlChar buffer[64];
00447   snprintf ((char *) buffer, 64, "%u", value);
00448   xmlSetProp (node, prop, buffer);
00449 }
00450
00462 void
00463 xml_node_set_float (xmlNode * node, const xmlChar * prop,
      double value)
00464 {
00465   xmlChar buffer[64];
00466   snprintf ((char *) buffer, 64, "%.14lg", value);
00467   xmlSetProp (node, prop, buffer);
00468 }
00469
00474 void
00475 input_new ()
00476 {
00477   unsigned int i;
00478 #if DEBUG
00479   fprintf (stderr, "input_new: start\n");
```

```
00480 #endif
00481   input->nvariables = input->nexperiments = input->
      ninputs = input->nsteps = 0;
00482   input->simulator = input->evaluator = input->directory
       = input->name
00483     = input->result = input->variables = NULL;
00484   input->experiment = input->label = NULL;
00485   input->precision = input->nsweeps = input->nbits = NULL;
00486   input->rangemin = input->rangemax = input->rangeminabs
       = input->rangemaxabs
00487     = input->weight = input->step = NULL;
00488   for (i = 0; i < MAX_NINPUTS; ++i)
00489     input->template[i] = NULL;
00490 #if DEBUG
00491   fprintf (stderr, "input_new: end\n");
00492 #endif
00493 }
00494
00499 void
00500 input_free ()
00501 {
00502   unsigned int i, j;
00503 #if DEBUG
00504   fprintf (stderr, "input_free: start\n");
00505 #endif
00506   g_free (input->name);
00507   g_free (input->directory);
00508   for (i = 0; i < input->nexperiments; ++i)
00509     {
00510       xmlFree (input->experiment[i]);
00511       for (j = 0; j < input->ninputs; ++j)
00512         xmlFree (input->template[j][i]);
00513       g_free (input->template[j]);
00514     }
00515   g_free (input->experiment);
00516   for (i = 0; i < input->ninputs; ++i)
00517     g_free (input->template[i]);
00518   for (i = 0; i < input->nvariables; ++i)
00519     xmlFree (input->label[i]);
00520   g_free (input->label);
00521   g_free (input->precision);
00522   g_free (input->rangemin);
00523   g_free (input->rangemax);
00524   g_free (input->rangeminabs);
00525   g_free (input->rangemaxabs);
00526   g_free (input->weight);
00527   g_free (input->step);
00528   g_free (input->nsweeps);
00529   g_free (input->nbits);
00530   xmlFree (input->evaluator);
00531   xmlFree (input->simulator);
00532   xmlFree (input->result);
00533   xmlFree (input->variables);
00534   input->nexperiments = input->ninputs = input->nvariables
       = input->nsteps = 0;
00535 #if DEBUG
00536   fprintf (stderr, "input_free: end\n");
00537 #endif
00538 }
00539
00547 int
00548 input_open (char *filename)
00549 {
00550   char buffer2[64];
00551   char *buffert[MAX_NINPUTS] =
00552     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00553   xmlDoc *doc;
00554   xmlNode *node, *child;
00555   xmlChar *buffer;
00556   char *msg;
00557   int error_code;
00558   unsigned int i;
00559
00560 #if DEBUG
00561   fprintf (stderr, "input_open: start\n");
00562 #endif
00563
00564   // Resetting input data
00565   buffer = NULL;
00566   input_new ();
00567
00568   // Parsing the input file
00569 #if DEBUG
00570   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00571 #endif
00572   doc = xmlParseFile (filename);
00573   if (!doc)
```

```
00574     {
00575       msg = gettext ("Unable to parse the input file");
00576       goto exit_on_error;
00577     }
00578
00579   // Getting the root node
00580 #if DEBUG
00581   fprintf (stderr, "input_open: getting the root node\n");
00582 #endif
00583   node = xmlDocGetRootElement (doc);
00584   if (xmlStrcmp (node->name, XML_CALIBRATE))
00585     {
00586       msg = gettext ("Bad root XML node");
00587       goto exit_on_error;
00588     }
00589
00590   // Getting results file names
00591   input->result = (char *) xmlGetProp (node, XML_RESULT);
00592   if (!input->result)
00593     input->result = (char *) xmlStrdup (result_name);
00594   input->variables = (char *) xmlGetProp (node, XML_VARIABLES
      );
00595   if (!input->variables)
00596     input->variables = (char *) xmlStrdup (variables_name
      );
00597
00598   // Opening simulator program name
00599   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR
      );
00600   if (!input->simulator)
00601     {
00602       msg = gettext ("Bad simulator program");
00603       goto exit_on_error;
00604     }
00605
00606   // Opening evaluator program name
00607   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR
      );
00608
00609   // Obtaining pseudo-random numbers generator seed
00610   input->seed
00611     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00612                                       &error_code);
00613   if (error_code)
00614     {
00615       msg = gettext ("Bad pseudo-random numbers generator seed");
00616       goto exit_on_error;
00617     }
00618
00619   // Opening algorithm
00620   buffer = xmlGetProp (node, XML_ALGORITHM);
00621   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00622     {
00623       input->algorithm = ALGORITHM_MONTE_CARLO;
00624
00625       // Obtaining simulations number
00626       input->nsimulations
00627         = xml_node_get_int (node, XML_NSIMULATIONS
      , &error_code);
00628       if (error_code)
00629         {
00630           msg = gettext ("Bad simulations number");
00631           goto exit_on_error;
00632         }
00633     }
00634   else if (!xmlStrcmp (buffer, XML_SWEEP))
00635     input->algorithm = ALGORITHM_SWEEP;
00636   else if (!xmlStrcmp (buffer, XML_GENETIC))
00637     {
00638       input->algorithm = ALGORITHM_GENETIC;
00639
00640       // Obtaining population
00641       if (xmlHasProp (node, XML_NPOPULATION))
00642         {
00643           input->nsimulations
00644             = xml_node_get_uint (node, XML_NPOPULATION
      , &error_code);
00645           if (error_code || input->nsimulations < 3)
00646             {
00647               msg = gettext ("Invalid population number");
00648               goto exit_on_error;
00649             }
00650         }
00651       else
00652         {
00653           msg = gettext ("No population number");
```

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

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

```
00813                                                 DEFAULT_RELAXATION
      , &error_code);
00814           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00815             {
00816               msg = gettext ("Invalid relaxation parameter");
00817               goto exit_on_error;
00818             }
00819         }
00820       else
00821         input->nsteps = 0;
00822     }
00823
00824   // Reading the experimental data
00825   for (child = node->children; child; child = child->next)
00826     {
00827       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00828         break;
00829 #if DEBUG
00830       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments
      );
00831 #endif
00832       if (xmlHasProp (child, XML_NAME))
00833         buffer = xmlGetProp (child, XML_NAME);
00834       else
00835         {
00836           snprintf (buffer2, 64, "%s %u: %s",
00837                     gettext ("Experiment"),
00838                     input->nexperiments + 1, gettext ("no data file
       name"));
00839           msg = buffer2;
00840           goto exit_on_error;
00841         }
00842 #if DEBUG
00843       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00844 #endif
00845       input->weight = g_realloc (input->weight,
00846                                  (1 + input->nexperiments) * sizeof
       (double));
00847       input->weight[input->nexperiments]
00848         = xml_node_get_float_with_default (child
      , XML_WEIGHT, 1., &error_code);
00849       if (error_code)
00850         {
00851           snprintf (buffer2, 64, "%s %s: %s",
00852                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00853           msg = buffer2;
00854           goto exit_on_error;
00855         }
00856 #if DEBUG
00857       fprintf (stderr, "input_open: weight=%lg\n",
00858                input->weight[input->nexperiments]);
00859 #endif
00860       if (!input->nexperiments)
00861         input->ninputs = 0;
00862 #if DEBUG
00863       fprintf (stderr, "input_open: template[0]\n");
00864 #endif
00865       if (xmlHasProp (child, XML_TEMPLATE1))
00866         {
00867           input->template[0]
00868             = (char **) g_realloc (input->template[0],
00869                                    (1 + input->nexperiments) *
      sizeof (char *));
00870           buffert[0] = (char *) xmlGetProp (child, template[0]);
00871 #if DEBUG
00872           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00873                    input->nexperiments, buffert[0]);
00874 #endif
00875           if (!input->nexperiments)
00876             ++input->ninputs;
00877 #if DEBUG
00878           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00879 #endif
00880         }
00881       else
00882         {
00883           snprintf (buffer2, 64, "%s %s: %s",
00884                     gettext ("Experiment"), buffer, gettext ("no template"));
00885           msg = buffer2;
00886           goto exit_on_error;
00887         }
00888       for (i = 1; i < MAX_NINPUTS; ++i)
00889         {
00890 #if DEBUG
00891           fprintf (stderr, "input_open: template%u\n", i + 1);
00892 #endif
```

```
00893           if (xmlHasProp (child, template[i]))
00894             {
00895               if (input->nexperiments && input->ninputs <= i
      )
00896                 {
00897                   snprintf (buffer2, 64, "%s %s: %s",
00898                             gettext ("Experiment"),
00899                             buffer, gettext ("bad templates number"));
00900                   msg = buffer2;
00901                   while (i-- > 0)
00902                     xmlFree (buffert[i]);
00903                   goto exit_on_error;
00904                 }
00905               input->template[i] = (char **)
00906                 g_realloc (input->template[i],
00907                            (1 + input->nexperiments) * sizeof (char
       *));
00908               buffert[i] = (char *) xmlGetProp (child, template[i]);
00909 #if DEBUG
00910               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00911                        input->nexperiments, i + 1,
00912                        input->template[i][input->nexperiments
      ]);
00913 #endif
00914               if (!input->nexperiments)
00915                 ++input->ninputs;
00916 #if DEBUG
00917               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs
      );
00918 #endif
00919             }
00920           else if (input->nexperiments && input->ninputs > i
      )
00921             {
00922               snprintf (buffer2, 64, "%s %s: %s%u",
00923                         gettext ("Experiment"),
00924                         buffer, gettext ("no template"), i + 1);
00925               msg = buffer2;
00926               while (i-- > 0)
00927                 xmlFree (buffert[i]);
00928               goto exit_on_error;
00929             }
00930           else
00931             break;
00932         }
00933       input->experiment
00934         = g_realloc (input->experiment,
00935                      (1 + input->nexperiments) * sizeof (char *));
00936       input->experiment[input->nexperiments] = (char *)
      buffer;
00937       for (i = 0; i < input->ninputs; ++i)
00938         input->template[i][input->nexperiments] = buffert[i
      ];
00939       ++input->nexperiments;
00940 #if DEBUG
00941       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments
      );
00942 #endif
00943     }
00944   if (!input->nexperiments)
00945     {
00946       msg = gettext ("No calibration experiments");
00947       goto exit_on_error;
00948     }
00949   buffer = NULL;
00950
00951   // Reading the variables data
00952   for (; child; child = child->next)
00953     {
00954       if (xmlStrcmp (child->name, XML_VARIABLE))
00955         {
00956           snprintf (buffer2, 64, "%s %u: %s",
00957                     gettext ("Variable"),
00958                     input->nvariables + 1, gettext ("bad XML node"));
00959           msg = buffer2;
00960           goto exit_on_error;
00961         }
00962       if (xmlHasProp (child, XML_NAME))
00963         buffer = xmlGetProp (child, XML_NAME);
00964       else
00965         {
00966           snprintf (buffer2, 64, "%s %u: %s",
00967                     gettext ("Variable"),
00968                     input->nvariables + 1, gettext ("no name"));
00969           msg = buffer2;
00970           goto exit_on_error;
00971         }
```

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

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

```
01126             {
01127               snprintf (buffer2, 64, "%s %s: %s",
01128                         gettext ("Variable"),
01129                         buffer, gettext ("no bits number"));
01130               msg = buffer2;
01131               goto exit_on_error;
01132             }
01133         }
01134       else if (input->nsteps)
01135         {
01136           input->step = (double *)
01137             g_realloc (input->step, (1 + input->nvariables) *
      sizeof (double));
01138           input->step[input->nvariables]
01139             = xml_node_get_float (child, XML_STEP, &
      error_code);
01140           if (error_code || input->step[input->nvariables] < 0.)
01141             {
01142               snprintf (buffer2, 64, "%s %s: %s",
01143                         gettext ("Variable"),
01144                         buffer, gettext ("bad step size"));
01145               msg = buffer2;
01146               goto exit_on_error;
01147             }
01148         }
01149       input->label = g_realloc
01150         (input->label, (1 + input->nvariables) * sizeof (char *)
      );
01151       input->label[input->nvariables] = (char *) buffer;
01152       ++input->nvariables;
01153     }
01154   if (!input->nvariables)
01155     {
01156       msg = gettext ("No calibration variables");
01157       goto exit_on_error;
01158     }
01159   buffer = NULL;
01160
01161   // Getting the working directory
01162   input->directory = g_path_get_dirname (filename);
01163   input->name = g_path_get_basename (filename);
01164
01165   // Closing the XML document
01166   xmlFreeDoc (doc);
01167
01168 #if DEBUG
01169   fprintf (stderr, "input_open: end\n");
01170 #endif
01171   return 1;
01172
01173 exit_on_error:
01174   xmlFree (buffer);
01175   xmlFreeDoc (doc);
01176   show_error (msg);
01177   input_free ();
01178 #if DEBUG
01179   fprintf (stderr, "input_open: end\n");
01180 #endif
01181   return 0;
01182 }
01183
01195 void
01196 calibrate_input (unsigned int simulation, char *input,
      GMappedFile * template)
01197 {
01198   unsigned int i;
01199   char buffer[32], value[32], *buffer2, *buffer3, *content;
01200   FILE *file;
01201   gsize length;
01202   GRegex *regex;
01203
01204 #if DEBUG
01205   fprintf (stderr, "calibrate_input: start\n");
01206 #endif
01207
01208   // Checking the file
01209   if (!template)
01210     goto calibrate_input_end;
01211
01212   // Opening template
01213   content = g_mapped_file_get_contents (template);
01214   length = g_mapped_file_get_length (template);
01215 #if DEBUG
01216   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01217            content);
01218 #endif
01219   file = g_fopen (input, "w");
```

```
01220
01221   // Parsing template
01222   for (i = 0; i < calibrate->nvariables; ++i)
01223     {
01224 #if DEBUG
01225       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01226 #endif
01227       snprintf (buffer, 32, "@variable%u@", i + 1);
01228       regex = g_regex_new (buffer, 0, 0, NULL);
01229       if (i == 0)
01230         {
01231           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01232                                              calibrate->label[i], 0, NULL)
      ;
01233 #if DEBUG
01234           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01235 #endif
01236         }
01237       else
01238         {
01239           length = strlen (buffer3);
01240           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01241                                              calibrate->label[i], 0, NULL)
      ;
01242           g_free (buffer3);
01243         }
01244       g_regex_unref (regex);
01245       length = strlen (buffer2);
01246       snprintf (buffer, 32, "@value%u@", i + 1);
01247       regex = g_regex_new (buffer, 0, 0, NULL);
01248       snprintf (value, 32, format[calibrate->precision[i]],
01249                 calibrate->value[simulation * calibrate->nvariables
       + i]);
01250
01251 #if DEBUG
01252       fprintf (stderr, "calibrate_input: value=%s\n", value);
01253 #endif
01254       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01255                                          0, NULL);
01256       g_free (buffer2);
01257       g_regex_unref (regex);
01258     }
01259
01260   // Saving input file
01261   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01262   g_free (buffer3);
01263   fclose (file);
01264
01265 calibrate_input_end:
01266 #if DEBUG
01267   fprintf (stderr, "calibrate_input: end\n");
01268 #endif
01269   return;
01270 }
01271
01282 double
01283 calibrate_parse (unsigned int simulation, unsigned int
      experiment)
01284 {
01285   unsigned int i;
01286   double e;
01287   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32],
      *buffer2,
01288     *buffer3, *buffer4;
01289   FILE *file_result;
01290
01291 #if DEBUG
01292   fprintf (stderr, "calibrate_parse: start\n");
01293   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation
      ,
01294            experiment);
01295 #endif
01296
01297   // Opening input files
01298   for (i = 0; i < calibrate->ninputs; ++i)
01299     {
01300       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01301 #if DEBUG
01302       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01303 #endif
01304       calibrate_input (simulation, &input[i][0],
01305                        calibrate->file[i][experiment]);
01306     }
01307   for (; i < MAX_NINPUTS; ++i)
01308     strcpy (&input[i][0], "");
01309 #if DEBUG
01310   fprintf (stderr, "calibrate_parse: parsing end\n");
```

```
01311 #endif
01312
01313   // Performing the simulation
01314   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01315   buffer2 = g_path_get_dirname (calibrate->simulator);
01316   buffer3 = g_path_get_basename (calibrate->simulator);
01317   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01318   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01319             buffer4, input[0], input[1], input[2], input[3], input[4], input[5]
      ,
01320             input[6], input[7], output);
01321   g_free (buffer4);
01322   g_free (buffer3);
01323   g_free (buffer2);
01324 #if DEBUG
01325   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01326 #endif
01327   system (buffer);
01328
01329   // Checking the objective value function
01330   if (calibrate->evaluator)
01331     {
01332       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01333       buffer2 = g_path_get_dirname (calibrate->evaluator);
01334       buffer3 = g_path_get_basename (calibrate->evaluator);
01335       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01336       snprintf (buffer, 512, "\"%s\" %s %s %s",
01337                 buffer4, output, calibrate->experiment[experiment],
      result);
01338       g_free (buffer4);
01339       g_free (buffer3);
01340       g_free (buffer2);
01341 #if DEBUG
01342       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01343 #endif
01344       system (buffer);
01345       file_result = g_fopen (result, "r");
01346       e = atof (fgets (buffer, 512, file_result));
01347       fclose (file_result);
01348     }
01349   else
01350     {
01351       strcpy (result, "");
01352       file_result = g_fopen (output, "r");
01353       e = atof (fgets (buffer, 512, file_result));
01354       fclose (file_result);
01355     }
01356
01357   // Removing files
01358 #if !DEBUG
01359   for (i = 0; i < calibrate->ninputs; ++i)
01360     {
01361       if (calibrate->file[i][0])
01362         {
01363           snprintf (buffer, 512, RM " %s", &input[i][0]);
01364           system (buffer);
01365         }
01366     }
01367   snprintf (buffer, 512, RM " %s %s", output, result);
01368   system (buffer);
01369 #endif
01370
01371 #if DEBUG
01372   fprintf (stderr, "calibrate_parse: end\n");
01373 #endif
01374
01375   // Returning the objective function
01376   return e * calibrate->weight[experiment];
01377 }
01378
01383 void
01384 calibrate_print ()
01385 {
01386   unsigned int i;
01387   char buffer[512];
01388 #if HAVE_MPI
01389   if (calibrate->mpi_rank)
01390     return;
01391 #endif
01392   printf ("%s\n", gettext ("Best result"));
01393   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01394   printf ("error = %.15le\n", calibrate->error_old[0]);
01395   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
      error_old[0]);
01396   for (i = 0; i < calibrate->nvariables; ++i)
01397     {
01398       snprintf (buffer, 512, "%s = %s\n",
```

```
01399                 calibrate->label[i], format[calibrate->precision
      [i]]);
01400       printf (buffer, calibrate->value_old[i]);
01401       fprintf (calibrate->file_result, buffer, calibrate->value_old
      [i]);
01402     }
01403   fflush (calibrate->file_result);
01404 }
01405
01414 void
01415 calibrate_save_variables (unsigned int simulation,
      double error)
01416 {
01417   unsigned int i;
01418   char buffer[64];
01419 #if DEBUG
01420   fprintf (stderr, "calibrate_save_variables: start\n");
01421 #endif
01422   for (i = 0; i < calibrate->nvariables; ++i)
01423     {
01424       snprintf (buffer, 64, "%s ", format[calibrate->precision[i
      ]]);
01425       fprintf (calibrate->file_variables, buffer,
01426                calibrate->value[simulation * calibrate->nvariables
       + i]);
01427     }
01428   fprintf (calibrate->file_variables, "%.14le\n", error);
01429 #if DEBUG
01430   fprintf (stderr, "calibrate_save_variables: end\n");
01431 #endif
01432 }
01433
01442 void
01443 calibrate_best (unsigned int simulation, double value)
01444 {
01445   unsigned int i, j;
01446   double e;
01447 #if DEBUG
01448   fprintf (stderr, "calibrate_best: start\n");
01449   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01450            calibrate->nsaveds, calibrate->nbest);
01451 #endif
01452   if (calibrate->nsaveds < calibrate->nbest
01453       || value < calibrate->error_best[calibrate->nsaveds - 1])
01454     {
01455       if (calibrate->nsaveds < calibrate->nbest)
01456         ++calibrate->nsaveds;
01457       calibrate->error_best[calibrate->nsaveds - 1] = value;
01458       calibrate->simulation_best[calibrate->nsaveds - 1]
      = simulation;
01459       for (i = calibrate->nsaveds; --i;)
01460         {
01461           if (calibrate->error_best[i] < calibrate->error_best
      [i - 1])
01462             {
01463               j = calibrate->simulation_best[i];
01464               e = calibrate->error_best[i];
01465               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01466               calibrate->error_best[i] = calibrate->error_best
      [i - 1];
01467               calibrate->simulation_best[i - 1] = j;
01468               calibrate->error_best[i - 1] = e;
01469             }
01470           else
01471             break;
01472         }
01473     }
01474 #if DEBUG
01475   fprintf (stderr, "calibrate_best: end\n");
01476 #endif
01477 }
01478
01483 void
01484 calibrate_sequential ()
01485 {
01486   unsigned int i, j;
01487   double e;
01488 #if DEBUG
01489   fprintf (stderr, "calibrate_sequential: start\n");
01490   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01491            calibrate->nstart, calibrate->nend);
01492 #endif
01493   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01494     {
01495       e = 0.;
01496       for (j = 0; j < calibrate->nexperiments; ++j)
```

```
01497         e += calibrate_parse (i, j);
01498       calibrate_best (i, e);
01499       calibrate_save_variables (i, e);
01500 #if DEBUG
01501       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01502 #endif
01503     }
01504 #if DEBUG
01505   fprintf (stderr, "calibrate_sequential: end\n");
01506 #endif
01507 }
01508
01516 void *
01517 calibrate_thread (ParallelData * data)
01518 {
01519   unsigned int i, j, thread;
01520   double e;
01521 #if DEBUG
01522   fprintf (stderr, "calibrate_thread: start\n");
01523 #endif
01524   thread = data->thread;
01525 #if DEBUG
01526   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01527            calibrate->thread[thread], calibrate->thread[thread + 1]
      );
01528 #endif
01529   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1];
      ++i)
01530     {
01531       e = 0.;
01532       for (j = 0; j < calibrate->nexperiments; ++j)
01533         e += calibrate_parse (i, j);
01534       g_mutex_lock (mutex);
01535       calibrate_best (i, e);
01536       calibrate_save_variables (i, e);
01537       g_mutex_unlock (mutex);
01538 #if DEBUG
01539       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01540 #endif
01541     }
01542 #if DEBUG
01543   fprintf (stderr, "calibrate_thread: end\n");
01544 #endif
01545   g_thread_exit (NULL);
01546   return NULL;
01547 }
01548
01560 void
01561 calibrate_merge (unsigned int nsaveds, unsigned int *
      simulation_best,
01562                  double *error_best)
01563 {
01564   unsigned int i, j, k, s[calibrate->nbest];
01565   double e[calibrate->nbest];
01566 #if DEBUG
01567   fprintf (stderr, "calibrate_merge: start\n");
01568 #endif
01569   i = j = k = 0;
01570   do
01571     {
01572       if (i == calibrate->nsaveds)
01573         {
01574           s[k] = simulation_best[j];
01575           e[k] = error_best[j];
01576           ++j;
01577           ++k;
01578           if (j == nsaveds)
01579             break;
01580         }
01581       else if (j == nsaveds)
01582         {
01583           s[k] = calibrate->simulation_best[i];
01584           e[k] = calibrate->error_best[i];
01585           ++i;
01586           ++k;
01587           if (i == calibrate->nsaveds)
01588             break;
01589         }
01590       else if (calibrate->error_best[i] > error_best[j])
01591         {
01592           s[k] = simulation_best[j];
01593           e[k] = error_best[j];
01594           ++j;
01595           ++k;
01596         }
01597       else
01598         {
```

```
01599           s[k] = calibrate->simulation_best[i];
01600           e[k] = calibrate->error_best[i];
01601           ++i;
01602           ++k;
01603         }
01604     }
01605   while (k < calibrate->nbest);
01606   calibrate->nsaveds = k;
01607   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned
      int));
01608   memcpy (calibrate->error_best, e, k * sizeof (double));
01609 #if DEBUG
01610   fprintf (stderr, "calibrate_merge: end\n");
01611 #endif
01612 }
01613
01618 #if HAVE_MPI
01619 void
01620 calibrate_synchronise ()
01621 {
01622   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01623   double error_best[calibrate->nbest];
01624   MPI_Status mpi_stat;
01625 #if DEBUG
01626   fprintf (stderr, "calibrate_synchronise: start\n");
01627 #endif
01628   if (calibrate->mpi_rank == 0)
01629     {
01630       for (i = 1; i < ntasks; ++i)
01631         {
01632           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01633           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01634                     MPI_COMM_WORLD, &mpi_stat);
01635           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01636                     MPI_COMM_WORLD, &mpi_stat);
01637           calibrate_merge (nsaveds, simulation_best, error_best)
      ;
01638         }
01639     }
01640   else
01641     {
01642       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01643       MPI_Send (calibrate->simulation_best, calibrate->nsaveds
      , MPI_INT, 0, 1,
01644                 MPI_COMM_WORLD);
01645       MPI_Send (calibrate->error_best, calibrate->nsaveds,
      MPI_DOUBLE, 0, 1,
01646                 MPI_COMM_WORLD);
01647     }
01648 #if DEBUG
01649   fprintf (stderr, "calibrate_synchronise: end\n");
01650 #endif
01651 }
01652 #endif
01653
01658 void
01659 calibrate_sweep ()
01660 {
01661   unsigned int i, j, k, l;
01662   double e;
01663   GThread *thread[nthreads];
01664   ParallelData data[nthreads];
01665 #if DEBUG
01666   fprintf (stderr, "calibrate_sweep: start\n");
01667 #endif
01668   for (i = 0; i < calibrate->nsimulations; ++i)
01669     {
01670       k = i;
01671       for (j = 0; j < calibrate->nvariables; ++j)
01672         {
01673           l = k % calibrate->nsweeps[j];
01674           k /= calibrate->nsweeps[j];
01675           e = calibrate->rangemin[j];
01676           if (calibrate->nsweeps[j] > 1)
01677             e += l * (calibrate->rangemax[j] - calibrate->rangemin
      [j])
01678                / (calibrate->nsweeps[j] - 1);
01679           calibrate->value[i * calibrate->nvariables + j] = e;
01680         }
01681     }
01682   calibrate->nsaveds = 0;
01683   if (nthreads <= 1)
01684     calibrate_sequential ();
01685   else
01686     {
01687       for (i = 0; i < nthreads; ++i)
01688         {
```

```
01689           data[i].thread = i;
01690           thread[i]
01691             = g_thread_new (NULL, (void (*)) calibrate_thread,
      &data[i]);
01692         }
01693       for (i = 0; i < nthreads; ++i)
01694         g_thread_join (thread[i]);
01695     }
01696 #if HAVE_MPI
01697   // Communicating tasks results
01698   calibrate_synchronise ();
01699 #endif
01700 #if DEBUG
01701   fprintf (stderr, "calibrate_sweep: end\n");
01702 #endif
01703 }
01704
01709 void
01710 calibrate_MonteCarlo ()
01711 {
01712   unsigned int i, j;
01713   GThread *thread[nthreads];
01714   ParallelData data[nthreads];
01715 #if DEBUG
01716   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01717 #endif
01718   for (i = 0; i < calibrate->nsimulations; ++i)
01719     for (j = 0; j < calibrate->nvariables; ++j)
01720       calibrate->value[i * calibrate->nvariables + j]
01721         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01722         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01723   calibrate->nsaveds = 0;
01724   if (nthreads <= 1)
01725     calibrate_sequential ();
01726   else
01727     {
01728       for (i = 0; i < nthreads; ++i)
01729         {
01730           data[i].thread = i;
01731           thread[i]
01732             = g_thread_new (NULL, (void (*)) calibrate_thread,
      &data[i]);
01733         }
01734       for (i = 0; i < nthreads; ++i)
01735         g_thread_join (thread[i]);
01736     }
01737 #if HAVE_MPI
01738   // Communicating tasks results
01739   calibrate_synchronise ();
01740 #endif
01741 #if DEBUG
01742   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01743 #endif
01744 }
01745
01755 void
01756 calibrate_best_gradient (unsigned int simulation, double
       value)
01757 {
01758 #if DEBUG
01759   fprintf (stderr, "calibrate_best_gradient: start\n");
01760   fprintf (stderr,
01761            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01762            simulation, value, calibrate->error_best[0]);
01763 #endif
01764   if (value < calibrate->error_best[0])
01765     {
01766       calibrate->error_best[0] = value;
01767       calibrate->simulation_best[0] = simulation;
01768 #if DEBUG
01769       fprintf (stderr,
01770                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01771                simulation, value);
01772 #endif
01773     }
01774 #if DEBUG
01775   fprintf (stderr, "calibrate_best_gradient: end\n");
01776 #endif
01777 }
01778
01785 void
01786 calibrate_gradient_sequential (unsigned int
      simulation)
01787 {
01788   unsigned int i, j, k;
01789   double e;
01790 #if DEBUG
```

```
01791   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01792   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01793            "nend_gradient=%u\n",
01794            calibrate->nstart_gradient, calibrate->nend_gradient
      );
01795 #endif
01796   for (i = calibrate->nstart_gradient; i < calibrate->
      nend_gradient; ++i)
01797     {
01798       k = simulation + i;
01799       e = 0.;
01800       for (j = 0; j < calibrate->nexperiments; ++j)
01801         e += calibrate_parse (k, j);
01802       calibrate_best_gradient (k, e);
01803       calibrate_save_variables (k, e);
01804 #if DEBUG
01805       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01806 #endif
01807     }
01808 #if DEBUG
01809   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01810 #endif
01811 }
01812
01820 void *
01821 calibrate_gradient_thread (ParallelData *
      data)
01822 {
01823   unsigned int i, j, thread;
01824   double e;
01825 #if DEBUG
01826   fprintf (stderr, "calibrate_gradient_thread: start\n");
01827 #endif
01828   thread = data->thread;
01829 #if DEBUG
01830   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01831            thread,
01832            calibrate->thread_gradient[thread],
01833            calibrate->thread_gradient[thread + 1]);
01834 #endif
01835   for (i = calibrate->thread_gradient[thread];
01836        i < calibrate->thread_gradient[thread + 1]; ++i)
01837     {
01838       e = 0.;
01839       for (j = 0; j < calibrate->nexperiments; ++j)
01840         e += calibrate_parse (i, j);
01841       g_mutex_lock (mutex);
01842       calibrate_best_gradient (i, e);
01843       calibrate_save_variables (i, e);
01844       g_mutex_unlock (mutex);
01845 #if DEBUG
01846       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01847 #endif
01848     }
01849 #if DEBUG
01850   fprintf (stderr, "calibrate_gradient_thread: end\n");
01851 #endif
01852   g_thread_exit (NULL);
01853   return NULL;
01854 }
01855
01865 double
01866 calibrate_estimate_gradient_random (unsigned
      int variable,
01867                                     unsigned int estimate)
01868 {
01869   double x;
01870 #if DEBUG
01871   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01872 #endif
01873   x = calibrate->gradient[variable]
01874     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[
      variable];
01875 #if DEBUG
01876   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01877            variable, x);
01878   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01879 #endif
01880   return x;
01881 }
01882
01892 double
01893 calibrate_estimate_gradient_coordinates
      (unsigned int variable,
01894                                          unsigned int estimate)
01895 {
01896   double x;
```

```
01897 #if DEBUG
01898   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01899 #endif
01900   x = calibrate->gradient[variable];
01901   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01902     {
01903       if (estimate & 1)
01904         x += calibrate->step[variable];
01905       else
01906         x -= calibrate->step[variable];
01907     }
01908 #if DEBUG
01909   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01910            variable, x);
01911   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01912 #endif
01913   return x;
01914 }
01915
01922 void
01923 calibrate_step_gradient (unsigned int simulation)
01924 {
01925   GThread *thread[nthreads_gradient];
01926   ParallelData data[nthreads_gradient];
01927   unsigned int i, j, k, b;
01928 #if DEBUG
01929   fprintf (stderr, "calibrate_step_gradient: start\n");
01930 #endif
01931   for (i = 0; i < calibrate->nestimates; ++i)
01932     {
01933       k = (simulation + i) * calibrate->nvariables;
01934       b = calibrate->simulation_best[0] * calibrate->nvariables
      ;
01935 #if DEBUG
01936       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01937                simulation + i, calibrate->simulation_best[0]);
01938 #endif
01939       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01940         {
01941 #if DEBUG
01942           fprintf (stderr,
01943                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01944                    i, j, calibrate->value[b]);
01945 #endif
01946           calibrate->value[k]
01947             = calibrate->value[b] + calibrate_estimate_gradient
       (j, i);
01948           calibrate->value[k] = fmin (fmax (calibrate->value[k],
01949                                             calibrate->rangeminabs[j
      ]),
01950                                       calibrate->rangemaxabs[j]);
01951 #if DEBUG
01952           fprintf (stderr,
01953                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01954                    i, j, calibrate->value[k]);
01955 #endif
01956         }
01957     }
01958   if (nthreads_gradient == 1)
01959     calibrate_gradient_sequential (simulation);
01960   else
01961     {
01962       for (i = 0; i <= nthreads_gradient; ++i)
01963         {
01964           calibrate->thread_gradient[i]
01965             = simulation + calibrate->nstart_gradient
01966             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01967             / nthreads_gradient;
01968 #if DEBUG
01969           fprintf (stderr,
01970                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01971                    i, calibrate->thread_gradient[i]);
01972 #endif
01973         }
01974       for (i = 0; i < nthreads_gradient; ++i)
01975         {
01976           data[i].thread = i;
01977           thread[i] = g_thread_new
01978             (NULL, (void (*)) calibrate_gradient_thread
      , &data[i]);
01979         }
01980       for (i = 0; i < nthreads_gradient; ++i)
01981         g_thread_join (thread[i]);
01982     }
01983 #if DEBUG
01984   fprintf (stderr, "calibrate_step_gradient: end\n");
```

```
01985 #endif
01986 }
01987
01992 void
01993 calibrate_gradient ()
01994 {
01995   unsigned int i, j, k, b, s, adjust;
01996 #if DEBUG
01997   fprintf (stderr, "calibrate_gradient: start\n");
01998 #endif
01999   for (i = 0; i < calibrate->nvariables; ++i)
02000     calibrate->gradient[i] = 0.;
02001   b = calibrate->simulation_best[0] * calibrate->nvariables
      ;
02002   s = calibrate->nsimulations;
02003   adjust = 1;
02004   for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates
      , b = k)
02005     {
02006 #if DEBUG
02007       fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
02008                i, calibrate->simulation_best[0]);
02009 #endif
02010       calibrate_step_gradient (s);
02011       k = calibrate->simulation_best[0] * calibrate->nvariables
      ;
02012 #if DEBUG
02013       fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
02014                i, calibrate->simulation_best[0]);
02015 #endif
02016       if (k == b)
02017         {
02018           if (adjust)
02019             for (j = 0; j < calibrate->nvariables; ++j)
02020               calibrate->step[j] *= 0.5;
02021           for (j = 0; j < calibrate->nvariables; ++j)
02022             calibrate->gradient[j] = 0.;
02023           adjust = 1;
02024         }
02025       else
02026         {
02027           for (j = 0; j < calibrate->nvariables; ++j)
02028             {
02029 #if DEBUG
02030               fprintf (stderr,
02031                        "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02032                        j, calibrate->value[k + j], j, calibrate->value
      [b + j]);
02033 #endif
02034               calibrate->gradient[j]
02035                 = (1. - calibrate->relaxation) * calibrate->gradient
      [j]
02036                 + calibrate->relaxation
02037                 * (calibrate->value[k + j] - calibrate->value[b + j])
      ;
02038 #if DEBUG
02039               fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02040                        j, calibrate->gradient[j]);
02041 #endif
02042             }
02043           adjust = 0;
02044         }
02045     }
02046 #if DEBUG
02047   fprintf (stderr, "calibrate_gradient: end\n");
02048 #endif
02049 }
02050
02058 double
02059 calibrate_genetic_objective (Entity * entity)
02060 {
02061   unsigned int j;
02062   double objective;
02063   char buffer[64];
02064 #if DEBUG
02065   fprintf (stderr, "calibrate_genetic_objective: start\n");
02066 #endif
02067   for (j = 0; j < calibrate->nvariables; ++j)
02068     {
02069       calibrate->value[entity->id * calibrate->nvariables + j]
02070         = genetic_get_variable (entity, calibrate->genetic_variable
       + j);
02071     }
02072   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02073     objective += calibrate_parse (entity->id, j);
02074   g_mutex_lock (mutex);
02075   for (j = 0; j < calibrate->nvariables; ++j)
```

```
02076     {
02077       snprintf (buffer, 64, "%s ", format[calibrate->precision[j
    ]]);
02078       fprintf (calibrate->file_variables, buffer,
02079                genetic_get_variable (entity, calibrate->genetic_variable
     + j));
02080     }
02081   fprintf (calibrate->file_variables, "%.14le\n", objective);
02082   g_mutex_unlock (mutex);
02083 #if DEBUG
02084   fprintf (stderr, "calibrate_genetic_objective: end\n");
02085 #endif
02086   return objective;
02087 }
02088
02093 void
02094 calibrate_genetic ()
02095 {
02096   char *best_genome;
02097   double best_objective, *best_variable;
02098 #if DEBUG
02099   fprintf (stderr, "calibrate_genetic: start\n");
02100   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02101            nthreads);
02102   fprintf (stderr,
02103            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02104            calibrate->nvariables, calibrate->nsimulations
    ,
02105            calibrate->niterations);
02106   fprintf (stderr,
02107            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02108            calibrate->mutation_ratio, calibrate->
    reproduction_ratio,
02109            calibrate->adaptation_ratio);
02110 #endif
02111   genetic_algorithm_default (calibrate->nvariables,
02112                              calibrate->genetic_variable,
02113                              calibrate->nsimulations,
02114                              calibrate->niterations,
02115                              calibrate->mutation_ratio,
02116                              calibrate->reproduction_ratio,
02117                              calibrate->adaptation_ratio,
02118                              &calibrate_genetic_objective
    ,
02119                              &best_genome, &best_variable, &best_objective);
02120 #if DEBUG
02121   fprintf (stderr, "calibrate_genetic: the best\n");
02122 #endif
02123   calibrate->error_old = (double *) g_malloc (sizeof (double));
02124   calibrate->value_old
02125     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02126   calibrate->error_old[0] = best_objective;
02127   memcpy (calibrate->value_old, best_variable,
02128           calibrate->nvariables * sizeof (double));
02129   g_free (best_genome);
02130   g_free (best_variable);
02131   calibrate_print ();
02132 #if DEBUG
02133   fprintf (stderr, "calibrate_genetic: end\n");
02134 #endif
02135 }
02136
02141 void
02142 calibrate_save_old ()
02143 {
02144   unsigned int i, j;
02145 #if DEBUG
02146   fprintf (stderr, "calibrate_save_old: start\n");
02147   fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds
    );
02148 #endif
02149   memcpy (calibrate->error_old, calibrate->error_best,
02150           calibrate->nbest * sizeof (double));
02151   for (i = 0; i < calibrate->nbest; ++i)
02152     {
02153       j = calibrate->simulation_best[i];
02154 #if DEBUG
02155       fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02156 #endif
02157       memcpy (calibrate->value_old + i * calibrate->nvariables
    ,
02158               calibrate->value + j * calibrate->nvariables,
02159               calibrate->nvariables * sizeof (double));
02160     }
02161 #if DEBUG
02162   for (i = 0; i < calibrate->nvariables; ++i)
02163     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
```

```
02164             i, calibrate->value_old[i]);
02165   fprintf (stderr, "calibrate_save_old: end\n");
02166 #endif
02167 }
02168
02174 void
02175 calibrate_merge_old ()
02176 {
02177   unsigned int i, j, k;
02178   double v[calibrate->nbest * calibrate->nvariables], e[
      calibrate->nbest],
02179     *enew, *eold;
02180 #if DEBUG
02181   fprintf (stderr, "calibrate_merge_old: start\n");
02182 #endif
02183   enew = calibrate->error_best;
02184   eold = calibrate->error_old;
02185   i = j = k = 0;
02186   do
02187     {
02188       if (*enew < *eold)
02189         {
02190           memcpy (v + k * calibrate->nvariables,
02191                   calibrate->value
02192                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
02193                   calibrate->nvariables * sizeof (double));
02194           e[k] = *enew;
02195           ++k;
02196           ++enew;
02197           ++i;
02198         }
02199       else
02200         {
02201           memcpy (v + k * calibrate->nvariables,
02202                   calibrate->value_old + j * calibrate->nvariables
      ,
02203                   calibrate->nvariables * sizeof (double));
02204           e[k] = *eold;
02205           ++k;
02206           ++eold;
02207           ++j;
02208         }
02209     }
02210   while (k < calibrate->nbest);
02211   memcpy (calibrate->value_old, v, k * calibrate->nvariables
       * sizeof (double));
02212   memcpy (calibrate->error_old, e, k * sizeof (double));
02213 #if DEBUG
02214   fprintf (stderr, "calibrate_merge_old: end\n");
02215 #endif
02216 }
02217
02223 void
02224 calibrate_refine ()
02225 {
02226   unsigned int i, j;
02227   double d;
02228 #if HAVE_MPI
02229   MPI_Status mpi_stat;
02230 #endif
02231 #if DEBUG
02232   fprintf (stderr, "calibrate_refine: start\n");
02233 #endif
02234 #if HAVE_MPI
02235   if (!calibrate->mpi_rank)
02236     {
02237 #endif
02238       for (j = 0; j < calibrate->nvariables; ++j)
02239         {
02240           calibrate->rangemin[j] = calibrate->rangemax[j]
02241             = calibrate->value_old[j];
02242         }
02243       for (i = 0; ++i < calibrate->nbest;)
02244         {
02245           for (j = 0; j < calibrate->nvariables; ++j)
02246             {
02247               calibrate->rangemin[j]
02248                 = fmin (calibrate->rangemin[j],
02249                         calibrate->value_old[i * calibrate->nvariables
       + j]);
02250               calibrate->rangemax[j]
02251                 = fmax (calibrate->rangemax[j],
02252                         calibrate->value_old[i * calibrate->nvariables
       + j]);
02253             }
02254         }
```

```
02255       for (j = 0; j < calibrate->nvariables; ++j)
02256         {
02257           d = calibrate->tolerance
02258             * (calibrate->rangemax[j] - calibrate->rangemin[j])
    ;
02259           switch (calibrate->algorithm)
02260             {
02261             case ALGORITHM_MONTE_CARLO:
02262               d *= 0.5;
02263               break;
02264             default:
02265               if (calibrate->nsweeps[j] > 1)
02266                 d /= calibrate->nsweeps[j] - 1;
02267               else
02268                 d = 0.;
02269             }
02270           calibrate->rangemin[j] -= d;
02271           calibrate->rangemin[j]
02272             = fmax (calibrate->rangemin[j], calibrate->rangeminabs
    [j]);
02273           calibrate->rangemax[j] += d;
02274           calibrate->rangemax[j]
02275             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs
    [j]);
02276           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02277                   calibrate->rangemin[j], calibrate->rangemax[j
    ]);
02278           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02279                    calibrate->label[j], calibrate->rangemin[j],
02280                    calibrate->rangemax[j]);
02281         }
02282 #if HAVE_MPI
02283       for (i = 1; i < ntasks; ++i)
02284         {
02285           MPI_Send (calibrate->rangemin, calibrate->nvariables
    , MPI_DOUBLE, i,
02286                     1, MPI_COMM_WORLD);
02287           MPI_Send (calibrate->rangemax, calibrate->nvariables
    , MPI_DOUBLE, i,
02288                     1, MPI_COMM_WORLD);
02289         }
02290     }
02291   else
02292     {
02293       MPI_Recv (calibrate->rangemin, calibrate->nvariables,
    MPI_DOUBLE, 0, 1,
02294                 MPI_COMM_WORLD, &mpi_stat);
02295       MPI_Recv (calibrate->rangemax, calibrate->nvariables,
    MPI_DOUBLE, 0, 1,
02296                 MPI_COMM_WORLD, &mpi_stat);
02297     }
02298 #endif
02299 #if DEBUG
02300   fprintf (stderr, "calibrate_refine: end\n");
02301 #endif
02302 }
02303
02308 void
02309 calibrate_step ()
02310 {
02311 #if DEBUG
02312   fprintf (stderr, "calibrate_step: start\n");
02313 #endif
02314   calibrate_algorithm ();
02315   if (calibrate->nsteps)
02316     calibrate_gradient ();
02317 #if DEBUG
02318   fprintf (stderr, "calibrate_step: end\n");
02319 #endif
02320 }
02321
02326 void
02327 calibrate_iterate ()
02328 {
02329   unsigned int i;
02330 #if DEBUG
02331   fprintf (stderr, "calibrate_iterate: start\n");
02332 #endif
02333   calibrate->error_old
02334     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02335   calibrate->value_old = (double *)
02336     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof
    (double));
02337   calibrate_step ();
02338   calibrate_save_old ();
02339   calibrate_refine ();
02340   calibrate_print ();
```

```
02341   for (i = 1; i < calibrate->niterations; ++i)
02342     {
02343       calibrate_step ();
02344       calibrate_merge_old ();
02345       calibrate_refine ();
02346       calibrate_print ();
02347     }
02348 #if DEBUG
02349   fprintf (stderr, "calibrate_iterate: end\n");
02350 #endif
02351 }
02352
02357 void
02358 calibrate_free ()
02359 {
02360   unsigned int i, j;
02361 #if DEBUG
02362   fprintf (stderr, "calibrate_free: start\n");
02363 #endif
02364   for (j = 0; j < calibrate->ninputs; ++j)
02365     {
02366       for (i = 0; i < calibrate->nexperiments; ++i)
02367         g_mapped_file_unref (calibrate->file[j][i]);
02368       g_free (calibrate->file[j]);
02369     }
02370   g_free (calibrate->error_old);
02371   g_free (calibrate->value_old);
02372   g_free (calibrate->value);
02373   g_free (calibrate->genetic_variable);
02374   g_free (calibrate->rangemax);
02375   g_free (calibrate->rangemin);
02376 #if DEBUG
02377   fprintf (stderr, "calibrate_free: end\n");
02378 #endif
02379 }
02380
02385 void
02386 calibrate_open ()
02387 {
02388   GTimeZone *tz;
02389   GDateTime *t0, *t;
02390   unsigned int i, j, *nbits;
02391
02392 #if DEBUG
02393   char *buffer;
02394   fprintf (stderr, "calibrate_open: start\n");
02395 #endif
02396
02397   // Getting initial time
02398 #if DEBUG
02399   fprintf (stderr, "calibrate_open: getting initial time\n");
02400 #endif
02401   tz = g_time_zone_new_utc ();
02402   t0 = g_date_time_new_now (tz);
02403
02404   // Obtaining and initing the pseudo-random numbers generator seed
02405 #if DEBUG
02406   fprintf (stderr, "calibrate_open: getting initial seed\n");
02407 #endif
02408   calibrate->seed = input->seed;
02409   gsl_rng_set (calibrate->rng, calibrate->seed);
02410
02411   // Replacing the working directory
02412 #if DEBUG
02413   fprintf (stderr, "calibrate_open: replacing the working directory\n");
02414 #endif
02415   g_chdir (input->directory);
02416
02417   // Getting results file names
02418   calibrate->result = input->result;
02419   calibrate->variables = input->variables;
02420
02421   // Obtaining the simulator file
02422   calibrate->simulator = input->simulator;
02423
02424   // Obtaining the evaluator file
02425   calibrate->evaluator = input->evaluator;
02426
02427   // Reading the algorithm
02428   calibrate->algorithm = input->algorithm;
02429   switch (calibrate->algorithm)
02430     {
02431     case ALGORITHM_MONTE_CARLO:
02432       calibrate_algorithm = calibrate_MonteCarlo
    ;
02433       break;
02434     case ALGORITHM_SWEEP:
```

```
02435       calibrate_algorithm = calibrate_sweep;
02436       break;
02437     default:
02438       calibrate_algorithm = calibrate_genetic
    ;
02439       calibrate->mutation_ratio = input->mutation_ratio
    ;
02440       calibrate->reproduction_ratio = input->
    reproduction_ratio;
02441       calibrate->adaptation_ratio = input->adaptation_ratio
    ;
02442     }
02443   calibrate->nvariables = input->nvariables;
02444   calibrate->nsimulations = input->nsimulations;
02445   calibrate->niterations = input->niterations;
02446   calibrate->nbest = input->nbest;
02447   calibrate->tolerance = input->tolerance;
02448   calibrate->nsteps = input->nsteps;
02449   calibrate->nestimates = 0;
02450   if (input->nsteps)
02451     {
02452       calibrate->gradient_method = input->gradient_method
    ;
02453       calibrate->relaxation = input->relaxation;
02454       switch (input->gradient_method)
02455         {
02456         case GRADIENT_METHOD_COORDINATES:
02457           calibrate->nestimates = 2 * calibrate->nvariables
    ;
02458           calibrate_estimate_gradient =
    calibrate_estimate_gradient_coordinates;
02459           break;
02460         default:
02461           calibrate->nestimates = input->nestimates;
02462           calibrate_estimate_gradient =
    calibrate_estimate_gradient_random;
02463         }
02464     }
02465
02466 #if DEBUG
02467   fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02468 #endif
02469   calibrate->simulation_best
02470     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02471   calibrate->error_best
02472     = (double *) alloca (calibrate->nbest * sizeof (double));
02473
02474   // Reading the experimental data
02475 #if DEBUG
02476   buffer = g_get_current_dir ();
02477   fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02478   g_free (buffer);
02479 #endif
02480   calibrate->nexperiments = input->nexperiments;
02481   calibrate->ninputs = input->ninputs;
02482   calibrate->experiment = input->experiment;
02483   calibrate->weight = input->weight;
02484   for (i = 0; i < input->ninputs; ++i)
02485     {
02486       calibrate->template[i] = input->template[i];
02487       calibrate->file[i]
02488         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02489     }
02490   for (i = 0; i < input->nexperiments; ++i)
02491     {
02492 #if DEBUG
02493       fprintf (stderr, "calibrate_open: i=%u\n", i);
02494       fprintf (stderr, "calibrate_open: experiment=%s\n",
02495                calibrate->experiment[i]);
02496       fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[
    i]);
02497 #endif
02498       for (j = 0; j < input->ninputs; ++j)
02499         {
02500 #if DEBUG
02501           fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02502           fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
02503                    i, j + 1, calibrate->template[j][i]);
02504 #endif
02505           calibrate->file[j][i]
02506             = g_mapped_file_new (input->template[j][i], 0, NULL);
02507         }
02508     }
02509
02510   // Reading the variables data
02511 #if DEBUG
02512   fprintf (stderr, "calibrate_open: reading variables\n");
```

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

```
02590     }
02591 #else
02592   calibrate->nstart = 0;
02593   calibrate->nend = calibrate->nsimulations;
02594   if (calibrate->nsteps)
02595     {
02596       calibrate->nstart_gradient = 0;
02597       calibrate->nend_gradient = calibrate->nestimates;
02598     }
02599 #endif
02600 #if DEBUG
02601   fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart
    ,
02602            calibrate->nend);
02603 #endif
02604
02605   // Calculating simulations to perform for each thread
02606   calibrate->thread
02607     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02608   for (i = 0; i <= nthreads; ++i)
02609     {
02610       calibrate->thread[i] = calibrate->nstart
02611         + i * (calibrate->nend - calibrate->nstart) / nthreads
    ;
02612 #if DEBUG
02613       fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02614                calibrate->thread[i]);
02615 #endif
02616     }
02617   if (calibrate->nsteps)
02618     calibrate->thread_gradient = (unsigned int *)
02619       alloca ((1 + nthreads_gradient) * sizeof (unsigned int))
    ;
02620
02621   // Opening result files
02622   calibrate->file_result = g_fopen (calibrate->result, "w");
02623   calibrate->file_variables = g_fopen (calibrate->variables
    , "w");
02624
02625   // Performing the algorithm
02626   switch (calibrate->algorithm)
02627     {
02628       // Genetic algorithm
02629     case ALGORITHM_GENETIC:
02630       calibrate_genetic ();
02631       break;
02632
02633       // Iterative algorithm
02634     default:
02635       calibrate_iterate ();
02636     }
02637
02638   // Getting calculation time
02639   t = g_date_time_new_now (tz);
02640   calibrate->calculation_time = 0.000001 *
    g_date_time_difference (t, t0);
02641   g_date_time_unref (t);
02642   g_date_time_unref (t0);
02643   g_time_zone_unref (tz);
02644   printf ("%s = %.6lg s\n",
02645           gettext ("Calculation time"), calibrate->calculation_time
    );
02646   fprintf (calibrate->file_result, "%s = %.6lg s\n",
02647            gettext ("Calculation time"), calibrate->calculation_time
    );
02648
02649   // Closing result files
02650   fclose (calibrate->file_variables);
02651   fclose (calibrate->file_result);
02652
02653 #if DEBUG
02654   fprintf (stderr, "calibrate_open: end\n");
02655 #endif
02656 }
02657
02658 #if HAVE_GTK
02659
02666 void
02667 input_save_gradient (xmlNode * node)
02668 {
02669 #if DEBUG
02670   fprintf (stderr, "input_save_gradient: start\n");
02671 #endif
02672   if (input->nsteps)
02673     {
02674       xml_node_set_uint (node, XML_NSTEPS, input->
    nsteps);
```

```
02675       if (input->relaxation != DEFAULT_RELAXATION)
02676         xml_node_set_float (node, XML_RELAXATION
      , input->relaxation);
02677       switch (input->gradient_method)
02678         {
02679         case GRADIENT_METHOD_COORDINATES:
02680           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02681           break;
02682         default:
02683           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM
      );
02684           xml_node_set_uint (node, XML_NESTIMATES
      , input->nestimates);
02685         }
02686     }
02687 #if DEBUG
02688   fprintf (stderr, "input_save_gradient: end\n");
02689 #endif
02690 }
02691
02698 void
02699 input_save (char *filename)
02700 {
02701   unsigned int i, j;
02702   char *buffer;
02703   xmlDoc *doc;
02704   xmlNode *node, *child;
02705   GFile *file, *file2;
02706
02707 #if DEBUG
02708   fprintf (stderr, "input_save: start\n");
02709 #endif
02710
02711   // Getting the input file directory
02712   input->name = g_path_get_basename (filename);
02713   input->directory = g_path_get_dirname (filename);
02714   file = g_file_new_for_path (input->directory);
02715
02716   // Opening the input file
02717   doc = xmlNewDoc ((const xmlChar *) "1.0");
02718
02719   // Setting root XML node
02720   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02721   xmlDocSetRootElement (doc, node);
02722
02723   // Adding properties to the root XML node
02724   if (xmlStrcmp ((const xmlChar *) input->result, result_name)
      )
02725     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02726   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name
      ))
02727     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables
      );
02728   file2 = g_file_new_for_path (input->simulator);
02729   buffer = g_file_get_relative_path (file, file2);
02730   g_object_unref (file2);
02731   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02732   g_free (buffer);
02733   if (input->evaluator)
02734     {
02735       file2 = g_file_new_for_path (input->evaluator);
02736       buffer = g_file_get_relative_path (file, file2);
02737       g_object_unref (file2);
02738       if (xmlStrlen ((xmlChar *) buffer))
02739         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02740       g_free (buffer);
02741     }
02742   if (input->seed != DEFAULT_RANDOM_SEED)
02743     xml_node_set_uint (node, XML_SEED, input->seed
      );
02744
02745   // Setting the algorithm
02746   buffer = (char *) g_malloc (64);
02747   switch (input->algorithm)
02748     {
02749     case ALGORITHM_MONTE_CARLO:
02750       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO
      );
02751       snprintf (buffer, 64, "%u", input->nsimulations);
02752       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02753       snprintf (buffer, 64, "%u", input->niterations);
02754       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02755       snprintf (buffer, 64, "%.3lg", input->tolerance);
02756       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02757       snprintf (buffer, 64, "%u", input->nbest);
02758       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
```

```
02759       input_save_gradient (node);
02760       break;
02761     case ALGORITHM_SWEEP:
02762       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02763       snprintf (buffer, 64, "%u", input->niterations);
02764       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02765       snprintf (buffer, 64, "%.3lg", input->tolerance);
02766       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02767       snprintf (buffer, 64, "%u", input->nbest);
02768       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02769       input_save_gradient (node);
02770       break;
02771     default:
02772       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02773       snprintf (buffer, 64, "%u", input->nsimulations);
02774       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02775       snprintf (buffer, 64, "%u", input->niterations);
02776       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02777       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02778       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02779       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio
    );
02780       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02781       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02782       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02783       break;
02784     }
02785   g_free (buffer);
02786
02787   // Setting the experimental data
02788   for (i = 0; i < input->nexperiments; ++i)
02789     {
02790       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02791       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment
    [i]);
02792       if (input->weight[i] != 1.)
02793         xml_node_set_float (child, XML_WEIGHT,
    input->weight[i]);
02794       for (j = 0; j < input->ninputs; ++j)
02795         xmlSetProp (child, template[j], (xmlChar *) input->template[j][
    i]);
02796     }
02797
02798   // Setting the variables data
02799   for (i = 0; i < input->nvariables; ++i)
02800     {
02801       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02802       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02803       xml_node_set_float (child, XML_MINIMUM,
    input->rangemin[i]);
02804       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02805         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM
    , input->rangeminabs[i]);
02806       xml_node_set_float (child, XML_MAXIMUM,
    input->rangemax[i]);
02807       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02808         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM
    , input->rangemaxabs[i]);
02809       if (input->precision[i] != DEFAULT_PRECISION)
02810         xml_node_set_uint (child, XML_PRECISION,
    input->precision[i]);
02811       if (input->algorithm == ALGORITHM_SWEEP)
02812         xml_node_set_uint (child, XML_NSWEEPS,
    input->nsweeps[i]);
02813       else if (input->algorithm == ALGORITHM_GENETIC)
02814         xml_node_set_uint (child, XML_NBITS, input->
    nbits[i]);
02815       if (input->nsteps)
02816         xml_node_set_float (child, XML_STEP, input->
    step[i]);
02817     }
02818
02819   // Saving the XML file
02820   xmlSaveFormatFile (filename, doc, 1);
02821
02822   // Freeing memory
02823   xmlFreeDoc (doc);
02824
02825 #if DEBUG
02826   fprintf (stderr, "input_save: end\n");
02827 #endif
02828 }
02829
02834 void
02835 options_new ()
02836 {
02837 #if DEBUG
```

```
02838   fprintf (stderr, "options_new: start\n");
02839 #endif
02840   options->label_seed = (GtkLabel *)
02841     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02842   options->spin_seed = (GtkSpinButton *)
02843     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02844   gtk_widget_set_tooltip_text
02845     (GTK_WIDGET (options->spin_seed),
02846      gettext ("Seed to init the pseudo-random numbers generator"));
02847   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed
      );
02848   options->label_threads = (GtkLabel *)
02849     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02850   options->spin_threads
02851     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02852   gtk_widget_set_tooltip_text
02853     (GTK_WIDGET (options->spin_threads),
02854      gettext ("Number of threads to perform the calibration/optimization for "
02855               "the stochastic algorithm"));
02856   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
02857   options->label_gradient = (GtkLabel *)
02858     gtk_label_new (gettext ("Threads number for the gradient based method"));
02859   options->spin_gradient
02860     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02861   gtk_widget_set_tooltip_text
02862     (GTK_WIDGET (options->spin_gradient),
02863      gettext ("Number of threads to perform the calibration/optimization for "
02864               "the gradient based method"));
02865   gtk_spin_button_set_value (options->spin_gradient,
02866                              (gdouble) nthreads_gradient);
02867   options->grid = (GtkGrid *) gtk_grid_new ();
02868   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed
      ), 0, 0, 1, 1);
02869   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed),
       1, 0, 1, 1);
02870   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads
      ),
02871                    0, 1, 1, 1);
02872   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads
      ),
02873                    1, 1, 1, 1);
02874   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient
      ),
02875                    0, 2, 1, 1);
02876   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient
      ),
02877                    1, 2, 1, 1);
02878   gtk_widget_show_all (GTK_WIDGET (options->grid));
02879   options->dialog = (GtkDialog *)
02880     gtk_dialog_new_with_buttons (gettext ("Options"),
02881                                  window->window,
02882                                  GTK_DIALOG_MODAL,
02883                                  gettext ("_OK"), GTK_RESPONSE_OK,
02884                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02885                                  NULL);
02886   gtk_container_add
02887     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02888      GTK_WIDGET (options->grid));
02889   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02890     {
02891       input->seed
02892         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed
      );
02893       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads
      );
02894       nthreads_gradient
02895         = gtk_spin_button_get_value_as_int (options->spin_gradient
      );
02896     }
02897   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02898 #if DEBUG
02899   fprintf (stderr, "options_new: end\n");
02900 #endif
02901 }
02902
02907 void
02908 running_new ()
02909 {
02910 #if DEBUG
02911   fprintf (stderr, "running_new: start\n");
02912 #endif
02913   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ...")
      );
02914   running->dialog = (GtkDialog *)
02915     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02916                                  window->window, GTK_DIALOG_MODAL, NULL,
```

```
      NULL);
02917   gtk_container_add
02918     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02919      GTK_WIDGET (running->label));
02920   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02921 #if DEBUG
02922   fprintf (stderr, "running_new: end\n");
02923 #endif
02924 }
02925
02931 int
02932 window_get_algorithm ()
02933 {
02934   unsigned int i;
02935 #if DEBUG
02936   fprintf (stderr, "window_get_algorithm: start\n");
02937 #endif
02938   for (i = 0; i < NALGORITHMS; ++i)
02939     if (gtk_toggle_button_get_active
02940         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02941       break;
02942 #if DEBUG
02943   fprintf (stderr, "window_get_algorithm: %u\n", i);
02944   fprintf (stderr, "window_get_algorithm: end\n");
02945 #endif
02946   return i;
02947 }
02948
02954 int
02955 window_get_gradient ()
02956 {
02957   unsigned int i;
02958 #if DEBUG
02959   fprintf (stderr, "window_get_gradient: start\n");
02960 #endif
02961   for (i = 0; i < NGRADIENTS; ++i)
02962     if (gtk_toggle_button_get_active
02963         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02964       break;
02965 #if DEBUG
02966   fprintf (stderr, "window_get_gradient: %u\n", i);
02967   fprintf (stderr, "window_get_gradient: end\n");
02968 #endif
02969   return i;
02970 }
02971
02976 void
02977 window_save_gradient ()
02978 {
02979 #if DEBUG
02980   fprintf (stderr, "window_save_gradient: start\n");
02981 #endif
02982   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient
      )))
02983     {
02984       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
02985       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
02986       switch (window_get_gradient ())
02987         {
02988         case GRADIENT_METHOD_COORDINATES:
02989           input->gradient_method = GRADIENT_METHOD_COORDINATES
      ;
02990           break;
02991         default:
02992           input->gradient_method = GRADIENT_METHOD_RANDOM
      ;
02993           input->nestimates
02994             = gtk_spin_button_get_value_as_int (window->spin_estimates
      );
02995         }
02996     }
02997   else
02998     input->nsteps = 0;
02999 #if DEBUG
03000   fprintf (stderr, "window_save_gradient: end\n");
03001 #endif
03002 }
03003
03009 int
03010 window_save ()
03011 {
03012   GtkFileChooserDialog *dlg;
03013   GtkFileFilter *filter;
03014   char *buffer;
03015
```

```
03016 #if DEBUG
03017   fprintf (stderr, "window_save: start\n");
03018 #endif
03019
03020   // Opening the saving dialog
03021   dlg = (GtkFileChooserDialog *)
03022     gtk_file_chooser_dialog_new (gettext ("Save file"),
03023                                  window->window,
03024                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03025                                  gettext ("_Cancel"),
03026                                  GTK_RESPONSE_CANCEL,
03027                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03028   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE)
      ;
03029   buffer = g_build_filename (input->directory, input->name, NULL);
03030   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03031   g_free (buffer);
03032
03033   // Adding XML filter
03034   filter = (GtkFileFilter *) gtk_file_filter_new ();
03035   gtk_file_filter_set_name (filter, "XML");
03036   gtk_file_filter_add_pattern (filter, "*.xml");
03037   gtk_file_filter_add_pattern (filter, "*.XML");
03038   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03039
03040   // If OK response then saving
03041   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03042     {
03043
03044       // Adding properties to the root XML node
03045       input->simulator = gtk_file_chooser_get_filename
03046         (GTK_FILE_CHOOSER (window->button_simulator));
03047       if (gtk_toggle_button_get_active
03048           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03049         input->evaluator = gtk_file_chooser_get_filename
03050           (GTK_FILE_CHOOSER (window->button_evaluator));
03051       else
03052         input->evaluator = NULL;
03053       input->result
03054         = (char *) xmlStrdup ((const xmlChar *)
03055                               gtk_entry_get_text (window->entry_result
      ));
03056       input->variables
03057         = (char *) xmlStrdup ((const xmlChar *)
03058                               gtk_entry_get_text (window->entry_variables
      ));
03059
03060       // Setting the algorithm
03061       switch (window_get_algorithm ())
03062         {
03063         case ALGORITHM_MONTE_CARLO:
03064           input->algorithm = ALGORITHM_MONTE_CARLO
      ;
03065           input->nsimulations
03066             = gtk_spin_button_get_value_as_int (window->spin_simulations
      );
03067           input->niterations
03068             = gtk_spin_button_get_value_as_int (window->spin_iterations
      );
03069           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03070           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03071           window_save_gradient ();
03072           break;
03073         case ALGORITHM_SWEEP:
03074           input->algorithm = ALGORITHM_SWEEP;
03075           input->niterations
03076             = gtk_spin_button_get_value_as_int (window->spin_iterations
      );
03077           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03078           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03079           window_save_gradient ();
03080           break;
03081         default:
03082           input->algorithm = ALGORITHM_GENETIC;
03083           input->nsimulations
03084             = gtk_spin_button_get_value_as_int (window->spin_population
      );
03085           input->niterations
03086             = gtk_spin_button_get_value_as_int (window->spin_generations
      );
03087           input->mutation_ratio
03088             = gtk_spin_button_get_value (window->spin_mutation);
03089           input->reproduction_ratio
```

```
03090             = gtk_spin_button_get_value (window->spin_reproduction
      );
03091           input->adaptation_ratio
03092             = gtk_spin_button_get_value (window->spin_adaptation
      );
03093           break;
03094         }
03095
03096       // Saving the XML file
03097       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03098       input_save (buffer);
03099
03100       // Closing and freeing memory
03101       g_free (buffer);
03102       gtk_widget_destroy (GTK_WIDGET (dlg));
03103 #if DEBUG
03104       fprintf (stderr, "window_save: end\n");
03105 #endif
03106       return 1;
03107     }
03108
03109   // Closing and freeing memory
03110   gtk_widget_destroy (GTK_WIDGET (dlg));
03111 #if DEBUG
03112   fprintf (stderr, "window_save: end\n");
03113 #endif
03114   return 0;
03115 }
03116
03121 void
03122 window_run ()
03123 {
03124   unsigned int i;
03125   char *msg, *msg2, buffer[64], buffer2[64];
03126 #if DEBUG
03127   fprintf (stderr, "window_run: start\n");
03128 #endif
03129   if (!window_save ())
03130     {
03131 #if DEBUG
03132       fprintf (stderr, "window_run: end\n");
03133 #endif
03134       return;
03135     }
03136   running_new ();
03137   while (gtk_events_pending ())
03138     gtk_main_iteration ();
03139   calibrate_open ();
03140   gtk_widget_destroy (GTK_WIDGET (running->dialog));
03141   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03142   msg2 = g_strdup (buffer);
03143   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03144     {
03145       snprintf (buffer, 64, "%s = %s\n",
03146                 calibrate->label[i], format[calibrate->precision
      [i]]);
03147       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03148       msg = g_strconcat (msg2, buffer2, NULL);
03149       g_free (msg2);
03150     }
03151   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03152             calibrate->calculation_time);
03153   msg = g_strconcat (msg2, buffer, NULL);
03154   g_free (msg2);
03155   show_message (gettext ("Best result"), msg, INFO_TYPE);
03156   g_free (msg);
03157   calibrate_free ();
03158 #if DEBUG
03159   fprintf (stderr, "window_run: end\n");
03160 #endif
03161 }
03162
03167 void
03168 window_help ()
03169 {
03170   char *buffer, *buffer2;
03171 #if DEBUG
03172   fprintf (stderr, "window_help: start\n");
03173 #endif
03174   buffer2 = g_build_filename (window->application_directory
      , "..", "manuals",
03175                               gettext ("user-manual.pdf"), NULL);
03176   buffer = g_filename_to_uri (buffer2, NULL, NULL);
03177   g_free (buffer2);
03178   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03179 #if DEBUG
03180   fprintf (stderr, "window_help: uri=%s\n", buffer);
```

```
03181 #endif
03182   g_free (buffer);
03183 #if DEBUG
03184   fprintf (stderr, "window_help: end\n");
03185 #endif
03186 }
03187
03192 void
03193 window_about ()
03194 {
03195   static const gchar *authors[] = {
03196     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03197     "Borja Latorre Garcés <borja.latorre@csic.es>",
03198     NULL
03199   };
03200 #if DEBUG
03201   fprintf (stderr, "window_about: start\n");
03202 #endif
03203   gtk_show_about_dialog
03204     (window->window,
03205      "program_name", "MPCOTool",
03206      "comments",
03207      gettext ("A software to perform calibrations/optimizations of empirical "
03208               "parameters"),
03209      "authors", authors,
03210      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03211      "version", "1.2.5",
03212      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03213      "logo", window->logo,
03214      "website", "https://github.com/jburguete/mpcotool",
03215      "license-type", GTK_LICENSE_BSD, NULL);
03216 #if DEBUG
03217   fprintf (stderr, "window_about: end\n");
03218 #endif
03219 }
03220
03226 void
03227 window_update_gradient ()
03228 {
03229 #if DEBUG
03230   fprintf (stderr, "window_update_gradient: start\n");
03231 #endif
03232   gtk_widget_show (GTK_WIDGET (window->check_gradient));
03233   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient
      )))
03234     {
03235       gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03236       gtk_widget_show (GTK_WIDGET (window->label_step));
03237       gtk_widget_show (GTK_WIDGET (window->spin_step));
03238     }
03239   switch (window_get_gradient ())
03240     {
03241     case GRADIENT_METHOD_COORDINATES:
03242       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03243       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03244       break;
03245     default:
03246       gtk_widget_show (GTK_WIDGET (window->label_estimates));
03247       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03248     }
03249 #if DEBUG
03250   fprintf (stderr, "window_update_gradient: end\n");
03251 #endif
03252 }
03253
03258 void
03259 window_update ()
03260 {
03261   unsigned int i;
03262 #if DEBUG
03263   fprintf (stderr, "window_update: start\n");
03264 #endif
03265   gtk_widget_set_sensitive
03266     (GTK_WIDGET (window->button_evaluator),
03267      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03268                                    (window->check_evaluator)));
03269   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03270   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03271   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03272   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03273   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03274   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03275   gtk_widget_hide (GTK_WIDGET (window->label_bests));
03276   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03277   gtk_widget_hide (GTK_WIDGET (window->label_population));
03278   gtk_widget_hide (GTK_WIDGET (window->spin_population));
03279   gtk_widget_hide (GTK_WIDGET (window->label_generations));
```

```
03280   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03281   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03282   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03283   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03284   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03285   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03286   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03287   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03288   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03289   gtk_widget_hide (GTK_WIDGET (window->label_bits));
03290   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03291   gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03292   gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03293   gtk_widget_hide (GTK_WIDGET (window->label_step));
03294   gtk_widget_hide (GTK_WIDGET (window->spin_step));
03295   i = gtk_spin_button_get_value_as_int (window->spin_iterations)
    ;
03296   switch (window_get_algorithm ())
03297     {
03298     case ALGORITHM_MONTE_CARLO:
03299       gtk_widget_show (GTK_WIDGET (window->label_simulations))
    ;
03300       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03301       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03302       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03303       if (i > 1)
03304         {
03305           gtk_widget_show (GTK_WIDGET (window->label_tolerance))
    ;
03306           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03307           gtk_widget_show (GTK_WIDGET (window->label_bests));
03308           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03309         }
03310       window_update_gradient ();
03311       break;
03312     case ALGORITHM_SWEEP:
03313       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03314       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03315       if (i > 1)
03316         {
03317           gtk_widget_show (GTK_WIDGET (window->label_tolerance))
    ;
03318           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03319           gtk_widget_show (GTK_WIDGET (window->label_bests));
03320           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03321         }
03322       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03323       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03324       gtk_widget_show (GTK_WIDGET (window->check_gradient));
03325       window_update_gradient ();
03326       break;
03327     default:
03328       gtk_widget_show (GTK_WIDGET (window->label_population));
03329       gtk_widget_show (GTK_WIDGET (window->spin_population));
03330       gtk_widget_show (GTK_WIDGET (window->label_generations))
    ;
03331       gtk_widget_show (GTK_WIDGET (window->spin_generations));
03332       gtk_widget_show (GTK_WIDGET (window->label_mutation));
03333       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03334       gtk_widget_show (GTK_WIDGET (window->label_reproduction
    ));
03335       gtk_widget_show (GTK_WIDGET (window->spin_reproduction))
    ;
03336       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03337       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03338       gtk_widget_show (GTK_WIDGET (window->label_bits));
03339       gtk_widget_show (GTK_WIDGET (window->spin_bits));
03340     }
03341   gtk_widget_set_sensitive
03342     (GTK_WIDGET (window->button_remove_experiment),
    input->nexperiments > 1);
03343   gtk_widget_set_sensitive
03344     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
03345   for (i = 0; i < input->ninputs; ++i)
03346     {
03347       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03348       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03349       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template
    [i]), 0);
03350       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template
    [i]), 1);
03351       g_signal_handler_block
03352         (window->check_template[i], window->id_template
    [i]);
03353       g_signal_handler_block (window->button_template[i], window
    ->id_input[i]);
```

```
03354       gtk_toggle_button_set_active
03355         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03356       g_signal_handler_unblock
03357         (window->button_template[i], window->id_input[i]
      );
03358       g_signal_handler_unblock
03359         (window->check_template[i], window->id_template
      [i]);
03360     }
03361   if (i > 0)
03362     {
03363       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template
      [i - 1]), 1);
03364       gtk_widget_set_sensitive
03365         (GTK_WIDGET (window->button_template[i - 1]),
03366          gtk_toggle_button_get_active
03367          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
03368     }
03369   if (i < MAX_NINPUTS)
03370     {
03371       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03372       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03373       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template
      [i]), 1);
03374       gtk_widget_set_sensitive
03375         (GTK_WIDGET (window->button_template[i]),
03376          gtk_toggle_button_get_active
03377          GTK_TOGGLE_BUTTON (window->check_template[i]));
03378       g_signal_handler_block
03379         (window->check_template[i], window->id_template
      [i]);
03380       g_signal_handler_block (window->button_template[i], window
      ->id_input[i]);
03381       gtk_toggle_button_set_active
03382         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03383       g_signal_handler_unblock
03384         (window->button_template[i], window->id_input[i]
      );
03385       g_signal_handler_unblock
03386         (window->check_template[i], window->id_template
      [i]);
03387     }
03388   while (++i < MAX_NINPUTS)
03389     {
03390       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03391       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03392     }
03393   gtk_widget_set_sensitive
03394     (GTK_WIDGET (window->spin_minabs),
03395      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs
      )));
03396   gtk_widget_set_sensitive
03397     (GTK_WIDGET (window->spin_maxabs),
03398      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs
      )));
03399 #if DEBUG
03400   fprintf (stderr, "window_update: end\n");
03401 #endif
03402 }
03403
03408 void
03409 window_set_algorithm ()
03410 {
03411   int i;
03412 #if DEBUG
03413   fprintf (stderr, "window_set_algorithm: start\n");
03414 #endif
03415   i = window_get_algorithm ();
03416   switch (i)
03417     {
03418     case ALGORITHM_SWEEP:
03419       input->nsweeps = (unsigned int *) g_realloc
03420         (input->nsweeps, input->nvariables * sizeof (unsigned
      int));
03421       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03422       if (i < 0)
03423         i = 0;
03424       gtk_spin_button_set_value (window->spin_sweeps,
03425                                  (gdouble) input->nsweeps[i]);
03426       break;
03427     case ALGORITHM_GENETIC:
03428       input->nbits = (unsigned int *) g_realloc
03429         (input->nbits, input->nvariables * sizeof (unsigned int)
      );
03430       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
```

```
03431       if (i < 0)
03432         i = 0;
03433       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03434     }
03435   window_update ();
03436 #if DEBUG
03437   fprintf (stderr, "window_set_algorithm: end\n");
03438 #endif
03439 }
03440
03445 void
03446 window_set_experiment ()
03447 {
03448   unsigned int i, j;
03449   char *buffer1, *buffer2;
03450 #if DEBUG
03451   fprintf (stderr, "window_set_experiment: start\n");
03452 #endif
03453   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment
      ));
03454   gtk_spin_button_set_value (window->spin_weight, input->weight
      [i]);
03455   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment
      );
03456   buffer2 = g_build_filename (input->directory, buffer1, NULL);
03457   g_free (buffer1);
03458   g_signal_handler_block
03459     (window->button_experiment, window->id_experiment_name
      );
03460   gtk_file_chooser_set_filename
03461     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03462   g_signal_handler_unblock
03463     (window->button_experiment, window->id_experiment_name
      );
03464   g_free (buffer2);
03465   for (j = 0; j < input->ninputs; ++j)
03466     {
03467       g_signal_handler_block (window->button_template[j], window
      ->id_input[j]);
03468       buffer2
03469         = g_build_filename (input->directory, input->template[
      j][i], NULL);
03470       gtk_file_chooser_set_filename
03471         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2)
      ;
03472       g_free (buffer2);
03473       g_signal_handler_unblock
03474         (window->button_template[j], window->id_input[j]
      );
03475     }
03476 #if DEBUG
03477   fprintf (stderr, "window_set_experiment: end\n");
03478 #endif
03479 }
03480
03485 void
03486 window_remove_experiment ()
03487 {
03488   unsigned int i, j;
03489 #if DEBUG
03490   fprintf (stderr, "window_remove_experiment: start\n");
03491 #endif
03492   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment
      ));
03493   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03494   gtk_combo_box_text_remove (window->combo_experiment, i);
03495   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03496   xmlFree (input->experiment[i]);
03497   --input->nexperiments;
03498   for (j = i; j < input->nexperiments; ++j)
03499     {
03500       input->experiment[j] = input->experiment[j + 1];
03501       input->weight[j] = input->weight[j + 1];
03502     }
03503   j = input->nexperiments - 1;
03504   if (i > j)
03505     i = j;
03506   for (j = 0; j < input->ninputs; ++j)
03507     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03508   g_signal_handler_block
03509     (window->button_experiment, window->id_experiment_name
      );
03510   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment
```

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

```
03587   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03588   g_free (buffer);
03589   g_object_unref (file2);
03590   g_object_unref (file1);
03591 #if DEBUG
03592   fprintf (stderr, "window_name_experiment: end\n");
03593 #endif
03594 }
03595
03600 void
03601 window_weight_experiment ()
03602 {
03603   unsigned int i;
03604 #if DEBUG
03605   fprintf (stderr, "window_weight_experiment: start\n");
03606 #endif
03607   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment
      ));
03608   input->weight[i] = gtk_spin_button_get_value (window->spin_weight
      );
03609 #if DEBUG
03610   fprintf (stderr, "window_weight_experiment: end\n");
03611 #endif
03612 }
03613
03619 void
03620 window_inputs_experiment ()
03621 {
03622   unsigned int j;
03623 #if DEBUG
03624   fprintf (stderr, "window_inputs_experiment: start\n");
03625 #endif
03626   j = input->ninputs - 1;
03627   if (j
03628       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03629                                         (window->check_template[j
      ])))
03630     --input->ninputs;
03631   if (input->ninputs < MAX_NINPUTS
03632       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03633                                        (window->check_template[j]
      )))
03634     {
03635       ++input->ninputs;
03636       for (j = 0; j < input->ninputs; ++j)
03637         {
03638           input->template[j] = (char **)
03639             g_realloc (input->template[j], input->nvariables
      * sizeof (char *));
03640         }
03641     }
03642   window_update ();
03643 #if DEBUG
03644   fprintf (stderr, "window_inputs_experiment: end\n");
03645 #endif
03646 }
03647
03655 void
03656 window_template_experiment (void *data)
03657 {
03658   unsigned int i, j;
03659   char *buffer;
03660   GFile *file1, *file2;
03661 #if DEBUG
03662   fprintf (stderr, "window_template_experiment: start\n");
03663 #endif
03664   i = (size_t) data;
03665   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment
      ));
03666   file1
03667     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template
      [i]));
03668   file2 = g_file_new_for_path (input->directory);
03669   buffer = g_file_get_relative_path (file2, file1);
03670   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03671   g_free (buffer);
03672   g_object_unref (file2);
03673   g_object_unref (file1);
03674 #if DEBUG
03675   fprintf (stderr, "window_template_experiment: end\n");
03676 #endif
03677 }
03678
03683 void
03684 window_set_variable ()
03685 {
```

```
03686   unsigned int i;
03687 #if DEBUG
03688   fprintf (stderr, "window_set_variable: start\n");
03689 #endif
03690   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03691   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03692   gtk_entry_set_text (window->entry_variable, input->label[i
      ]);
03693   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03694   gtk_spin_button_set_value (window->spin_min, input->rangemin[
      i]);
03695   gtk_spin_button_set_value (window->spin_max, input->rangemax[
      i]);
03696   if (input->rangeminabs[i] != -G_MAXDOUBLE)
03697     {
03698       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
03699       gtk_toggle_button_set_active
03700         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03701     }
03702   else
03703     {
03704       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03705       gtk_toggle_button_set_active
03706         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03707     }
03708   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03709     {
03710       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
03711       gtk_toggle_button_set_active
03712         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03713     }
03714   else
03715     {
03716       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03717       gtk_toggle_button_set_active
03718         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03719     }
03720   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
03721   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
03722   if (input->nsteps)
03723     gtk_spin_button_set_value (window->spin_step, input->step[i]);
03724 #if DEBUG
03725   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03726            input->precision[i]);
03727 #endif
03728   switch (window_get_algorithm ())
03729     {
03730     case ALGORITHM_SWEEP:
03731       gtk_spin_button_set_value (window->spin_sweeps,
03732                                  (gdouble) input->nsweeps[i]);
03733 #if DEBUG
03734       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03735                input->nsweeps[i]);
03736 #endif
03737       break;
03738     case ALGORITHM_GENETIC:
03739       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03740 #if DEBUG
03741       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03742                input->nbits[i]);
03743 #endif
03744       break;
03745     }
03746   window_update ();
03747 #if DEBUG
03748   fprintf (stderr, "window_set_variable: end\n");
03749 #endif
03750 }
03751
03756 void
03757 window_remove_variable ()
03758 {
03759   unsigned int i, j;
03760 #if DEBUG
03761   fprintf (stderr, "window_remove_variable: start\n");
03762 #endif
03763   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03764   g_signal_handler_block (window->combo_variable, window->
```

```
id_variable);
03765   gtk_combo_box_text_remove (window->combo_variable, i);
03766   g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03767   xmlFree (input->label[i]);
03768   --input->nvariables;
03769   for (j = i; j < input->nvariables; ++j)
03770     {
03771       input->label[j] = input->label[j + 1];
03772       input->rangemin[j] = input->rangemin[j + 1];
03773       input->rangemax[j] = input->rangemax[j + 1];
03774       input->rangeminabs[j] = input->rangeminabs[j + 1];
03775       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03776       input->precision[j] = input->precision[j + 1];
03777       input->step[j] = input->step[j + 1];
03778       switch (window_get_algorithm ())
03779         {
03780         case ALGORITHM_SWEEP:
03781           input->nsweeps[j] = input->nsweeps[j + 1];
03782           break;
03783         case ALGORITHM_GENETIC:
03784           input->nbits[j] = input->nbits[j + 1];
03785         }
03786     }
03787   j = input->nvariables - 1;
03788   if (i > j)
03789     i = j;
03790   g_signal_handler_block (window->entry_variable, window->
id_variable_label);
03791   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable
), i);
03792   g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
03793   window_update ();
03794 #if DEBUG
03795   fprintf (stderr, "window_remove_variable: end\n");
03796 #endif
03797 }
03798
03803 void
03804 window_add_variable ()
03805 {
03806   unsigned int i, j;
03807 #if DEBUG
03808   fprintf (stderr, "window_add_variable: start\n");
03809 #endif
03810   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
));
03811   g_signal_handler_block (window->combo_variable, window->
id_variable);
03812   gtk_combo_box_text_insert_text (window->combo_variable, i,
input->label[i]);
03813   g_signal_handler_unblock (window->combo_variable, window->
id_variable);
03814   input->label = (char **) g_realloc
03815     (input->label, (input->nvariables + 1) * sizeof (char *));
03816   input->rangemin = (double *) g_realloc
03817     (input->rangemin, (input->nvariables + 1) * sizeof (
double));
03818   input->rangemax = (double *) g_realloc
03819     (input->rangemax, (input->nvariables + 1) * sizeof (
double));
03820   input->rangeminabs = (double *) g_realloc
03821     (input->rangeminabs, (input->nvariables + 1) * sizeof
(double));
03822   input->rangemaxabs = (double *) g_realloc
03823     (input->rangemaxabs, (input->nvariables + 1) * sizeof
(double));
03824   input->precision = (unsigned int *) g_realloc
03825     (input->precision, (input->nvariables + 1) * sizeof (
unsigned int));
03826   input->step = (double *) g_realloc
03827     (input->step, (input->nvariables + 1) * sizeof (double));
03828   for (j = input->nvariables - 1; j > i; --j)
03829     {
03830       input->label[j + 1] = input->label[j];
03831       input->rangemin[j + 1] = input->rangemin[j];
03832       input->rangemax[j + 1] = input->rangemax[j];
03833       input->rangeminabs[j + 1] = input->rangeminabs[j];
03834       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03835       input->precision[j + 1] = input->precision[j];
03836       input->step[j + 1] = input->step[j];
03837     }
03838   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[
j]);
03839   input->rangemin[j + 1] = input->rangemin[j];
03840   input->rangemax[j + 1] = input->rangemax[j];
```

```
03841   input->rangeminabs[j + 1] = input->rangeminabs[j];
03842   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03843   input->precision[j + 1] = input->precision[j];
03844   input->step[j + 1] = input->step[j];
03845   switch (window_get_algorithm ())
03846     {
03847     case ALGORITHM_SWEEP:
03848       input->nsweeps = (unsigned int *) g_realloc
03849         (input->nsweeps, (input->nvariables + 1) * sizeof (
      unsigned int));
03850       for (j = input->nvariables - 1; j > i; --j)
03851         input->nsweeps[j + 1] = input->nsweeps[j];
03852       input->nsweeps[j + 1] = input->nsweeps[j];
03853       break;
03854     case ALGORITHM_GENETIC:
03855       input->nbits = (unsigned int *) g_realloc
03856         (input->nbits, (input->nvariables + 1) * sizeof (
      unsigned int));
03857       for (j = input->nvariables - 1; j > i; --j)
03858         input->nbits[j + 1] = input->nbits[j];
03859       input->nbits[j + 1] = input->nbits[j];
03860     }
03861   ++input->nvariables;
03862   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03863   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable
      ), i + 1);
03864   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03865   window_update ();
03866 #if DEBUG
03867   fprintf (stderr, "window_add_variable: end\n");
03868 #endif
03869 }
03870
03875 void
03876 window_label_variable ()
03877 {
03878   unsigned int i;
03879   const char *buffer;
03880 #if DEBUG
03881   fprintf (stderr, "window_label_variable: start\n");
03882 #endif
03883   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03884   buffer = gtk_entry_get_text (window->entry_variable);
03885   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03886   gtk_combo_box_text_remove (window->combo_variable, i);
03887   gtk_combo_box_text_insert_text (window->combo_variable, i,
      buffer);
03888   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable
      ), i);
03889   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03890 #if DEBUG
03891   fprintf (stderr, "window_label_variable: end\n");
03892 #endif
03893 }
03894
03899 void
03900 window_precision_variable ()
03901 {
03902   unsigned int i;
03903 #if DEBUG
03904   fprintf (stderr, "window_precision_variable: start\n");
03905 #endif
03906   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03907   input->precision[i]
03908     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision
      );
03909   gtk_spin_button_set_digits (window->spin_min, input->precision
      [i]);
03910   gtk_spin_button_set_digits (window->spin_max, input->precision
      [i]);
03911   gtk_spin_button_set_digits (window->spin_minabs, input->precision
      [i]);
03912   gtk_spin_button_set_digits (window->spin_maxabs, input->precision
      [i]);
03913 #if DEBUG
03914   fprintf (stderr, "window_precision_variable: end\n");
03915 #endif
03916 }
03917
03922 void
03923 window_rangemin_variable ()
```

```
03924 {
03925   unsigned int i;
03926 #if DEBUG
03927   fprintf (stderr, "window_rangemin_variable: start\n");
03928 #endif
03929   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03930   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min
      );
03931 #if DEBUG
03932   fprintf (stderr, "window_rangemin_variable: end\n");
03933 #endif
03934 }
03935
03940 void
03941 window_rangemax_variable ()
03942 {
03943   unsigned int i;
03944 #if DEBUG
03945   fprintf (stderr, "window_rangemax_variable: start\n");
03946 #endif
03947   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03948   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max
      );
03949 #if DEBUG
03950   fprintf (stderr, "window_rangemax_variable: end\n");
03951 #endif
03952 }
03953
03958 void
03959 window_rangeminabs_variable ()
03960 {
03961   unsigned int i;
03962 #if DEBUG
03963   fprintf (stderr, "window_rangeminabs_variable: start\n");
03964 #endif
03965   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03966   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03967 #if DEBUG
03968   fprintf (stderr, "window_rangeminabs_variable: end\n");
03969 #endif
03970 }
03971
03976 void
03977 window_rangemaxabs_variable ()
03978 {
03979   unsigned int i;
03980 #if DEBUG
03981   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03982 #endif
03983   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
03984   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03985 #if DEBUG
03986   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03987 #endif
03988 }
03989
03994 void
03995 window_step_variable ()
03996 {
03997   unsigned int i;
03998 #if DEBUG
03999   fprintf (stderr, "window_step_variable: start\n");
04000 #endif
04001   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
04002   input->step[i] = gtk_spin_button_get_value (window->spin_step);
04003 #if DEBUG
04004   fprintf (stderr, "window_step_variable: end\n");
04005 #endif
04006 }
04007
04012 void
04013 window_update_variable ()
04014 {
04015   int i;
04016 #if DEBUG
04017   fprintf (stderr, "window_update_variable: start\n");
04018 #endif
04019   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable
      ));
04020   if (i < 0)
```

```
04021     i = 0;
04022   switch (window_get_algorithm ())
04023     {
04024     case ALGORITHM_SWEEP:
04025       input->nsweeps[i]
04026         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
04027 #if DEBUG
04028       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
04029                input->nsweeps[i]);
04030 #endif
04031       break;
04032     case ALGORITHM_GENETIC:
04033       input->nbits[i] = gtk_spin_button_get_value_as_int (window->
      spin_bits);
04034 #if DEBUG
04035       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
04036                input->nbits[i]);
04037 #endif
04038     }
04039 #if DEBUG
04040   fprintf (stderr, "window_update_variable: end\n");
04041 #endif
04042 }
04043
04051 int
04052 window_read (char *filename)
04053 {
04054   unsigned int i;
04055   char *buffer;
04056 #if DEBUG
04057   fprintf (stderr, "window_read: start\n");
04058 #endif
04059
04060   // Reading new input file
04061   input_free ();
04062   if (!input_open (filename))
04063     return 0;
04064
04065   // Setting GTK+ widgets data
04066   gtk_entry_set_text (window->entry_result, input->result);
04067   gtk_entry_set_text (window->entry_variables, input->variables
      );
04068   buffer = g_build_filename (input->directory, input->simulator
      , NULL);
04069   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04070                                  (window->button_simulator),
      buffer);
04071   g_free (buffer);
04072   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator
      ),
04073                                 (size_t) input->evaluator);
04074   if (input->evaluator)
04075     {
04076       buffer = g_build_filename (input->directory, input->evaluator
      , NULL);
04077       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04078                                      (window->button_evaluator)
      , buffer);
04079       g_free (buffer);
04080     }
04081   gtk_toggle_button_set_active
04082     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04083   switch (input->algorithm)
04084     {
04085     case ALGORITHM_MONTE_CARLO:
04086       gtk_spin_button_set_value (window->spin_simulations,
04087                                  (gdouble) input->nsimulations);
04088     case ALGORITHM_SWEEP:
04089       gtk_spin_button_set_value (window->spin_iterations,
04090                                  (gdouble) input->niterations);
04091       gtk_spin_button_set_value (window->spin_bests, (gdouble) input
      ->nbest);
04092       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
04093       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient
      ),
04094                                     input->nsteps);
04095       if (input->nsteps)
04096         {
04097           gtk_toggle_button_set_active
04098             (GTK_TOGGLE_BUTTON (window->button_gradient
04099                                 [input->gradient_method]), TRUE)
      ;
04100           gtk_spin_button_set_value (window->spin_steps,
04101                                      (gdouble) input->nsteps);
04102           gtk_spin_button_set_value (window->spin_relaxation,
```

```
04103                                     (gdouble) input->relaxation);
04104           switch (input->gradient_method)
04105             {
04106             case GRADIENT_METHOD_RANDOM:
04107               gtk_spin_button_set_value (window->spin_estimates,
04108                                          (gdouble) input->nestimates)
    ;
04109             }
04110         }
04111       break;
04112     default:
04113       gtk_spin_button_set_value (window->spin_population,
04114                                  (gdouble) input->nsimulations);
04115       gtk_spin_button_set_value (window->spin_generations,
04116                                  (gdouble) input->niterations);
04117       gtk_spin_button_set_value (window->spin_mutation, input->
    mutation_ratio);
04118       gtk_spin_button_set_value (window->spin_reproduction,
04119                                  input->reproduction_ratio);
04120       gtk_spin_button_set_value (window->spin_adaptation,
04121                                  input->adaptation_ratio);
04122     }
04123   g_signal_handler_block (window->combo_experiment, window->
    id_experiment);
04124   g_signal_handler_block (window->button_experiment,
04125                           window->id_experiment_name);
04126   gtk_combo_box_text_remove_all (window->combo_experiment);
04127   for (i = 0; i < input->nexperiments; ++i)
04128     gtk_combo_box_text_append_text (window->combo_experiment,
04129                                     input->experiment[i]);
04130   g_signal_handler_unblock
04131     (window->button_experiment, window->id_experiment_name
    );
04132   g_signal_handler_unblock (window->combo_experiment, window->
    id_experiment);
04133   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment
    ), 0);
04134   g_signal_handler_block (window->combo_variable, window->
    id_variable);
04135   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
04136   gtk_combo_box_text_remove_all (window->combo_variable);
04137   for (i = 0; i < input->nvariables; ++i)
04138     gtk_combo_box_text_append_text (window->combo_variable, input
    ->label[i]);
04139   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
04140   g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
04141   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable
    ), 0);
04142   window_set_variable ();
04143   window_update ();
04144
04145 #if DEBUG
04146   fprintf (stderr, "window_read: end\n");
04147 #endif
04148   return 1;
04149 }
04150
04155 void
04156 window_open ()
04157 {
04158   GtkFileChooserDialog *dlg;
04159   GtkFileFilter *filter;
04160   char *buffer, *directory, *name;
04161
04162 #if DEBUG
04163   fprintf (stderr, "window_open: start\n");
04164 #endif
04165
04166   // Saving a backup of the current input file
04167   directory = g_strdup (input->directory);
04168   name = g_strdup (input->name);
04169
04170   // Opening dialog
04171   dlg = (GtkFileChooserDialog *)
04172     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04173                                  window->window,
04174                                  GTK_FILE_CHOOSER_ACTION_OPEN,
04175                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04176                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04177
04178   // Adding XML filter
04179   filter = (GtkFileFilter *) gtk_file_filter_new ();
04180   gtk_file_filter_set_name (filter, "XML");
04181   gtk_file_filter_add_pattern (filter, "*.xml");
```

```
04182   gtk_file_filter_add_pattern (filter, "*.XML");
04183   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
04184
04185   // If OK saving
04186   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04187     {
04188
04189       // Traying to open the input file
04190       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04191       if (!window_read (buffer))
04192         {
04193 #if DEBUG
04194           fprintf (stderr, "window_open: error reading input file\n");
04195 #endif
04196           g_free (buffer);
04197
04198           // Reading backup file on error
04199           buffer = g_build_filename (directory, name, NULL);
04200           if (!input_open (buffer))
04201             {
04202
04203               // Closing on backup file reading error
04204 #if DEBUG
04205               fprintf (stderr, "window_read: error reading backup file\n");
04206 #endif
04207               g_free (buffer);
04208               break;
04209             }
04210           g_free (buffer);
04211         }
04212       else
04213         {
04214           g_free (buffer);
04215           break;
04216         }
04217     }
04218
04219   // Freeing and closing
04220   g_free (name);
04221   g_free (directory);
04222   gtk_widget_destroy (GTK_WIDGET (dlg));
04223 #if DEBUG
04224   fprintf (stderr, "window_open: end\n");
04225 #endif
04226 }
04227
04232 void
04233 window_new ()
04234 {
04235   unsigned int i;
04236   char *buffer, *buffer2, buffer3[64];
04237   char *label_algorithm[NALGORITHMS] = {
04238     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04239   };
04240   char *tip_algorithm[NALGORITHMS] = {
04241     gettext ("Monte-Carlo brute force algorithm"),
04242     gettext ("Sweep brute force algorithm"),
04243     gettext ("Genetic algorithm")
04244   };
04245   char *label_gradient[NGRADIENTS] = {
04246     gettext ("_Coordinates descent"), gettext ("_Random")
04247   };
04248   char *tip_gradient[NGRADIENTS] = {
04249     gettext ("Coordinates descent gradient estimate method"),
04250     gettext ("Random gradient estimate method")
04251   };
04252
04253 #if DEBUG
04254   fprintf (stderr, "window_new: start\n");
04255 #endif
04256
04257   // Creating the window
04258   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04259
04260   // Finish when closing the window
04261   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04262
04263   // Setting the window title
04264   gtk_window_set_title (window->window, "MPCOTool");
04265
04266   // Creating the open button
04267   window->button_open = (GtkToolButton *) gtk_tool_button_new
04268     (gtk_image_new_from_icon_name ("document-open",
04269                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04270      gettext ("Open"));
04271   g_signal_connect (window->button_open, "clicked", window_open
      , NULL);
```

```
04272
04273   // Creating the save button
04274   window->button_save = (GtkToolButton *) gtk_tool_button_new
04275     (gtk_image_new_from_icon_name ("document-save",
04276                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04277      gettext ("Save"));
04278   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
04279                     NULL);
04280
04281   // Creating the run button
04282   window->button_run = (GtkToolButton *) gtk_tool_button_new
04283     (gtk_image_new_from_icon_name ("system-run",
04284                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04285      gettext ("Run"));
04286   g_signal_connect (window->button_run, "clicked", window_run
      , NULL);
04287
04288   // Creating the options button
04289   window->button_options = (GtkToolButton *) gtk_tool_button_new
04290     (gtk_image_new_from_icon_name ("preferences-system",
04291                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04292      gettext ("Options"));
04293   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
04294
04295   // Creating the help button
04296   window->button_help = (GtkToolButton *) gtk_tool_button_new
04297     (gtk_image_new_from_icon_name ("help-browser",
04298                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04299      gettext ("Help"));
04300   g_signal_connect (window->button_help, "clicked", window_help
      , NULL);
04301
04302   // Creating the about button
04303   window->button_about = (GtkToolButton *) gtk_tool_button_new
04304     (gtk_image_new_from_icon_name ("help-about",
04305                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04306      gettext ("About"));
04307   g_signal_connect (window->button_about, "clicked", window_about
      , NULL);
04308
04309   // Creating the exit button
04310   window->button_exit = (GtkToolButton *) gtk_tool_button_new
04311     (gtk_image_new_from_icon_name ("application-exit",
04312                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04313      gettext ("Exit"));
04314   g_signal_connect (window->button_exit, "clicked", gtk_main_quit,
      NULL);
04315
04316   // Creating the buttons bar
04317   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04318   gtk_toolbar_insert
04319     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open
      ), 0);
04320   gtk_toolbar_insert
04321     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save
      ), 1);
04322   gtk_toolbar_insert
04323     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run
      ), 2);
04324   gtk_toolbar_insert
04325     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options
      ), 3);
04326   gtk_toolbar_insert
04327     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help
      ), 4);
04328   gtk_toolbar_insert
04329     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about
      ), 5);
04330   gtk_toolbar_insert
04331     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit
      ), 6);
04332   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04333
04334   // Creating the simulator program label and entry
04335   window->label_simulator
04336     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04337   window->button_simulator = (GtkFileChooserButton *)
04338     gtk_file_chooser_button_new (gettext ("Simulator program"),
04339                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04340   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator
      ),
04341                                gettext ("Simulator program executable file"));
04342
04343   // Creating the evaluator program label and entry
04344   window->check_evaluator = (GtkCheckButton *)
```

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

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

```
      label_relaxation),
04493                     0, NGRADIENTS + 2, 1, 1);
04494   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_relaxation),
04495                     1, NGRADIENTS + 2, 1, 1);
04496
04497   // Creating the array of algorithms
04498   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
04499   window->button_algorithm[0] = (GtkRadioButton *)
04500     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04501   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm
      [0]),
04502                                tip_algorithm[0]);
04503   gtk_grid_attach (window->grid_algorithm,
04504                    GTK_WIDGET (window->button_algorithm[0]), 0,
       0, 1, 1);
04505   g_signal_connect (window->button_algorithm[0], "clicked",
04506                     window_set_algorithm, NULL);
04507   for (i = 0; ++i < NALGORITHMS;)
04508     {
04509       window->button_algorithm[i] = (GtkRadioButton *)
04510         gtk_radio_button_new_with_mnemonic
04511         (gtk_radio_button_get_group (window->button_algorithm[0
      ]),
04512          label_algorithm[i]);
04513       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm
      [i]),
04514                                    tip_algorithm[i]);
04515       gtk_grid_attach (window->grid_algorithm,
04516                        GTK_WIDGET (window->button_algorithm[i])
      , 0, i, 1, 1);
04517       g_signal_connect (window->button_algorithm[i], "clicked",
04518                         window_set_algorithm, NULL);
04519     }
04520   gtk_grid_attach (window->grid_algorithm,
04521                    GTK_WIDGET (window->label_simulations), 0,
04522                    NALGORITHMS, 1, 1);
04523   gtk_grid_attach (window->grid_algorithm,
04524                    GTK_WIDGET (window->spin_simulations), 1,
      NALGORITHMS, 1, 1);
04525   gtk_grid_attach (window->grid_algorithm,
04526                    GTK_WIDGET (window->label_iterations), 0,
04527                    NALGORITHMS + 1, 1, 1);
04528   gtk_grid_attach (window->grid_algorithm,
04529                    GTK_WIDGET (window->spin_iterations), 1,
04530                    NALGORITHMS + 1, 1, 1);
04531   gtk_grid_attach (window->grid_algorithm,
04532                    GTK_WIDGET (window->label_tolerance), 0,
04533                    NALGORITHMS + 2, 1, 1);
04534   gtk_grid_attach (window->grid_algorithm,
04535                    GTK_WIDGET (window->spin_tolerance), 1,
04536                    NALGORITHMS + 2, 1, 1);
04537   gtk_grid_attach (window->grid_algorithm,
04538                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS
      + 3, 1, 1);
04539   gtk_grid_attach (window->grid_algorithm,
04540                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS +
      3, 1, 1);
04541   gtk_grid_attach (window->grid_algorithm,
04542                    GTK_WIDGET (window->label_population), 0,
04543                    NALGORITHMS + 4, 1, 1);
04544   gtk_grid_attach (window->grid_algorithm,
04545                    GTK_WIDGET (window->spin_population), 1,
04546                    NALGORITHMS + 4, 1, 1);
04547   gtk_grid_attach (window->grid_algorithm,
04548                    GTK_WIDGET (window->label_generations), 0,
04549                    NALGORITHMS + 5, 1, 1);
04550   gtk_grid_attach (window->grid_algorithm,
04551                    GTK_WIDGET (window->spin_generations), 1,
04552                    NALGORITHMS + 5, 1, 1);
04553   gtk_grid_attach (window->grid_algorithm,
04554                    GTK_WIDGET (window->label_mutation), 0,
04555                    NALGORITHMS + 6, 1, 1);
04556   gtk_grid_attach (window->grid_algorithm,
04557                    GTK_WIDGET (window->spin_mutation), 1,
04558                    NALGORITHMS + 6, 1, 1);
04559   gtk_grid_attach (window->grid_algorithm,
04560                    GTK_WIDGET (window->label_reproduction), 0
      ,
04561                    NALGORITHMS + 7, 1, 1);
04562   gtk_grid_attach (window->grid_algorithm,
04563                    GTK_WIDGET (window->spin_reproduction), 1,
04564                    NALGORITHMS + 7, 1, 1);
04565   gtk_grid_attach (window->grid_algorithm,
04566                    GTK_WIDGET (window->label_adaptation), 0,
04567                    NALGORITHMS + 8, 1, 1);
04568   gtk_grid_attach (window->grid_algorithm,
```

```
04569                   GTK_WIDGET (window->spin_adaptation), 1,
04570                   NALGORITHMS + 8, 1, 1);
04571   gtk_grid_attach (window->grid_algorithm,
04572                   GTK_WIDGET (window->check_gradient), 0,
04573                   NALGORITHMS + 9, 2, 1);
04574   gtk_grid_attach (window->grid_algorithm,
04575                   GTK_WIDGET (window->grid_gradient), 0,
04576                   NALGORITHMS + 10, 2, 1);
04577   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext
      ("Algorithm"));
04578   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04579                      GTK_WIDGET (window->grid_algorithm));
04580
04581   // Creating the variable widgets
04582   window->combo_variable = (GtkComboBoxText *)
      gtk_combo_box_text_new ();
04583   gtk_widget_set_tooltip_text
04584     (GTK_WIDGET (window->combo_variable), gettext ("Variables
       selector"));
04585   window->id_variable = g_signal_connect
04586     (window->combo_variable, "changed", window_set_variable
      , NULL);
04587   window->button_add_variable
04588     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04589                                                    GTK_ICON_SIZE_BUTTON);
04590   g_signal_connect
04591     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
04592   gtk_widget_set_tooltip_text
04593     (GTK_WIDGET (window->button_add_variable), gettext ("Add
       variable"));
04594   window->button_remove_variable
04595     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04596                                                    GTK_ICON_SIZE_BUTTON);
04597   g_signal_connect
04598     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
04599   gtk_widget_set_tooltip_text
04600     (GTK_WIDGET (window->button_remove_variable), gettext
       ("Remove variable"));
04601   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("
      Name"));
04602   window->entry_variable = (GtkEntry *) gtk_entry_new ();
04603   gtk_widget_set_tooltip_text
04604     (GTK_WIDGET (window->entry_variable), gettext ("Variable name
      "));
04605   window->id_variable_label = g_signal_connect
04606     (window->entry_variable, "changed", window_label_variable
      , NULL);
04607   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"))
      ;
04608   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04609     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION
      ]);
04610   gtk_widget_set_tooltip_text
04611     (GTK_WIDGET (window->spin_min),
04612      gettext ("Minimum initial value of the variable"));
04613   window->scrolled_min
04614     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04615   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04616                      GTK_WIDGET (window->spin_min));
04617   g_signal_connect (window->spin_min, "value-changed",
04618                     window_rangemin_variable, NULL);
04619   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"))
      ;
04620   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04621     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04622   gtk_widget_set_tooltip_text
04623     (GTK_WIDGET (window->spin_max),
04624      gettext ("Maximum initial value of the variable"));
04625   window->scrolled_max
04626     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04627   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04628                      GTK_WIDGET (window->spin_max));
04629   g_signal_connect (window->spin_max, "value-changed",
04630                     window_rangemax_variable, NULL);
04631   window->check_minabs = (GtkCheckButton *)
04632     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
04633   g_signal_connect (window->check_minabs, "toggled", window_update
      , NULL);
04634   window->spin_minabs = (GtkSpinButton *)
      gtk_spin_button_new_with_range
04635     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04636   gtk_widget_set_tooltip_text
04637     (GTK_WIDGET (window->spin_minabs),
04638      gettext ("Minimum allowed value of the variable"));
04639   window->scrolled_minabs
```

```
04640     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04641   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04642                      GTK_WIDGET (window->spin_minabs));
04643   g_signal_connect (window->spin_minabs, "value-changed",
04644                     window_rangeminabs_variable,
      NULL);
04645   window->check_maxabs = (GtkCheckButton *)
04646     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
04647   g_signal_connect (window->check_maxabs, "toggled", window_update
      , NULL);
04648   window->spin_maxabs = (GtkSpinButton *)
      gtk_spin_button_new_with_range
04649     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04650   gtk_widget_set_tooltip_text
04651     (GTK_WIDGET (window->spin_maxabs),
04652      gettext ("Maximum allowed value of the variable"));
04653   window->scrolled_maxabs
04654     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04655   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04656                      GTK_WIDGET (window->spin_maxabs));
04657   g_signal_connect (window->spin_maxabs, "value-changed",
04658                     window_rangemaxabs_variable,
      NULL);
04659   window->label_precision
04660     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04661   window->spin_precision = (GtkSpinButton *)
04662     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION
      , 1.);
04663   gtk_widget_set_tooltip_text
04664     (GTK_WIDGET (window->spin_precision),
04665      gettext ("Number of precision floating point digits\n"
04666               "0 is for integer numbers"));
04667   g_signal_connect (window->spin_precision, "value-changed",
04668                     window_precision_variable, NULL);
04669   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("
      Sweeps number"));
04670   window->spin_sweeps
04671     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04672   gtk_widget_set_tooltip_text
04673     (GTK_WIDGET (window->spin_sweeps),
04674      gettext ("Number of steps sweeping the variable"));
04675   g_signal_connect
04676     (window->spin_sweeps, "value-changed", window_update_variable
      , NULL);
04677   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits
       number"));
04678   window->spin_bits
04679     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
04680   gtk_widget_set_tooltip_text
04681     (GTK_WIDGET (window->spin_bits),
04682      gettext ("Number of bits to encode the variable"));
04683   g_signal_connect
04684     (window->spin_bits, "value-changed", window_update_variable
      , NULL);
04685   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step
       size"));
04686   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
04687     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04688   gtk_widget_set_tooltip_text
04689     (GTK_WIDGET (window->spin_step),
04690      gettext ("Initial step size for the gradient based method"));
04691   window->scrolled_step
04692     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04693   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
04694                      GTK_WIDGET (window->spin_step));
04695   g_signal_connect
04696     (window->spin_step, "value-changed", window_step_variable
      , NULL);
04697   window->grid_variable = (GtkGrid *) gtk_grid_new ();
04698   gtk_grid_attach (window->grid_variable,
04699                    GTK_WIDGET (window->combo_variable), 0, 0, 2,
      1);
04700   gtk_grid_attach (window->grid_variable,
04701                    GTK_WIDGET (window->button_add_variable),
       2, 0, 1, 1);
04702   gtk_grid_attach (window->grid_variable,
04703                    GTK_WIDGET (window->button_remove_variable
      ), 3, 0, 1, 1);
04704   gtk_grid_attach (window->grid_variable,
04705                    GTK_WIDGET (window->label_variable), 0, 1, 1,
      1);
04706   gtk_grid_attach (window->grid_variable,
04707                    GTK_WIDGET (window->entry_variable), 1, 1, 3,
      1);
04708   gtk_grid_attach (window->grid_variable,
04709                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
04710   gtk_grid_attach (window->grid_variable,
```

```
04711                     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04712   gtk_grid_attach (window->grid_variable,
04713                     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04714   gtk_grid_attach (window->grid_variable,
04715                     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04716   gtk_grid_attach (window->grid_variable,
04717                     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
04718   gtk_grid_attach (window->grid_variable,
04719                     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3
      , 1);
04720   gtk_grid_attach (window->grid_variable,
04721                     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04722   gtk_grid_attach (window->grid_variable,
04723                     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3
      , 1);
04724   gtk_grid_attach (window->grid_variable,
04725                     GTK_WIDGET (window->label_precision), 0, 6, 1
      , 1);
04726   gtk_grid_attach (window->grid_variable,
04727                     GTK_WIDGET (window->spin_precision), 1, 6, 3,
      1);
04728   gtk_grid_attach (window->grid_variable,
04729                     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04730   gtk_grid_attach (window->grid_variable,
04731                     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04732   gtk_grid_attach (window->grid_variable,
04733                     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04734   gtk_grid_attach (window->grid_variable,
04735                     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04736   gtk_grid_attach (window->grid_variable,
04737                     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
04738   gtk_grid_attach (window->grid_variable,
04739                     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1)
      ;
04740   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("
      Variable"));
04741   gtk_container_add (GTK_CONTAINER (window->frame_variable),
04742                      GTK_WIDGET (window->grid_variable));
04743
04744   // Creating the experiment widgets
04745   window->combo_experiment = (GtkComboBoxText *)
      gtk_combo_box_text_new ();
04746   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment
      ),
04747                                gettext ("Experiment selector"));
04748   window->id_experiment = g_signal_connect
04749     (window->combo_experiment, "changed", window_set_experiment
      , NULL);
04750   window->button_add_experiment
04751     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04752                                                    GTK_ICON_SIZE_BUTTON);
04753   g_signal_connect
04754     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
04755   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment
      ),
04756                                gettext ("Add experiment"));
04757   window->button_remove_experiment
04758     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04759                                                    GTK_ICON_SIZE_BUTTON);
04760   g_signal_connect (window->button_remove_experiment, "
      clicked",
04761                     window_remove_experiment, NULL);
04762   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment
      ),
04763                                gettext ("Remove experiment"));
04764   window->label_experiment
04765     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04766   window->button_experiment = (GtkFileChooserButton *)
04767     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04768                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04769   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment
      ),
04770                                gettext ("Experimental data file"));
04771   window->id_experiment_name
04772     = g_signal_connect (window->button_experiment, "
      selection-changed",
04773                         window_name_experiment, NULL);
04774   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("
      Weight"));
04775   window->spin_weight
04776     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04777   gtk_widget_set_tooltip_text
04778     (GTK_WIDGET (window->spin_weight),
04779      gettext ("Weight factor to build the objective function"));
04780   g_signal_connect
04781     (window->spin_weight, "value-changed", window_weight_experiment
```

```
      , NULL);
04782   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04783   gtk_grid_attach (window->grid_experiment,
04784                    GTK_WIDGET (window->combo_experiment), 0, 0,
       2, 1);
04785   gtk_grid_attach (window->grid_experiment,
04786                    GTK_WIDGET (window->button_add_experiment
      ), 2, 0, 1, 1);
04787   gtk_grid_attach (window->grid_experiment,
04788                    GTK_WIDGET (window->button_remove_experiment
      ), 3, 0, 1, 1);
04789   gtk_grid_attach (window->grid_experiment,
04790                    GTK_WIDGET (window->label_experiment), 0, 1,
       1, 1);
04791   gtk_grid_attach (window->grid_experiment,
04792                    GTK_WIDGET (window->button_experiment), 1,
      1, 3, 1);
04793   gtk_grid_attach (window->grid_experiment,
04794                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04795   gtk_grid_attach (window->grid_experiment,
04796                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04797   for (i = 0; i < MAX_NINPUTS; ++i)
04798     {
04799       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04800       window->check_template[i] = (GtkCheckButton *)
04801         gtk_check_button_new_with_label (buffer3);
04802       window->id_template[i]
04803         = g_signal_connect (window->check_template[i], "toggled",
04804                             window_inputs_experiment,
      NULL);
04805       gtk_grid_attach (window->grid_experiment,
04806                        GTK_WIDGET (window->check_template[i]), 0,
       3 + i, 1, 1);
04807       window->button_template[i] = (GtkFileChooserButton *)
04808         gtk_file_chooser_button_new (gettext ("Input template"),
04809                                      GTK_FILE_CHOOSER_ACTION_OPEN);
04810       gtk_widget_set_tooltip_text
04811         (GTK_WIDGET (window->button_template[i]),
04812          gettext ("Experimental input template file"));
04813       window->id_input[i]
04814         = g_signal_connect_swapped (window->button_template[i],
04815                                     "selection-changed",
04816                                     (void (*)) window_template_experiment
      ,
04817                                     (void *) (size_t) i);
04818       gtk_grid_attach (window->grid_experiment,
04819                        GTK_WIDGET (window->button_template[i]),
      1, 3 + i, 3, 1);
04820     }
04821   window->frame_experiment
04822     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04823   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04824                      GTK_WIDGET (window->grid_experiment));
04825
04826   // Creating the grid and attaching the widgets to the grid
04827   window->grid = (GtkGrid *) gtk_grid_new ();
04828   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons
      ), 0, 0, 3, 1);
04829   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files),
       0, 1, 3, 1);
04830   gtk_grid_attach (window->grid,
04831                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1
      , 1);
04832   gtk_grid_attach (window->grid,
04833                    GTK_WIDGET (window->frame_variable), 1, 2, 1,
      1);
04834   gtk_grid_attach (window->grid,
04835                    GTK_WIDGET (window->frame_experiment), 2, 2,
       1, 1);
04836   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
04837
04838   // Setting the window logo
04839   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04840   gtk_window_set_icon (window->window, window->logo);
04841
04842   // Showing the window
04843   gtk_widget_show_all (GTK_WIDGET (window->window));
04844
04845   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04846 #if GTK_MINOR_VERSION >= 16
04847   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -
      1, 40);
04848   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -
      1, 40);
04849   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs
      ), -1, 40);
```

```
04850   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs
      ), -1, 40);
04851   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step),
       -1, 40);
04852 #endif
04853
04854   // Reading initial example
04855   input_new ();
04856   buffer2 = g_get_current_dir ();
04857   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE
      , NULL);
04858   g_free (buffer2);
04859   window_read (buffer);
04860   g_free (buffer);
04861
04862 #if DEBUG
04863   fprintf (stderr, "window_new: start\n");
04864 #endif
04865 }
04866
04867 #endif
04868
04874 int
04875 cores_number ()
04876 {
04877 #ifdef G_OS_WIN32
04878   SYSTEM_INFO sysinfo;
04879   GetSystemInfo (&sysinfo);
04880   return sysinfo.dwNumberOfProcessors;
04881 #else
04882   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04883 #endif
04884 }
04885
04895 int
04896 main (int argn, char **argc)
04897 {
04898 #if HAVE_GTK
04899   char *buffer;
04900 #endif
04901
04902   // Starting pseudo-random numbers generator
04903   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04904   calibrate->seed = DEFAULT_RANDOM_SEED;
04905
04906   // Allowing spaces in the XML data file
04907   xmlKeepBlanksDefault (0);
04908
04909   // Starting MPI
04910 #if HAVE_MPI
04911   MPI_Init (&argn, &argc);
04912   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04913   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04914   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04915 #else
04916   ntasks = 1;
04917 #endif
04918
04919 #if HAVE_GTK
04920
04921   // Getting threads number
04922   nthreads_gradient = nthreads = cores_number
       ();
04923
04924   // Setting local language and international floating point numbers notation
04925   setlocale (LC_ALL, "");
04926   setlocale (LC_NUMERIC, "C");
04927   window->application_directory = g_get_current_dir ();
04928   buffer = g_build_filename (window->application_directory
      , LOCALE_DIR, NULL);
04929   bindtextdomain (PROGRAM_INTERFACE, buffer);
04930   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04931   textdomain (PROGRAM_INTERFACE);
04932
04933   // Initing GTK+
04934   gtk_disable_setlocale ();
04935   gtk_init (&argn, &argc);
04936
04937   // Opening the main window
04938   window_new ();
04939   gtk_main ();
04940
04941   // Freeing memory
04942   input_free ();
04943   g_free (buffer);
04944   gtk_widget_destroy (GTK_WIDGET (window->window));
04945   g_free (window->application_directory);
```

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

5.7 mpcotool.h File Reference

Header file of the mpcotool.

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

Data Structures

- struct Input

  Struct to define the calibration input file.
- struct Calibrate

  Struct to define the calibration data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

  Enum to define the algorithms.
- enum GradientMethod { GRADIENT_METHOD_COORDINATES = 0, GRADIENT_METHOD_RANDOM = 1 }

  Enum to define the methods to estimate the gradient.

Functions

- void show_message (char *title, char *msg, int type)

  Function to show a dialog with a message.
- void show_error (char *msg)

  Function to show a dialog with an error message.
- int xml_node_get_int (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an integer number of a XML node property.
- unsigned int xml_node_get_uint (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get an unsigned integer number of a XML node property.
- unsigned int xml_node_get_uint_with_default (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)

  Function to get an unsigned integer number of a XML node property with a default value.
- double xml_node_get_float (xmlNode *node, const xmlChar *prop, int *error_code)

  Function to get a floating point number of a XML node property.
- double xml_node_get_float_with_default (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)

  Function to get a floating point number of a XML node property with a default value.
- void xml_node_set_int (xmlNode *node, const xmlChar *prop, int value)

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

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

  Function to set a floating point number in a XML node property.
- void input_new ()

  Function to create a new Input struct.
- void input_free ()

  Function to free the memory of the input file data.
- int input_open (char *filename)

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

  Function to write the simulation input file.
- double calibrate_parse (unsigned int simulation, unsigned int experiment)

  Function to parse input files, simulating and calculating the \ objective function.
- void calibrate_print ()

  Function to print the results.
- void calibrate_save_variables (unsigned int simulation, double error)

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

  Function to save the best simulations.
- void calibrate_sequential ()

  Function to calibrate sequentially.
- void * calibrate_thread (ParallelData *data)

  Function to calibrate on a thread.
- void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 calibration results.
- void calibrate_synchronise ()

  Function to synchronise the calibration results of MPI tasks.
- void calibrate_sweep ()

  Function to calibrate with the sweep algorithm.
- void calibrate_MonteCarlo ()

Function to calibrate with the Monte-Carlo algorithm.

- void calibrate_best_gradient (unsigned int simulation, double value)

  Function to save the best simulation in a gradient based method.
- void **calibrate_gradient_sequential** ()
- 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.

```
{
  ALGORITHM_MONTE_CARLO = 0,
  ALGORITHM_SWEEP = 1,
  ALGORITHM_GENETIC = 2
};
```

5.7.2.2 enum GradientMethod

Enum to define the methods to estimate the gradient.

Enumerator:

GRADIENT_METHOD_COORDINATES Coordinates descent method.

GRADIENT_METHOD_RANDOM Random method.

Definition at line 54 of file mpcotool.h.

```
{
  GRADIENT_METHOD_COORDINATES = 0,
  GRADIENT_METHOD_RANDOM = 1,
};
```

5.7.3 Function Documentation

5.7.3.1 void calibrate_best (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1443 of file mpcotool.c.

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

```
              calibrate->simulation_best[i - 1] = j;
              calibrate->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG
  fprintf (stderr, "calibrate_best: end\n");
#endif
}
```

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

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

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1756 of file mpcotool.c.

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

5.7.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

Returns

objective function value.

Definition at line 2059 of file mpcotool.c.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG
  fprintf (stderr, "calibrate_genetic_objective: start\n");
#endif
  for (j = 0; j < calibrate->nvariables; ++j)
    {
```

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

Here is the call graph for this function:

5.7.3.4 void* calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1821 of file mpcotool.c.

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

Here is the call graph for this function:

5.7.3.5 void calibrate_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

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

Definition at line 1196 of file mpcotool.c.

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

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

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

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

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

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

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

```
  fclose (file);

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

5.7.3.6 void calibrate_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 calibration results.

Parameters

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

Definition at line 1561 of file mpcotool.c.

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

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

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1283 of file mpcotool.c.

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

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

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

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

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

```
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

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

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

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

Here is the call graph for this function:

5.7.3.8 void calibrate_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1415 of file mpcotool.c.

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

5.7.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

simulation	Simulation number.

Definition at line 1923 of file mpcotool.c.

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

Here is the call graph for this function:

5.7.3.10 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1517 of file mpcotool.c.

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

Here is the call graph for this function:

5.7.3.11 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 548 of file mpcotool.c.

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

```
  char *msg;
  int error_code;
  unsigned int i;

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

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

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

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

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

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

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

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

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

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

```
else if (!xmlStrcmp (buffer, XML_GENETIC))
  {
    input->algorithm = ALGORITHM_GENETIC;

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

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

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

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

    // Obtaining adaptation probability
    if (xmlHasProp (node, XML_ADAPTATION))
      {
        input->adaptation_ratio
          = xml_node_get_float (node, XML_ADAPTATION
    , &error_code);
        if (error_code || input->adaptation_ratio < 0.
            || input->adaptation_ratio >= 1.)
```

```
            {
              msg = gettext ("Invalid adaptation probability");
              goto exit_on_error;
            }
        }
      else
        {
          msg = gettext ("No adaptation probability");
          goto exit_on_error;
        }

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

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

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

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

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

      // Getting gradient method parameters
      if (xmlHasProp (node, XML_NSTEPS))
        {
          input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
          if (error_code || !input->nsteps)
            {
              msg = gettext ("Invalid steps number");
              goto exit_on_error;
            }
          buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
          if (!xmlStrcmp (buffer, XML_COORDINATES))
            input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, XML_RANDOM))
            {
              input->gradient_method =
```

```
      GRADIENT_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, XML_NESTIMATES
      , &error_code);
              if (error_code || !input->nestimates)
                {
                  msg = gettext ("Invalid estimates number");
                  goto exit_on_error;
                }
            }
          else
            {
              msg = gettext ("Unknown method to estimate the gradient");
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (
      node, XML_RELAXATION,
                                             DEFAULT_RELAXATION
      , &error_code);
          if (error_code || input->relaxation < 0. || input
      ->relaxation > 2.)
            {
              msg = gettext ("Invalid relaxation parameter");
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }

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

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

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

```
              goto exit_on_error;
            }
          input->rangemaxabs[input->nvariables]
            = xml_node_get_float_with_default (
        child, XML_ABSOLUTE_MAXIMUM,
                                             G_MAXDOUBLE, &error_code);
          if (error_code)
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("bad absolute maximum"));
              msg = buffer2;
              goto exit_on_error;
            }
          if (input->rangemax[input->nvariables]
              > input->rangemaxabs[input->nvariables
        ])
            {
              snprintf (buffer2, 64, "%s %s: %s",
                        gettext ("Variable"),
                        buffer, gettext ("maximum range not allowed"));
              msg = buffer2;
              goto exit_on_error;
            }
        }
      else
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("no maximum range"))
      ;
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->rangemax[input->nvariables]
          < input->rangemin[input->nvariables])
        {
          snprintf (buffer2, 64, "%s %s: %s",
                    gettext ("Variable"), buffer, gettext ("bad range"));
          msg = buffer2;
          goto exit_on_error;
        }
      input->precision = g_realloc
        (input->precision, (1 + input->nvariables)
       * sizeof (unsigned int));
      input->precision[input->nvariables]
        = xml_node_get_uint_with_default (child,
      XML_PRECISION,
                                        DEFAULT_PRECISION, &
      error_code);
      if (error_code || input->precision[input->nvariables
      ] >= NPRECISIONS)
        {
          snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                    gettext ("bad precision"));
          msg = buffer2;
          goto exit_on_error;
        }
      if (input->algorithm == ALGORITHM_SWEEP)
        {
          if (xmlHasProp (child, XML_NSWEEPS))
            {
              input->nsweeps = (unsigned int *)
                g_realloc (input->nsweeps,
                           (1 + input->nvariables) * sizeof (
      unsigned int));
              input->nsweeps[input->nvariables]
                = xml_node_get_uint (child, XML_NSWEEPS
      , &error_code);
              if (error_code || !input->nsweeps[input->
      nvariables])
                {
                  snprintf (buffer2, 64, "%s %s: %s",
                            gettext ("Variable"),
                            buffer, gettext ("bad sweeps"));
                  msg = buffer2;
                  goto exit_on_error;
                }
            }
          else
            {
              snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
                        gettext ("no sweeps number"));
              msg = buffer2;
              goto exit_on_error;
            }
#if DEBUG
          fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
                   input->nsweeps[input->nvariables]
```

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

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

  // Closing the XML document
  xmlFreeDoc (doc);

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

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

Here is the call graph for this function:

5.7.3.12 void show_error (char * msg)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 256 of file mpcotool.c.

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file mpcotool.c.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

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

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

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

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

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

5.7.3.14 double xml_node_get_float (xmlNode * node, const xmlChar * prop, int * error_code)

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

Parameters

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

Returns

Floating point number value.

Definition at line 366 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 400 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Returns

Integer number value.

Definition at line 274 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 305 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 339 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Definition at line 463 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 425 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 444 of file mpcotool.c.

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

5.8 mpcotool.h

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

```
00080   double *weight;
00081   double *step;
00082   unsigned int *precision;
00083   unsigned int *nsweeps;
00084   unsigned int *nbits;
00086   double tolerance;
00087   double mutation_ratio;
00088   double reproduction_ratio;
00089   double adaptation_ratio;
00090   double relaxation;
00091   unsigned long int seed;
00093   unsigned int nvariables;
00094   unsigned int nexperiments;
00095   unsigned int ninputs;
00096   unsigned int nsimulations;
00097   unsigned int algorithm;
00098   unsigned int nsteps;
00100   unsigned int gradient_method;
00101   unsigned int nestimates;
00103   unsigned int niterations;
00104   unsigned int nbest;
00105 } Input;
00106
00111 typedef struct
00112 {
00113   GMappedFile **file[MAX_NINPUTS];
00114   char **template[MAX_NINPUTS];
00115   char **experiment;
00116   char **label;
00117   gsl_rng *rng;
00118   GeneticVariable *genetic_variable;
00120   FILE *file_result;
00121   FILE *file_variables;
00122   char *result;
00123   char *variables;
00124   char *simulator;
00125   char *evaluator;
00127   double *value;
00128   double *rangemin;
00129   double *rangemax;
00130   double *rangeminabs;
00131   double *rangemaxabs;
00132   double *error_best;
00133   double *weight;
00134   double *step;
00135   double *gradient;
00136   double *value_old;
00138   double *error_old;
00140   unsigned int *precision;
00141   unsigned int *nsweeps;
00142   unsigned int *thread;
00144   unsigned int *thread_gradient;
00147   unsigned int *simulation_best;
00148   double tolerance;
00149   double mutation_ratio;
00150   double reproduction_ratio;
00151   double adaptation_ratio;
00152   double relaxation;
00153   double calculation_time;
00154   unsigned long int seed;
00156   unsigned int nvariables;
00157   unsigned int nexperiments;
00158   unsigned int ninputs;
00159   unsigned int nsimulations;
00160   unsigned int gradient_method;
00161   unsigned int nsteps;
00163   unsigned int nestimates;
00165   unsigned int algorithm;
00166   unsigned int nstart;
00167   unsigned int nend;
00168   unsigned int nstart_gradient;
00170   unsigned int nend_gradient;
00172   unsigned int niterations;
00173   unsigned int nbest;
00174   unsigned int nsaveds;
00175 #if HAVE_MPI
00176   int mpi_rank;
00177 #endif
00178 } Calibrate;
00179
00184 typedef struct
00185 {
00186   unsigned int thread;
00187 } ParallelData;
00188
00189 // Public functions
00190 void show_message (char *title, char *msg, int type);
```

```
00191 void show_error (char *msg);
00192 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int
       *error_code);
00193 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar
      * prop,
00194                                 int *error_code);
00195 unsigned int xml_node_get_uint_with_default (
      xmlNode * node,
00196                                              const xmlChar * prop,
00197                                              unsigned int default_value,
00198                                              int *error_code);
00199 double xml_node_get_float (xmlNode * node, const xmlChar *
      prop,
00200                            int *error_code);
00201 double xml_node_get_float_with_default (xmlNode
      * node, const xmlChar * prop,
00202                                         double default_value, int *error_code);
00203 void xml_node_set_int (xmlNode * node, const xmlChar * prop,
      int value);
00204 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00205                         unsigned int value);
00206 void xml_node_set_float (xmlNode * node, const xmlChar * prop
      , double value);
00207 void input_new ();
00208 void input_free ();
00209 int input_open (char *filename);
00210 void calibrate_input (unsigned int simulation, char *input,
00211                       GMappedFile * template);
00212 double calibrate_parse (unsigned int simulation, unsigned int
      experiment);
00213 void calibrate_print ();
00214 void calibrate_save_variables (unsigned int simulation,
       double error);
00215 void calibrate_best (unsigned int simulation, double value);
00216 void calibrate_sequential ();
00217 void *calibrate_thread (ParallelData * data);
00218 void calibrate_merge (unsigned int nsaveds, unsigned int *
      simulation_best,
00219                       double *error_best);
00220 #if HAVE_MPI
00221 void calibrate_synchronise ();
00222 #endif
00223 void calibrate_sweep ();
00224 void calibrate_MonteCarlo ();
00225 void calibrate_best_gradient (unsigned int simulation,
      double value);
00226 void calibrate_gradient_sequential ();
00227 void *calibrate_gradient_thread (ParallelData
       * data);
00228 double calibrate_variable_step_gradient (unsigned int variable);
00229 void calibrate_step_gradient (unsigned int simulation);
00230 void calibrate_gradient ();
00231 double calibrate_genetic_objective (Entity * entity)
      ;
00232 void calibrate_genetic ();
00233 void calibrate_save_old ();
00234 void calibrate_merge_old ();
00235 void calibrate_refine ();
00236 void calibrate_step ();
00237 void calibrate_iterate ();
00238 void calibrate_open ();
00239
00240 #endif
```

Index