Calibrator

1.2.4

Generated by Doxygen 1.8.9.1

Fri Jan 8 2016 01:32:37

Contents

Chapter 1

MPCOTool

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

VERSIONS

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

AUTHORS

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

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

- gcc or clang (to compile the source code)
- make (to build the executable file)
- autoconf (to generate the Makefile in different operative systems)
- automake (to check the operative system)
- pkg-config (to find the libraries to compile)
- gsl (to generate random numbers)
- libxml (to deal with XML files)
- glib (extended utilities of C to work with data, lists, mapped files, regular expressions, using multicores in shared memory machines, ...)
- genetic (genetic algorithm)

OPTIONAL TOOLS AND LIBRARIES

- gettext (to work with different locales)
- gtk+ (to create the interactive GUI tool)
- openmpi or mpich (to run in parallelized tasks on multiple computers)

- `doxygen` (standard comments format to generate documentation)
- `latex` (to build the PDF manuals)

FILES

The source code has to have the following files:

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

BUILDING INSTRUCTIONS

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

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

FreeBSD 10.2

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

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

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

2. Download this repository:

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

3. Link the latest genetic version to genetic:

   $ cd mpcotool/1.2.4
   $ ln -s ../../genetic/0.6.1 genetic

4. Build doing on a terminal:

   $ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

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

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

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

   $ make windist

Fedora Linux 23

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

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

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

MAKING MANUALS INSTRUCTIONS

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

$ make manuals

MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory mpcotool/1.2.4):

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

2. Build all tests doing in the same terminal:

```
$ cd ../../1.2.4
$ make tests
```

USER INSTRUCTIONS

- Command line in sequential mode:

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

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

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

- The syntax of the simulator has to be:

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

- The syntax of the program to evaluate the objetive function has to be (where the first data in the results file has to be the objective function value):

```
$ ./evaluator_name simulated_file data_file results_file
```

- On UNIX type systems the GUI application can be open doing on a terminal:

```
$ ./mpcotool
```

INPUT FILE FORMAT

The format of the main input file is as:

“‘xml <?xml version="1.0"?> <calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm↩
_type" nsimulations="simulations_number" niterations="iterations_number" tolerance="tolerance_value" nbest="best↩
_number" npopulation="population_number" ngenerations="generations_number" mutation="mutation_↩
ratio" reproduction="reproduction_ratio" adaptation="adaptation_ratio" gradient_type="gradient_method_type"
nsteps="steps_number" relaxation="relaxation_paramter" nestimates="estimates_number" seed="random_↩
seed" result="result_file" variables="variables_file"> <experiment name="data_file_1" template1="template_1_1"
template2="template_1_2" ... weight="weight_1"/> ... <experiment name="data_file_N" template1="template↩
_N_1" template2="template_N_2" ... weight="weight_N"/> <variable name="variable_1" minimum="min_value"
maximum="max_value" precision="precision_digits" sweeps="sweeps_number" nbits="bits_number" step="step↩
_size"> ... <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_↩
digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> </calibrate> “‘

with:

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

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

Implemented algorithms are:

- **sweep**: Sweep brute force algorithm. It requires for each variable:
  - *sweeps*: number of sweeps to generate for each variable in every experiment.
    The total number of simulations to run is:
    (number of experiments) x (variable 1 number of sweeps) x ... x (variable n number of sweeps) x (number of iterations)
- **Monte-Carlo**: Monte-Carlo brute force algorithm. It requires on calibrate:
  - *nsimulations*: number of simulations to run in every experiment.
    The total number of simulations to run is:
    (number of experiments) x (number of simulations) x (number of iterations)
- Both brute force algorithms can be iterated to improve convergence by using the following parameters:
  - *nbest*: number of best simulations to calculate convergence interval on next iteration (default 1).
  - *tolerance*: tolerance parameter to increase convergence interval (default 0).
  - *niterations*: number of iterations (default 1).
    It multiplies the total number of simulations:
    x (number of iterations)
- Moreover, both brute force algorithms can be coupled with a gradient based method by using:
  - *gradient_type*: method to estimate the gradient. Two options are currently available:
    - coordinates: coordinates descent method.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x 2 x (number of variables)
    - random: random method. It requires:
    - nestimates: number of random checks to estimate the gradient.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x (number of estimates)

  Both methods require also:
  - nsteps: number of steps to perform the gradient based method,
  - relaxation: relaxation parameter,

  and for each variable:
  - step: initial step size for the gradient based method.
- **genetic**: Genetic algorithm. It requires the following parameters:
  - *npopulation*: number of population.
  - *ngenerations*: number of generations.
  - *mutation*: mutation ratio.
  - *reproduction*: reproduction ratio.
  - *adaptation*: adaptation ratio.

and for each variable:

- *nbits*: number of bits to encode each variable.

The total number of simulations to run is:

(number of experiments) x (npopulation) x [1 + (ngenerations - 1) x (mutation + reproduction + adaptation)]

SOME EXAMPLES OF INPUT FILES

Example 1

- The simulator program name is: *pivot*
- The syntax is:

  $ ./pivot input_file output_file

- The program to evaluate the objective function is: *compare*
- The syntax is:

  $ ./compare simulated_file data_file result_file

- The calibration is performed with a *sweep brute force algorithm*.
- The experimental data files are:

  27-48.txt
  42.txt
  52.txt
  100.txt

- Templates to get input files to simulator for each experiment are:

  template1.js
  template2.js
  template3.js
  template4.js

- The variables to calibrate, ranges, precision and sweeps number to perform are:

  alpha1, [179.70, 180.20], 2, 5
  alpha2, [179.30, 179.60], 2, 5
  random, [0.00, 0.20], 2, 5
  boot-time, [0.0, 3.0], 1, 5

- Then, the number of simulations to run is: 4x5x5x5x5=2500.
- The input file is:

“‘xml <?xml version="1.0"?> <calibrate simulator="pivot" evaluator="compare" algorithm="sweep"> <experiment name="27-48.txt" template1="template1.js"> <experiment name="42.txt" template1="template2.js"> <experiment name="52.txt" template1="template3.js"> <experiment name="100.txt" template1="template4.js"> <variable name="alpha1" minimum="179.70" maximum="180.20" precision="2" nsweeps="5"> <variable name="alpha2" minimum="179.30" maximum="179.60" precision="2" nsweeps="5"> <variable name="random" minimum="0.↩ 00" maximum="0.20" precision="2" nsweeps="5"> <variable name="boot-time" minimum="0.0" maximum="3.0" precision="1" nsweeps="5"> </calibrate> “‘

- A template file as *template1.js*:

“‘ { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.02824, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03008, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03753, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

- produces simulator input files to reproduce the experimental data file *27-48.txt* as:

“‘json { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.02824, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03008, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03753, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

`#include <mpcotool.h>`

Data Fields

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

Array of maximum variable values.

- double * rangeminabs

  Array of absolute minimum variable values.
- double * rangemaxabs

  Array of absolute maximum variable values.
- double * error_best

  Array of the best minimum errors.
- double * weight

  Array of the experiment weights.
- double * step

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

  Vector of gradient estimation.
- double * value_old

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

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

  Array of variable precisions.
- unsigned int * nsweeps

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

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

  Array of best simulation numbers.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- double calculation_time

  Calculation time.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int gradient_method

  Method to estimate the gradient.
- unsigned int nsteps

Number of steps for the gradient based method.

- unsigned int nestimates

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

  Algorithm type.
- unsigned int nstart

  Beginning simulation number of the task.
- unsigned int nend

  Ending simulation number of the task.
- unsigned int nstart_gradient

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

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 111 of file mpcotool.h.

4.1.2 Field Documentation

4.1.2.1 unsigned int* Calibrate::thread_gradient

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

Definition at line 144 of file mpcotool.h.

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

- mpcotool.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

- char * template [MAX_NINPUTS]

  Array of input template names.
- char * name

  File name.
- double weight

  Weight to calculate the objective function value.

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <mpcotool.h>`

Data Fields

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

- unsigned int * nbits

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

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int algorithm

  Algorithm type.
- unsigned int nsteps

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

  Method to estimate the gradient.
- unsigned int nestimates

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 64 of file mpcotool.h.

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

- mpcotool.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_seed

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

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

  Threads number GtkLabel.
- GtkSpinButton * spin_threads

  Threads number GtkSpinButton.
- GtkLabel * label_gradient

  Gradient threads number GtkLabel.
- GtkSpinButton * spin_gradient

  Gradient threads number GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 76 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <mpcotool.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 184 of file mpcotool.h.

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

- mpcotool.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkLabel * label

  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 94 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label

  Variable label.
- double rangemin

  Minimum value.
- double rangemax

  Maximum value.
- double rangeminabs

  Minimum allowed value.
- double rangemaxabs

  Maximum allowed value.
- double step

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

  Precision digits.
- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file interface.h.

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

- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:

Data Fields

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

Result file GtkEntry.

- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

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

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

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton * check_gradient

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

  GtkGrid to pack the gradient based method widgets.

- GtkRadioButton * button_gradient [NGRADIENTS]

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

  GtkLabel to set the steps number.
- GtkSpinButton * spin_steps

  GtkSpinButton to set the steps number.
- GtkLabel * label_estimates

  GtkLabel to set the estimates number.
- GtkSpinButton * spin_estimates

  GtkSpinButton to set the estimates number.
- GtkLabel * label_relaxation

  GtkLabel to set the relaxation parameter.
- GtkSpinButton * spin_relaxation

  GtkSpinButton to set the relaxation parameter.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.
- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

Absolute maximum GtkScrolledWindow.

- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkLabel * label_step

  GtkLabel to set the step.
- GtkSpinButton * spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow * scrolled_step

  step GtkScrolledWindow.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

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

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

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

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

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.

- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

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

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

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

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 104 of file interface.h.

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

- interface.h

Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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

5.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
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 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00064 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00066 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00068 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00070 #define XML_COORDINATES (const xmlChar*)"coordinates"
00072 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00074 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00076 #define XML_GENETIC (const xmlChar*)"genetic"
00078 #define XML_GRADIENT_METHOD (const xmlChar*)"gradient_method"
00079 #define XML_MINIMUM (const xmlChar*)"minimum"
00081 #define XML_MAXIMUM (const xmlChar*)"maximum"
00082 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00083 #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 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00091 #define XML_NITERATIONS (const xmlChar*)"niterations"
00093 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00095 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00097 #define XML_NSTEPS (const xmlChar*)"nsteps"
00099 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00100 #define XML_PRECISION (const xmlChar*)"precision"
00101 #define XML_RANDOM (const xmlChar*)"random"
00103 #define XML_RELAXATION (const xmlChar*)"relaxation"
00104 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00106 #define XML_RESULT (const xmlChar*)"result"
00108 #define XML_SIMULATOR (const xmlChar*)"simulator"
00109 #define XML_SEED (const xmlChar*)"seed"
00111 #define XML_STEP (const xmlChar*)"step"
00112 #define XML_SWEEP (const xmlChar*)"sweep"
00113 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00114 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00116 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00118 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00120 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00122 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00124 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00126 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00128 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00130 #define XML_VARIABLE (const xmlChar*)"variable"
00132 #define XML_VARIABLES (const xmlChar*)"variables"
00133 #define XML_WEIGHT (const xmlChar*)"weight"
00135
00136 #endif
```

5.3 interface.h File Reference

Header file of the interface.

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

Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- void input_save (char *filename)

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

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

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

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to add a variable in the main window.

- void window_label_variable ()

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

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

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

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

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

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

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

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

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

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

  Function to obtain the cores number.

5.3.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.3.2 Function Documentation

5.3.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4876 of file mpcotool.c.

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

5.3.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 2700 of file mpcotool.c.

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

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

Here is the call graph for this function:

5.3.2.3 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2933 of file mpcotool.c.

```
02934 {
02935   unsigned int i;
02936 #if DEBUG
02937   fprintf (stderr, "window_get_algorithm: start\n");
02938 #endif
02939   for (i = 0; i < NALGORITHMS; ++i)
```

```
02940     if (gtk_toggle_button_get_active
02941         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02942       break;
02943 #if DEBUG
02944   fprintf (stderr, "window_get_algorithm: %u\n", i);
02945   fprintf (stderr, "window_get_algorithm: end\n");
02946 #endif
02947   return i;
02948 }
```

5.3.2.4 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2956 of file mpcotool.c.

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

5.3.2.5 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 4053 of file mpcotool.c.

```
04054 {
04055   unsigned int i;
04056   char *buffer;
04057 #if DEBUG
04058   fprintf (stderr, "window_read: start\n");
04059 #endif
04060
04061   // Reading new input file
04062   input_free ();
04063   if (!input_open (filename))
04064     return 0;
04065
04066   // Setting GTK+ widgets data
04067   gtk_entry_set_text (window->entry_result, input->result);
04068   gtk_entry_set_text (window->entry_variables, input->
      variables);
04069   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04070   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04071                                  (window->button_simulator), buffer);
```

```
04072   g_free (buffer);
04073   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04074                                 (size_t) input->evaluator);
04075   if (input->evaluator)
04076     {
04077       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04078       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04079                                      (window->button_evaluator), buffer);
04080       g_free (buffer);
04081     }
04082   gtk_toggle_button_set_active
04083     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04084   switch (input->algorithm)
04085     {
04086     case ALGORITHM_MONTE_CARLO:
04087       gtk_spin_button_set_value (window->spin_simulations,
04088                                  (gdouble) input->nsimulations);
04089     case ALGORITHM_SWEEP:
04090       gtk_spin_button_set_value (window->spin_iterations,
04091                                  (gdouble) input->niterations);
04092       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04093       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04094       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04095                                     input->nsteps);
04096       if (input->nsteps)
04097         {
04098           gtk_toggle_button_set_active
04099             (GTK_TOGGLE_BUTTON (window->button_gradient
04100                                 [input->gradient_method]), TRUE);
04101           gtk_spin_button_set_value (window->spin_steps,
04102                                      (gdouble) input->nsteps);
04103           gtk_spin_button_set_value (window->spin_relaxation,
04104                                      (gdouble) input->relaxation);
04105           switch (input->gradient_method)
04106             {
04107             case GRADIENT_METHOD_RANDOM:
04108               gtk_spin_button_set_value (window->spin_estimates,
04109                                          (gdouble) input->nestimates);
04110             }
04111         }
04112       break;
04113     default:
04114       gtk_spin_button_set_value (window->spin_population,
04115                                  (gdouble) input->nsimulations);
04116       gtk_spin_button_set_value (window->spin_generations,
04117                                  (gdouble) input->niterations);
04118       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04119       gtk_spin_button_set_value (window->spin_reproduction,
04120                                  input->reproduction_ratio);
04121       gtk_spin_button_set_value (window->spin_adaptation,
04122                                  input->adaptation_ratio);
04123     }
04124   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04125   g_signal_handler_block (window->button_experiment,
04126                           window->id_experiment_name);
04127   gtk_combo_box_text_remove_all (window->combo_experiment);
04128   for (i = 0; i < input->nexperiments; ++i)
04129     gtk_combo_box_text_append_text (window->combo_experiment,
04130                                     input->experiment[i]);
04131   g_signal_handler_unblock
04132     (window->button_experiment, window->
      id_experiment_name);
04133   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04134   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04135   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04136   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04137   gtk_combo_box_text_remove_all (window->combo_variable);
04138   for (i = 0; i < input->nvariables; ++i)
04139     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
04140   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04141   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04142   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04143   window_set_variable ();
04144   window_update ();
```

```
04145
04146 #if DEBUG
04147   fprintf (stderr, "window_read: end\n");
04148 #endif
04149   return 1;
04150 }
```

Here is the call graph for this function:

5.3.2.6 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 3011 of file mpcotool.c.

```
03012 {
03013   GtkFileChooserDialog *dlg;
03014   GtkFileFilter *filter;
03015   char *buffer;
03016
03017 #if DEBUG
03018   fprintf (stderr, "window_save: start\n");
03019 #endif
03020
03021   // Opening the saving dialog
03022   dlg = (GtkFileChooserDialog *)
03023     gtk_file_chooser_dialog_new (gettext ("Save file"),
03024                                  window->window,
03025                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03026                                  gettext ("_Cancel"),
03027                                  GTK_RESPONSE_CANCEL,
03028                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03029   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03030   buffer = g_build_filename (input->directory, input->name, NULL);
03031   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03032   g_free (buffer);
03033
03034   // Adding XML filter
03035   filter = (GtkFileFilter *) gtk_file_filter_new ();
03036   gtk_file_filter_set_name (filter, "XML");
03037   gtk_file_filter_add_pattern (filter, "*.xml");
03038   gtk_file_filter_add_pattern (filter, "*.XML");
03039   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03040
03041   // If OK response then saving
03042   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03043     {
03044
03045       // Adding properties to the root XML node
03046       input->simulator = gtk_file_chooser_get_filename
03047         (GTK_FILE_CHOOSER (window->button_simulator));
03048       if (gtk_toggle_button_get_active
03049           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03050         input->evaluator = gtk_file_chooser_get_filename
03051           (GTK_FILE_CHOOSER (window->button_evaluator));
03052       else
03053         input->evaluator = NULL;
03054       input->result
03055         = (char *) xmlStrdup ((const xmlChar *)
03056                               gtk_entry_get_text (window->entry_result));
03057       input->variables
03058         = (char *) xmlStrdup ((const xmlChar *)
03059                               gtk_entry_get_text (window->entry_variables));
03060
03061       // Setting the algorithm
03062       switch (window_get_algorithm ())
03063         {
03064         case ALGORITHM_MONTE_CARLO:
03065           input->algorithm = ALGORITHM_MONTE_CARLO;
03066           input->nsimulations
03067             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03068           input->niterations
03069             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03070           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
```

```
03071           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03072           window_save_gradient ();
03073           break;
03074         case ALGORITHM_SWEEP:
03075           input->algorithm = ALGORITHM_SWEEP;
03076           input->niterations
03077             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03078           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03079           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03080           window_save_gradient ();
03081           break;
03082         default:
03083           input->algorithm = ALGORITHM_GENETIC;
03084           input->nsimulations
03085             = gtk_spin_button_get_value_as_int (window->spin_population);
03086           input->niterations
03087             = gtk_spin_button_get_value_as_int (window->spin_generations);
03088           input->mutation_ratio
03089             = gtk_spin_button_get_value (window->spin_mutation);
03090           input->reproduction_ratio
03091             = gtk_spin_button_get_value (window->spin_reproduction);
03092           input->adaptation_ratio
03093             = gtk_spin_button_get_value (window->spin_adaptation);
03094           break;
03095         }
03096
03097       // Saving the XML file
03098       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03099       input_save (buffer);
03100
03101       // Closing and freeing memory
03102       g_free (buffer);
03103       gtk_widget_destroy (GTK_WIDGET (dlg));
03104 #if DEBUG
03105       fprintf (stderr, "window_save: end\n");
03106 #endif
03107       return 1;
03108     }
03109
03110   // Closing and freeing memory
03111   gtk_widget_destroy (GTK_WIDGET (dlg));
03112 #if DEBUG
03113   fprintf (stderr, "window_save: end\n");
03114 #endif
03115   return 0;
03116 }
```

Here is the call graph for this function:

5.3.2.7 void window_template_experiment (void * *data*)

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

Parameters

data	Callback data (i-th input template).

Definition at line 3657 of file mpcotool.c.

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

```
03677 #endif
03678 }
```

5.4 interface.h

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier 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
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 1444 of file mpcotool.c.

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

```
01475 #if DEBUG
01476   fprintf (stderr, "calibrate_best: end\n");
01477 #endif
01478 }
```

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 1757 of file mpcotool.c.

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

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

Function to estimate a component of the gradient vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1894 of file mpcotool.c.

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

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

Function to estimate a component of the gradient vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 1867 of file mpcotool.c.

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

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 2060 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.6 void calibrate_gradient_sequential (unsigned int *simulation*)

Function to estimate the gradient sequentially.

Parameters

simulation	Simulation number.

Definition at line 1787 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.7 void * calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1822 of file mpcotool.c.

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

```
01850 #if DEBUG
01851   fprintf (stderr, "calibrate_gradient_thread: end\n");
01852 #endif
01853   g_thread_exit (NULL);
01854   return NULL;
01855 }
```

Here is the call graph for this function:

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

Function to write the simulation input file.

Parameters

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

Definition at line 1197 of file mpcotool.c.

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

```
01256                                           0, NULL);
01257       g_free (buffer2);
01258       g_regex_unref (regex);
01259     }
01260
01261   // Saving input file
01262   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01263   g_free (buffer3);
01264   fclose (file);
01265
01266 calibrate_input_end:
01267 #if DEBUG
01268   fprintf (stderr, "calibrate_input: end\n");
01269 #endif
01270   return;
01271 }
```

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 1562 of file mpcotool.c.

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

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

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1284 of file mpcotool.c.

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

```
01351     {
01352       strcpy (result, "");
01353       file_result = g_fopen (output, "r");
01354       e = atof (fgets (buffer, 512, file_result));
01355       fclose (file_result);
01356     }
01357
01358   // Removing files
01359 #if !DEBUG
01360   for (i = 0; i < calibrate->ninputs; ++i)
01361     {
01362       if (calibrate->file[i][0])
01363         {
01364           snprintf (buffer, 512, RM " %s", &input[i][0]);
01365           system (buffer);
01366         }
01367     }
01368   snprintf (buffer, 512, RM " %s %s", output, result);
01369   system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373   fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376   // Returning the objective function
01377   return e * calibrate->weight[experiment];
01378 }
```

Here is the call graph for this function:

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

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

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1416 of file mpcotool.c.

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

5.5.2.12 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

simulation	Simulation number.

Definition at line 1924 of file mpcotool.c.

```
01925 {
01926   GThread *thread[nthreads_gradient];
01927   ParallelData data[nthreads_gradient];
```

```
01928   unsigned int i, j, k, b;
01929 #if DEBUG
01930   fprintf (stderr, "calibrate_step_gradient: start\n");
01931 #endif
01932   for (i = 0; i < calibrate->nestimates; ++i)
01933     {
01934       k = (simulation + i) * calibrate->nvariables;
01935       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01936 #if DEBUG
01937       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01938                simulation + i, calibrate->simulation_best[0]);
01939 #endif
01940       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01941         {
01942 #if DEBUG
01943           fprintf (stderr,
01944                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01945                    i, j, calibrate->value[b]);
01946 #endif
01947           calibrate->value[k]
01948             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01949           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01950                                             calibrate->rangeminabs[j]),
01951                                       calibrate->rangemaxabs[j]);
01952 #if DEBUG
01953           fprintf (stderr,
01954                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01955                    i, j, calibrate->value[k]);
01956 #endif
01957         }
01958     }
01959   if (nthreads_gradient == 1)
01960     calibrate_gradient_sequential (simulation);
01961   else
01962     {
01963       for (i = 0; i <= nthreads_gradient; ++i)
01964         {
01965           calibrate->thread_gradient[i]
01966             = simulation + calibrate->nstart_gradient
01967             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01968               / nthreads_gradient;
01969 #if DEBUG
01970           fprintf (stderr,
01971                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01972                    i, calibrate->thread_gradient[i]);
01973 #endif
01974         }
01975       for (i = 0; i < nthreads_gradient; ++i)
01976         {
01977           data[i].thread = i;
01978           thread[i] = g_thread_new
01979             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01980         }
01981       for (i = 0; i < nthreads_gradient; ++i)
01982         g_thread_join (thread[i]);
01983     }
01984 #if DEBUG
01985   fprintf (stderr, "calibrate_step_gradient: end\n");
01986 #endif
01987 }
```

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 1518 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.14 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4876 of file mpcotool.c.

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

5.5.2.15 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 549 of file mpcotool.c.

```
00550 {
00551   char buffer2[64];
00552   char *buffert[MAX_NINPUTS] =
```

```
00553     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00554   xmlDoc *doc;
00555   xmlNode *node, *child;
00556   xmlChar *buffer;
00557   char *msg;
00558   int error_code;
00559   unsigned int i;
00560
00561 #if DEBUG
00562   fprintf (stderr, "input_open: start\n");
00563 #endif
00564
00565   // Resetting input data
00566   buffer = NULL;
00567   input_new ();
00568
00569   // Parsing the input file
00570 #if DEBUG
00571   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00572 #endif
00573   doc = xmlParseFile (filename);
00574   if (!doc)
00575     {
00576       msg = gettext ("Unable to parse the input file");
00577       goto exit_on_error;
00578     }
00579
00580   // Getting the root node
00581 #if DEBUG
00582   fprintf (stderr, "input_open: getting the root node\n");
00583 #endif
00584   node = xmlDocGetRootElement (doc);
00585   if (xmlStrcmp (node->name, XML_CALIBRATE))
00586     {
00587       msg = gettext ("Bad root XML node");
00588       goto exit_on_error;
00589     }
00590
00591   // Getting results file names
00592   input->result = (char *) xmlGetProp (node, XML_RESULT);
00593   if (!input->result)
00594     input->result = (char *) xmlStrdup (result_name);
00595   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00596   if (!input->variables)
00597     input->variables = (char *) xmlStrdup (variables_name);
00598
00599   // Opening simulator program name
00600   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00601   if (!input->simulator)
00602     {
00603       msg = gettext ("Bad simulator program");
00604       goto exit_on_error;
00605     }
00606
00607   // Opening evaluator program name
00608   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00609
00610   // Obtaining pseudo-random numbers generator seed
00611   input->seed
00612     = xml_node_get_uint_with_default (node,
    XML_SEED, DEFAULT_RANDOM_SEED,
00613                                       &error_code);
00614   if (error_code)
00615     {
00616       msg = gettext ("Bad pseudo-random numbers generator seed");
00617       goto exit_on_error;
00618     }
00619
00620   // Opening algorithm
00621   buffer = xmlGetProp (node, XML_ALGORITHM);
00622   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00623     {
00624       input->algorithm = ALGORITHM_MONTE_CARLO;
00625
00626       // Obtaining simulations number
00627       input->nsimulations
00628         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00629       if (error_code)
00630         {
00631           msg = gettext ("Bad simulations number");
00632           goto exit_on_error;
00633         }
00634     }
00635   else if (!xmlStrcmp (buffer, XML_SWEEP))
00636     input->algorithm = ALGORITHM_SWEEP;
00637   else if (!xmlStrcmp (buffer, XML_GENETIC))
00638     {
```

```
00639       input->algorithm = ALGORITHM_GENETIC;
00640
00641       // Obtaining population
00642       if (xmlHasProp (node, XML_NPOPULATION))
00643         {
00644           input->nsimulations
00645             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00646           if (error_code || input->nsimulations < 3)
00647             {
00648               msg = gettext ("Invalid population number");
00649               goto exit_on_error;
00650             }
00651         }
00652       else
00653         {
00654           msg = gettext ("No population number");
00655           goto exit_on_error;
00656         }
00657
00658       // Obtaining generations
00659       if (xmlHasProp (node, XML_NGENERATIONS))
00660         {
00661           input->niterations
00662             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00663           if (error_code || !input->niterations)
00664             {
00665               msg = gettext ("Invalid generations number");
00666               goto exit_on_error;
00667             }
00668         }
00669       else
00670         {
00671           msg = gettext ("No generations number");
00672           goto exit_on_error;
00673         }
00674
00675       // Obtaining mutation probability
00676       if (xmlHasProp (node, XML_MUTATION))
00677         {
00678           input->mutation_ratio
00679             = xml_node_get_float (node, XML_MUTATION, &error_code);
00680           if (error_code || input->mutation_ratio < 0.
00681               || input->mutation_ratio >= 1.)
00682             {
00683               msg = gettext ("Invalid mutation probability");
00684               goto exit_on_error;
00685             }
00686         }
00687       else
00688         {
00689           msg = gettext ("No mutation probability");
00690           goto exit_on_error;
00691         }
00692
00693       // Obtaining reproduction probability
00694       if (xmlHasProp (node, XML_REPRODUCTION))
00695         {
00696           input->reproduction_ratio
00697             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00698           if (error_code || input->reproduction_ratio < 0.
00699               || input->reproduction_ratio >= 1.0)
00700             {
00701               msg = gettext ("Invalid reproduction probability");
00702               goto exit_on_error;
00703             }
00704         }
00705       else
00706         {
00707           msg = gettext ("No reproduction probability");
00708           goto exit_on_error;
00709         }
00710
00711       // Obtaining adaptation probability
00712       if (xmlHasProp (node, XML_ADAPTATION))
00713         {
00714           input->adaptation_ratio
00715             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00716           if (error_code || input->adaptation_ratio < 0.
00717               || input->adaptation_ratio >= 1.)
00718             {
00719               msg = gettext ("Invalid adaptation probability");
00720               goto exit_on_error;
00721             }
00722         }
00723       else
00724         {
00725           msg = gettext ("No adaptation probability");
```

```
00726           goto exit_on_error;
00727         }
00728
00729       // Checking survivals
00730       i = input->mutation_ratio * input->nsimulations;
00731       i += input->reproduction_ratio * input->
    nsimulations;
00732       i += input->adaptation_ratio * input->
    nsimulations;
00733       if (i > input->nsimulations - 2)
00734         {
00735           msg = gettext
00736             ("No enough survival entities to reproduce the population");
00737           goto exit_on_error;
00738         }
00739     }
00740   else
00741     {
00742       msg = gettext ("Unknown algorithm");
00743       goto exit_on_error;
00744     }
00745   xmlFree (buffer);
00746   buffer = NULL;
00747
00748   if (input->algorithm == ALGORITHM_MONTE_CARLO
00749       || input->algorithm == ALGORITHM_SWEEP)
00750     {
00751
00752       // Obtaining iterations number
00753       input->niterations
00754         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00755       if (error_code == 1)
00756         input->niterations = 1;
00757       else if (error_code)
00758         {
00759           msg = gettext ("Bad iterations number");
00760           goto exit_on_error;
00761         }
00762
00763       // Obtaining best number
00764       input->nbest
00765         = xml_node_get_uint_with_default (node,
    XML_NBEST, 1, &error_code);
00766       if (error_code || !input->nbest)
00767         {
00768           msg = gettext ("Invalid best number");
00769           goto exit_on_error;
00770         }
00771
00772       // Obtaining tolerance
00773       input->tolerance
00774         = xml_node_get_float_with_default (node,
    XML_TOLERANCE, 0.,
00775                                            &error_code);
00776       if (error_code || input->tolerance < 0.)
00777         {
00778           msg = gettext ("Invalid tolerance");
00779           goto exit_on_error;
00780         }
00781
00782       // Getting gradient method parameters
00783       if (xmlHasProp (node, XML_NSTEPS))
00784         {
00785           input->nsteps = xml_node_get_uint (node,
    XML_NSTEPS, &error_code);
00786           if (error_code || !input->nsteps)
00787             {
00788               msg = gettext ("Invalid steps number");
00789               goto exit_on_error;
00790             }
00791           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00792           if (!xmlStrcmp (buffer, XML_COORDINATES))
00793             input->gradient_method =
    GRADIENT_METHOD_COORDINATES;
00794           else if (!xmlStrcmp (buffer, XML_RANDOM))
00795             {
00796               input->gradient_method =
    GRADIENT_METHOD_RANDOM;
00797               input->nestimates
00798                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00799               if (error_code || !input->nestimates)
00800                 {
00801                   msg = gettext ("Invalid estimates number");
00802                   goto exit_on_error;
00803                 }
00804             }
00805           else
```

```
00806             {
00807               msg = gettext ("Unknown method to estimate the gradient");
00808               goto exit_on_error;
00809             }
00810           xmlFree (buffer);
00811           buffer = NULL;
00812           input->relaxation
00813             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00814                                                DEFAULT_RELAXATION, &error_code);
00815           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00816             {
00817               msg = gettext ("Invalid relaxation parameter");
00818               goto exit_on_error;
00819             }
00820         }
00821       else
00822         input->nsteps = 0;
00823     }
00824
00825   // Reading the experimental data
00826   for (child = node->children; child; child = child->next)
00827     {
00828       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833       if (xmlHasProp (child, XML_NAME))
00834         buffer = xmlGetProp (child, XML_NAME);
00835       else
00836         {
00837           snprintf (buffer2, 64, "%s %u: %s",
00838                     gettext ("Experiment"),
00839                     input->nexperiments + 1, gettext ("no data file name"));
00840           msg = buffer2;
00841           goto exit_on_error;
00842         }
00843 #if DEBUG
00844       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845 #endif
00846       input->weight = g_realloc (input->weight,
00847                                  (1 + input->nexperiments) * sizeof (double));
00848       input->weight[input->nexperiments]
00849         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00850       if (error_code)
00851         {
00852           snprintf (buffer2, 64, "%s %s: %s",
00853                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00854           msg = buffer2;
00855           goto exit_on_error;
00856         }
00857 #if DEBUG
00858       fprintf (stderr, "input_open: weight=%lg\n",
00859                input->weight[input->nexperiments]);
00860 #endif
00861       if (!input->nexperiments)
00862         input->ninputs = 0;
00863 #if DEBUG
00864       fprintf (stderr, "input_open: template[0]\n");
00865 #endif
00866       if (xmlHasProp (child, XML_TEMPLATE1))
00867         {
00868           input->template[0]
00869             = (char **) g_realloc (input->template[0],
00870                                    (1 + input->nexperiments) * sizeof (char *));
00871           buffert[0] = (char *) xmlGetProp (child, template[0]);
00872 #if DEBUG
00873           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00874                    input->nexperiments, buffert[0]);
00875 #endif
00876           if (!input->nexperiments)
00877             ++input->ninputs;
00878 #if DEBUG
00879           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00880 #endif
00881         }
00882       else
00883         {
00884           snprintf (buffer2, 64, "%s %s: %s",
00885                     gettext ("Experiment"), buffer, gettext ("no template"));
00886           msg = buffer2;
00887           goto exit_on_error;
00888         }
00889       for (i = 1; i < MAX_NINPUTS; ++i)
```

```
00890         {
00891 #if DEBUG
00892           fprintf (stderr, "input_open: template%u\n", i + 1);
00893 #endif
00894           if (xmlHasProp (child, template[i]))
00895             {
00896               if (input->nexperiments && input->ninputs <= i)
00897                 {
00898                   snprintf (buffer2, 64, "%s %s: %s",
00899                             gettext ("Experiment"),
00900                             buffer, gettext ("bad templates number"));
00901                   msg = buffer2;
00902                   while (i-- > 0)
00903                     xmlFree (buffert[i]);
00904                   goto exit_on_error;
00905                 }
00906               input->template[i] = (char **)
00907                 g_realloc (input->template[i],
00908                            (1 + input->nexperiments) * sizeof (char *));
00909               buffert[i] = (char *) xmlGetProp (child, template[i]);
00910 #if DEBUG
00911               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00912                        input->nexperiments, i + 1,
00913                        input->template[i][input->nexperiments]);
00914 #endif
00915               if (!input->nexperiments)
00916                 ++input->ninputs;
00917 #if DEBUG
00918               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00919 #endif
00920             }
00921           else if (input->nexperiments && input->ninputs >= i)
00922             {
00923               snprintf (buffer2, 64, "%s %s: %s%u",
00924                         gettext ("Experiment"),
00925                         buffer, gettext ("no template"), i + 1);
00926               msg = buffer2;
00927               while (i-- > 0)
00928                 xmlFree (buffert[i]);
00929               goto exit_on_error;
00930             }
00931           else
00932             break;
00933         }
00934       input->experiment
00935         = g_realloc (input->experiment,
00936                      (1 + input->nexperiments) * sizeof (char *));
00937       input->experiment[input->nexperiments] = (char *) buffer;
00938       for (i = 0; i < input->ninputs; ++i)
00939         input->template[i][input->nexperiments] = buffert[i];
00940       ++input->nexperiments;
00941 #if DEBUG
00942       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00943 #endif
00944     }
00945   if (!input->nexperiments)
00946     {
00947       msg = gettext ("No calibration experiments");
00948       goto exit_on_error;
00949     }
00950   buffer = NULL;
00951
00952   // Reading the variables data
00953   for (; child; child = child->next)
00954     {
00955       if (xmlStrcmp (child->name, XML_VARIABLE))
00956         {
00957           snprintf (buffer2, 64, "%s %u: %s",
00958                     gettext ("Variable"),
00959                     input->nvariables + 1, gettext ("bad XML node"));
00960           msg = buffer2;
00961           goto exit_on_error;
00962         }
00963       if (xmlHasProp (child, XML_NAME))
00964         buffer = xmlGetProp (child, XML_NAME);
00965       else
00966         {
00967           snprintf (buffer2, 64, "%s %u: %s",
00968                     gettext ("Variable"),
00969                     input->nvariables + 1, gettext ("no name"));
00970           msg = buffer2;
00971           goto exit_on_error;
00972         }
00973       if (xmlHasProp (child, XML_MINIMUM))
00974         {
00975           input->rangemin = g_realloc
00976             (input->rangemin, (1 + input->nvariables) * sizeof (double));
```

```
00977           input->rangeminabs = g_realloc
00978             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00979           input->rangemin[input->nvariables]
00980             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00981           if (error_code)
00982             {
00983               snprintf (buffer2, 64, "%s %s: %s",
00984                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00985               msg = buffer2;
00986               goto exit_on_error;
00987             }
00988           input->rangeminabs[input->nvariables]
00989             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MINIMUM,
00990                                                -G_MAXDOUBLE, &error_code);
00991           if (error_code)
00992             {
00993               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00994                         gettext ("bad absolute minimum"));
00995               msg = buffer2;
00996               goto exit_on_error;
00997             }
00998           if (input->rangemin[input->nvariables]
00999               < input->rangeminabs[input->nvariables])
01000             {
01001               snprintf (buffer2, 64, "%s %s: %s",
01002                         gettext ("Variable"),
01003                         buffer, gettext ("minimum range not allowed"));
01004               msg = buffer2;
01005               goto exit_on_error;
01006             }
01007         }
01008       else
01009         {
01010           snprintf (buffer2, 64, "%s %s: %s",
01011                     gettext ("Variable"), buffer, gettext ("no minimum range"));
01012           msg = buffer2;
01013           goto exit_on_error;
01014         }
01015       if (xmlHasProp (child, XML_MAXIMUM))
01016         {
01017           input->rangemax = g_realloc
01018             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01019           input->rangemaxabs = g_realloc
01020             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01021           input->rangemax[input->nvariables]
01022             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01023           if (error_code)
01024             {
01025               snprintf (buffer2, 64, "%s %s: %s",
01026                         gettext ("Variable"), buffer, gettext ("bad maximum"));
01027               msg = buffer2;
01028               goto exit_on_error;
01029             }
01030           input->rangemaxabs[input->nvariables]
01031             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MAXIMUM,
01032                                                G_MAXDOUBLE, &error_code);
01033           if (error_code)
01034             {
01035               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01036                         gettext ("bad absolute maximum"));
01037               msg = buffer2;
01038               goto exit_on_error;
01039             }
01040           if (input->rangemax[input->nvariables]
01041               > input->rangemaxabs[input->nvariables])
01042             {
01043               snprintf (buffer2, 64, "%s %s: %s",
01044                         gettext ("Variable"),
01045                         buffer, gettext ("maximum range not allowed"));
01046               msg = buffer2;
01047               goto exit_on_error;
01048             }
01049         }
01050       else
01051         {
01052           snprintf (buffer2, 64, "%s %s: %s",
01053                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01054           msg = buffer2;
01055           goto exit_on_error;
01056         }
01057       if (input->rangemax[input->nvariables]
01058           < input->rangemin[input->nvariables])
01059         {
01060           snprintf (buffer2, 64, "%s %s: %s",
01061                     gettext ("Variable"), buffer, gettext ("bad range"));
```

```
01062           msg = buffer2;
01063           goto exit_on_error;
01064         }
01065       input->precision = g_realloc
01066         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01067       input->precision[input->nvariables]
01068         = xml_node_get_uint_with_default (child,
      XML_PRECISION,
01069                                           DEFAULT_PRECISION, &error_code);
01070       if (error_code || input->precision[input->nvariables] >=
      NPRECISIONS)
01071         {
01072           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01073                     gettext ("bad precision"));
01074           msg = buffer2;
01075           goto exit_on_error;
01076         }
01077       if (input->algorithm == ALGORITHM_SWEEP)
01078         {
01079           if (xmlHasProp (child, XML_NSWEEPS))
01080             {
01081               input->nsweeps = (unsigned int *)
01082                 g_realloc (input->nsweeps,
01083                            (1 + input->nvariables) * sizeof (unsigned int));
01084               input->nsweeps[input->nvariables]
01085                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01086               if (error_code || !input->nsweeps[input->
      nvariables])
01087                 {
01088                   snprintf (buffer2, 64, "%s %s: %s",
01089                             gettext ("Variable"),
01090                             buffer, gettext ("bad sweeps"));
01091                   msg = buffer2;
01092                   goto exit_on_error;
01093                 }
01094             }
01095           else
01096             {
01097               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                         gettext ("no sweeps number"));
01099               msg = buffer2;
01100               goto exit_on_error;
01101             }
01102 #if DEBUG
01103           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01104                    input->nsweeps[input->nvariables],
      input->nsimulations);
01105 #endif
01106         }
01107       if (input->algorithm == ALGORITHM_GENETIC)
01108         {
01109           // Obtaining bits representing each variable
01110           if (xmlHasProp (child, XML_NBITS))
01111             {
01112               input->nbits = (unsigned int *)
01113                 g_realloc (input->nbits,
01114                            (1 + input->nvariables) * sizeof (unsigned int));
01115               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01116               if (error_code || !i)
01117                 {
01118                   snprintf (buffer2, 64, "%s %s: %s",
01119                             gettext ("Variable"),
01120                             buffer, gettext ("invalid bits number"));
01121                   msg = buffer2;
01122                   goto exit_on_error;
01123                 }
01124               input->nbits[input->nvariables] = i;
01125             }
01126           else
01127             {
01128               snprintf (buffer2, 64, "%s %s: %s",
01129                         gettext ("Variable"),
01130                         buffer, gettext ("no bits number"));
01131               msg = buffer2;
01132               goto exit_on_error;
01133             }
01134         }
01135       else if (input->nsteps)
01136         {
01137           input->step = (double *)
01138             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01139           input->step[input->nvariables]
01140             = xml_node_get_float (child, XML_STEP, &error_code);
01141           if (error_code || input->step[input->nvariables] < 0.)
01142             {
01143               snprintf (buffer2, 64, "%s %s: %s",
01144                         gettext ("Variable"),
```

```
01145                         buffer, gettext ("bad step size"));
01146               msg = buffer2;
01147               goto exit_on_error;
01148             }
01149         }
01150       input->label = g_realloc
01151         (input->label, (1 + input->nvariables) * sizeof (char *));
01152       input->label[input->nvariables] = (char *) buffer;
01153       ++input->nvariables;
01154     }
01155   if (!input->nvariables)
01156     {
01157       msg = gettext ("No calibration variables");
01158       goto exit_on_error;
01159     }
01160   buffer = NULL;
01161
01162   // Getting the working directory
01163   input->directory = g_path_get_dirname (filename);
01164   input->name = g_path_get_basename (filename);
01165
01166   // Closing the XML document
01167   xmlFreeDoc (doc);
01168
01169 #if DEBUG
01170   fprintf (stderr, "input_open: end\n");
01171 #endif
01172   return 1;
01173
01174 exit_on_error:
01175   xmlFree (buffer);
01176   xmlFreeDoc (doc);
01177   show_error (msg);
01178   input_free ();
01179 #if DEBUG
01180   fprintf (stderr, "input_open: end\n");
01181 #endif
01182   return 0;
01183 }
```

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 2700 of file mpcotool.c.

```
02701 {
02702   unsigned int i, j;
02703   char *buffer;
02704   xmlDoc *doc;
02705   xmlNode *node, *child;
02706   GFile *file, *file2;
02707
02708 #if DEBUG
02709   fprintf (stderr, "input_save: start\n");
02710 #endif
02711
02712   // Getting the input file directory
02713   input->name = g_path_get_basename (filename);
02714   input->directory = g_path_get_dirname (filename);
02715   file = g_file_new_for_path (input->directory);
02716
02717   // Opening the input file
02718   doc = xmlNewDoc ((const xmlChar *) "1.0");
02719
02720   // Setting root XML node
02721   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02722   xmlDocSetRootElement (doc, node);
02723
02724   // Adding properties to the root XML node
02725   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02726     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02727   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02728     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
```

```
      variables);
02729   file2 = g_file_new_for_path (input->simulator);
02730   buffer = g_file_get_relative_path (file, file2);
02731   g_object_unref (file2);
02732   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02733   g_free (buffer);
02734   if (input->evaluator)
02735     {
02736       file2 = g_file_new_for_path (input->evaluator);
02737       buffer = g_file_get_relative_path (file, file2);
02738       g_object_unref (file2);
02739       if (xmlStrlen ((xmlChar *) buffer))
02740         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02741       g_free (buffer);
02742     }
02743   if (input->seed != DEFAULT_RANDOM_SEED)
02744     xml_node_set_uint (node, XML_SEED, input->seed);
02745
02746   // Setting the algorithm
02747   buffer = (char *) g_malloc (64);
02748   switch (input->algorithm)
02749     {
02750     case ALGORITHM_MONTE_CARLO:
02751       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02752       snprintf (buffer, 64, "%u", input->nsimulations);
02753       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02754       snprintf (buffer, 64, "%u", input->niterations);
02755       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02756       snprintf (buffer, 64, "%.3lg", input->tolerance);
02757       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02758       snprintf (buffer, 64, "%u", input->nbest);
02759       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02760       input_save_gradient (node);
02761       break;
02762     case ALGORITHM_SWEEP:
02763       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02764       snprintf (buffer, 64, "%u", input->niterations);
02765       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02766       snprintf (buffer, 64, "%.3lg", input->tolerance);
02767       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02768       snprintf (buffer, 64, "%u", input->nbest);
02769       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02770       input_save_gradient (node);
02771       break;
02772     default:
02773       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02774       snprintf (buffer, 64, "%u", input->nsimulations);
02775       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02776       snprintf (buffer, 64, "%u", input->niterations);
02777       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02778       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02779       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02780       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02781       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02782       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02783       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02784       break;
02785     }
02786   g_free (buffer);
02787
02788   // Setting the experimental data
02789   for (i = 0; i < input->nexperiments; ++i)
02790     {
02791       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02792       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02793       if (input->weight[i] != 1.)
02794         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02795       for (j = 0; j < input->ninputs; ++j)
02796         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02797     }
02798
02799   // Setting the variables data
02800   for (i = 0; i < input->nvariables; ++i)
02801     {
02802       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02803       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02804       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02805       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02806         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02807       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02808       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02809         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
```

```
02810       if (input->precision[i] != DEFAULT_PRECISION)
02811         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02812       if (input->algorithm == ALGORITHM_SWEEP)
02813         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02814       else if (input->algorithm == ALGORITHM_GENETIC)
02815         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02816       if (input->nsteps)
02817         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02818     }
02819
02820   // Saving the XML file
02821   xmlSaveFormatFile (filename, doc, 1);
02822
02823   // Freeing memory
02824   xmlFreeDoc (doc);
02825
02826 #if DEBUG
02827   fprintf (stderr, "input_save: end\n");
02828 #endif
02829 }
```

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 2668 of file mpcotool.c.

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

Here is the call graph for this function:

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

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

0 on success, >0 on error.

Definition at line 4897 of file mpcotool.c.

```
04898 {
04899 #if HAVE_GTK
04900   char *buffer;
04901 #endif
04902
04903   // Starting pseudo-random numbers generator
04904   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04905   calibrate->seed = DEFAULT_RANDOM_SEED;
04906
04907   // Allowing spaces in the XML data file
04908   xmlKeepBlanksDefault (0);
04909
04910   // Starting MPI
04911 #if HAVE_MPI
04912   MPI_Init (&argn, &argc);
04913   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04914   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04915   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04916 #else
04917   ntasks = 1;
04918 #endif
04919
04920 #if HAVE_GTK
04921
04922   // Getting threads number
04923   nthreads_gradient = nthreads = cores_number ();
04924
04925   // Setting local language and international floating point numbers notation
04926   setlocale (LC_ALL, "");
04927   setlocale (LC_NUMERIC, "C");
04928   window->application_directory = g_get_current_dir ();
04929   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04930   bindtextdomain (PROGRAM_INTERFACE, buffer);
04931   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04932   textdomain (PROGRAM_INTERFACE);
04933
04934   // Initing GTK+
04935   gtk_disable_setlocale ();
04936   gtk_init (&argn, &argc);
04937
04938   // Opening the main window
04939   window_new ();
04940   gtk_main ();
04941
04942   // Freeing memory
04943   input_free ();
04944   g_free (buffer);
04945   gtk_widget_destroy (GTK_WIDGET (window->window));
04946   g_free (window->application_directory);
04947
04948 #else
04949
04950   // Checking syntax
04951   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04952     {
04953       printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04954       return 1;
04955     }
04956
04957   // Getting threads number
04958   if (argn == 2)
04959     nthreads_gradient = nthreads = cores_number ();
04960   else
04961     {
04962       nthreads_gradient = nthreads = atoi (argc[2]);
04963       if (!nthreads)
04964         {
04965           printf ("Bad threads number\n");
04966           return 2;
04967         }
```

```
04968     }
04969   printf ("nthreads=%u\n", nthreads);
04970
04971   // Making calibration
04972   if (input_open (argc[argn - 1]))
04973     calibrate_open ();
04974
04975   // Freeing memory
04976   calibrate_free ();
04977
04978 #endif
04979
04980   // Closing MPI
04981 #if HAVE_MPI
04982   MPI_Finalize ();
04983 #endif
04984
04985   // Freeing memory
04986   gsl_rng_free (calibrate->rng);
04987
04988   // Closing
04989   return 0;
04990 }
```

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 257 of file mpcotool.c.

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 227 of file mpcotool.c.

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

```
00247 #endif
00248 }
```

5.5.2.21 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2933 of file mpcotool.c.

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

5.5.2.22 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2956 of file mpcotool.c.

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

5.5.2.23 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

1 on succes, 0 on error.

Definition at line 4053 of file mpcotool.c.

```
04054 {
04055   unsigned int i;
04056   char *buffer;
04057 #if DEBUG
04058   fprintf (stderr, "window_read: start\n");
04059 #endif
04060
04061   // Reading new input file
04062   input_free ();
04063   if (!input_open (filename))
04064     return 0;
04065
04066   // Setting GTK+ widgets data
04067   gtk_entry_set_text (window->entry_result, input->result);
04068   gtk_entry_set_text (window->entry_variables, input->
      variables);
04069   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04070   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04071                                  (window->button_simulator), buffer);
04072   g_free (buffer);
04073   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04074                                 (size_t) input->evaluator);
04075   if (input->evaluator)
04076     {
04077       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04078       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04079                                      (window->button_evaluator), buffer);
04080       g_free (buffer);
04081     }
04082   gtk_toggle_button_set_active
04083     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04084   switch (input->algorithm)
04085     {
04086     case ALGORITHM_MONTE_CARLO:
04087       gtk_spin_button_set_value (window->spin_simulations,
04088                                  (gdouble) input->nsimulations);
04089     case ALGORITHM_SWEEP:
04090       gtk_spin_button_set_value (window->spin_iterations,
04091                                  (gdouble) input->niterations);
04092       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04093       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04094       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04095                                     input->nsteps);
04096       if (input->nsteps)
04097         {
04098           gtk_toggle_button_set_active
04099             (GTK_TOGGLE_BUTTON (window->button_gradient
04100                                 [input->gradient_method]), TRUE);
04101           gtk_spin_button_set_value (window->spin_steps,
04102                                      (gdouble) input->nsteps);
04103           gtk_spin_button_set_value (window->spin_relaxation,
04104                                      (gdouble) input->relaxation);
04105           switch (input->gradient_method)
04106             {
04107             case GRADIENT_METHOD_RANDOM:
04108               gtk_spin_button_set_value (window->spin_estimates,
04109                                          (gdouble) input->nestimates);
04110             }
04111         }
04112       break;
04113     default:
04114       gtk_spin_button_set_value (window->spin_population,
04115                                  (gdouble) input->nsimulations);
04116       gtk_spin_button_set_value (window->spin_generations,
04117                                  (gdouble) input->niterations);
04118       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04119       gtk_spin_button_set_value (window->spin_reproduction,
04120                                  input->reproduction_ratio);
```

```
04121       gtk_spin_button_set_value (window->spin_adaptation,
04122                                  input->adaptation_ratio);
04123     }
04124   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04125   g_signal_handler_block (window->button_experiment,
04126                           window->id_experiment_name);
04127   gtk_combo_box_text_remove_all (window->combo_experiment);
04128   for (i = 0; i < input->nexperiments; ++i)
04129     gtk_combo_box_text_append_text (window->combo_experiment,
04130                                     input->experiment[i]);
04131   g_signal_handler_unblock
04132     (window->button_experiment, window->
      id_experiment_name);
04133   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04134   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04135   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04136   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04137   gtk_combo_box_text_remove_all (window->combo_variable);
04138   for (i = 0; i < input->nvariables; ++i)
04139     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
04140   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04141   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04142   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04143   window_set_variable ();
04144   window_update ();
04145
04146 #if DEBUG
04147   fprintf (stderr, "window_read: end\n");
04148 #endif
04149   return 1;
04150 }
```

Here is the call graph for this function:

5.5.2.24 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 3011 of file mpcotool.c.

```
03012 {
03013   GtkFileChooserDialog *dlg;
03014   GtkFileFilter *filter;
03015   char *buffer;
03016
03017 #if DEBUG
03018   fprintf (stderr, "window_save: start\n");
03019 #endif
03020
03021   // Opening the saving dialog
03022   dlg = (GtkFileChooserDialog *)
03023     gtk_file_chooser_dialog_new (gettext ("Save file"),
03024                                  window->window,
03025                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03026                                  gettext ("_Cancel"),
03027                                  GTK_RESPONSE_CANCEL,
03028                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03029   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03030   buffer = g_build_filename (input->directory, input->name, NULL);
03031   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03032   g_free (buffer);
03033
03034   // Adding XML filter
03035   filter = (GtkFileFilter *) gtk_file_filter_new ();
03036   gtk_file_filter_set_name (filter, "XML");
03037   gtk_file_filter_add_pattern (filter, "*.xml");
03038   gtk_file_filter_add_pattern (filter, "*.XML");
03039   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
```

```
03040
03041   // If OK response then saving
03042   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03043     {
03044
03045       // Adding properties to the root XML node
03046       input->simulator = gtk_file_chooser_get_filename
03047         (GTK_FILE_CHOOSER (window->button_simulator));
03048       if (gtk_toggle_button_get_active
03049           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03050         input->evaluator = gtk_file_chooser_get_filename
03051           (GTK_FILE_CHOOSER (window->button_evaluator));
03052       else
03053         input->evaluator = NULL;
03054       input->result
03055         = (char *) xmlStrdup ((const xmlChar *)
03056                               gtk_entry_get_text (window->entry_result));
03057       input->variables
03058         = (char *) xmlStrdup ((const xmlChar *)
03059                               gtk_entry_get_text (window->entry_variables));
03060
03061       // Setting the algorithm
03062       switch (window_get_algorithm ())
03063         {
03064         case ALGORITHM_MONTE_CARLO:
03065           input->algorithm = ALGORITHM_MONTE_CARLO;
03066           input->nsimulations
03067             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03068           input->niterations
03069             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03070           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03071           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03072           window_save_gradient ();
03073           break;
03074         case ALGORITHM_SWEEP:
03075           input->algorithm = ALGORITHM_SWEEP;
03076           input->niterations
03077             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03078           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03079           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03080           window_save_gradient ();
03081           break;
03082         default:
03083           input->algorithm = ALGORITHM_GENETIC;
03084           input->nsimulations
03085             = gtk_spin_button_get_value_as_int (window->spin_population);
03086           input->niterations
03087             = gtk_spin_button_get_value_as_int (window->spin_generations);
03088           input->mutation_ratio
03089             = gtk_spin_button_get_value (window->spin_mutation);
03090           input->reproduction_ratio
03091             = gtk_spin_button_get_value (window->spin_reproduction);
03092           input->adaptation_ratio
03093             = gtk_spin_button_get_value (window->spin_adaptation);
03094           break;
03095         }
03096
03097       // Saving the XML file
03098       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03099       input_save (buffer);
03100
03101       // Closing and freeing memory
03102       g_free (buffer);
03103       gtk_widget_destroy (GTK_WIDGET (dlg));
03104 #if DEBUG
03105       fprintf (stderr, "window_save: end\n");
03106 #endif
03107       return 1;
03108     }
03109
03110   // Closing and freeing memory
03111   gtk_widget_destroy (GTK_WIDGET (dlg));
03112 #if DEBUG
03113   fprintf (stderr, "window_save: end\n");
03114 #endif
03115   return 0;
03116 }
```

Here is the call graph for this function:

5.5.2.25 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 3657 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 367 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 401 of file mpcotool.c.

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

Here is the call graph for this function:

5.5.2.28 int xml_node_get_int (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 275 of file mpcotool.c.

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

5.5.2.29 int xml_node_get_uint (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 306 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 340 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Definition at line 464 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 426 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 445 of file mpcotool.c.

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

5.5.3 Variable Documentation

5.5.3.1 const char* format[NPRECISIONS]

Initial value:

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

Array of C-strings with variable formats.

Definition at line 118 of file mpcotool.c.

5.5.3.2 const double precision[NPRECISIONS]

Initial value:

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

Array of variable precisions.

Definition at line 123 of file mpcotool.c.

5.5.3.3 const xmlChar* template[MAX_NINPUTS]

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 111 of file mpcotool.c.

5.6 mpcotool.c

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

```
00051 #elif (!__BSD_VISIBLE)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "mpcotool.h"
00059 #if HAVE_GTK
00060 #include <gio/gio.h>
00061 #include <gtk/gtk.h>
00062 #include "interface.h"
00063 #endif
00064
00065 #define DEBUG 0
00066
00067
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00092 int ntasks;
00093 unsigned int nthreads;
00094 unsigned int nthreads_gradient;
00096 GMutex mutex[1];
00097 void (*calibrate_algorithm) ();
00099 double (*calibrate_estimate_gradient) (unsigned int variable,
00100                                        unsigned int estimate);
00102 Input input[1];
00104 Calibrate calibrate[1];
00105
00106 const xmlChar *result_name = (xmlChar *) "result";
00108 const xmlChar *variables_name = (xmlChar *) "variables";
00110
00111 const xmlChar *template[MAX_NINPUTS] = {
00112   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00113   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00114 };
00115
00117
00118 const char *format[NPRECISIONS] = {
00119   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00120   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00121 };
00122
00123 const double precision[NPRECISIONS] = {
00124   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00125   1e-13, 1e-14
00126 };
00127
00128 const char *logo[] = {
00129   "32 32 3 1",
00130   "      c None",
00131   ".     c #0000FF",
00132   "+     c #FF0000",
00133   "                                ",
00134   "                                ",
00135   "                                ",
00136   "      .       .       .       ",
00137   "      .       .       .       ",
00138   "      .       .       .       ",
00139   "      .       .       .       ",
00140   "      .       .      +++      ",
00141   "      .       .     +++++     ",
00142   "      .       .     +++++     ",
00143   "      .       .     +++++     ",
00144   "     +++      .      +++     +++     ",
00145   "    +++++     .       .     +++++    ",
00146   "    +++++     .       .     +++++    ",
00147   "    +++++     .       .     +++++    ",
00148   "     +++      .       .      +++     ",
00149   "      .       .       .       .      ",
00150   "      .      +++      .       .      ",
00151   "      .     +++++     .       .      ",
```

```
00152   "     .    +++++    .       .     ",
00153   "     .    +++++    .       .     ",
00154   "     .     +++     .       .     ",
00155   "     .       .      .       .     ",
00156   "     .       .      .       .     ",
00157   "     .       .      .       .     ",
00158   "     .       .      .       .     ",
00159   "     .       .      .       .     ",
00160   "     .       .      .       .     ",
00161   "     .       .      .       .     ",
00162   "                                  ",
00163   "                                  ",
00164   "                                  "
00165 };
00166
00167 /*
00168 const char * logo[] = {
00169 "32 32 3 1",
00170 "     c #FFFFFFFFFFFF",
00171 ".    c #00000000FFFF",
00172 "X    c #FFFF00000000",
00173 "                                  ",
00174 "                                  ",
00175 "                                  ",
00176 "     .      .      .      .     ",
00177 "     .      .      .      .     ",
00178 "     .      .      .      .     ",
00179 "     .      .      .      .     ",
00180 "     .      .     XXX     .     ",
00181 "     .      .    XXXXX    .     ",
00182 "     .      .    XXXXX    .     ",
00183 "     .      .    XXXXX    .     ",
00184 "    XXX     .     XXX    XXX    ",
00185 "   XXXXX    .      .    XXXXX   ",
00186 "   XXXXX    .      .    XXXXX   ",
00187 "   XXXXX    .      .    XXXXX   ",
00188 "    XXX     .      .     XXX    ",
00189 "     .      .      .      .     ",
00190 "     .     XXX     .      .     ",
00191 "     .    XXXXX    .      .     ",
00192 "     .    XXXXX    .      .     ",
00193 "     .    XXXXX    .      .     ",
00194 "     .     XXX     .      .     ",
00195 "     .      .      .      .     ",
00196 "     .      .      .      .     ",
00197 "     .      .      .      .     ",
00198 "     .      .      .      .     ",
00199 "     .      .      .      .     ",
00200 "     .      .      .      .     ",
00201 "     .      .      .      .     ",
00202 "                                  ",
00203 "                                  ",
00204 "                                  "};
00205 */
00206
00207 #if HAVE_GTK
00208 Options options[1];
00210 Running running[1];
00212 Window window[1];
00214 #endif
00215
00226 void
00227 show_message (char *title, char *msg, int type)
00228 {
00229 #if HAVE_GTK
00230   GtkMessageDialog *dlg;
00231
00232   // Creating the dialog
00233   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00234     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00235
00236   // Setting the dialog title
00237   gtk_window_set_title (GTK_WINDOW (dlg), title);
00238
00239   // Showing the dialog and waiting response
00240   gtk_dialog_run (GTK_DIALOG (dlg));
00241
00242   // Closing and freeing memory
00243   gtk_widget_destroy (GTK_WIDGET (dlg));
00244
00245 #else
00246   printf ("%s: %s\n", title, msg);
00247 #endif
00248 }
00249
00256 void
00257 show_error (char *msg)
```

```
00258 {
00259   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00260 }
00261
00274 int
00275 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00276 {
00277   int i = 0;
00278   xmlChar *buffer;
00279   buffer = xmlGetProp (node, prop);
00280   if (!buffer)
00281     *error_code = 1;
00282   else
00283     {
00284       if (sscanf ((char *) buffer, "%d", &i) != 1)
00285         *error_code = 2;
00286       else
00287         *error_code = 0;
00288       xmlFree (buffer);
00289     }
00290   return i;
00291 }
00292
00305 unsigned int
00306 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00307 {
00308   unsigned int i = 0;
00309   xmlChar *buffer;
00310   buffer = xmlGetProp (node, prop);
00311   if (!buffer)
00312     *error_code = 1;
00313   else
00314     {
00315       if (sscanf ((char *) buffer, "%u", &i) != 1)
00316         *error_code = 2;
00317       else
00318         *error_code = 0;
00319       xmlFree (buffer);
00320     }
00321   return i;
00322 }
00323
00339 unsigned int
00340 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00341                                 unsigned int default_value, int *error_code)
00342 {
00343   unsigned int i;
00344   if (xmlHasProp (node, prop))
00345     i = xml_node_get_uint (node, prop, error_code);
00346   else
00347     {
00348       i = default_value;
00349       *error_code = 0;
00350     }
00351   return i;
00352 }
00353
00366 double
00367 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00368 {
00369   double x = 0.;
00370   xmlChar *buffer;
00371   buffer = xmlGetProp (node, prop);
00372   if (!buffer)
00373     *error_code = 1;
00374   else
00375     {
00376       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00377         *error_code = 2;
00378       else
00379         *error_code = 0;
00380       xmlFree (buffer);
00381     }
00382   return x;
00383 }
00384
00400 double
00401 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00402                                  double default_value, int *error_code)
00403 {
00404   double x;
00405   if (xmlHasProp (node, prop))
00406     x = xml_node_get_float (node, prop, error_code);
00407   else
00408     {
00409       x = default_value;
00410       *error_code = 0;
```

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

```
00535   input->nexperiments = input->ninputs = input->nvariables = input->
      nsteps = 0;
00536 #if DEBUG
00537   fprintf (stderr, "input_free: end\n");
00538 #endif
00539 }
00540
00548 int
00549 input_open (char *filename)
00550 {
00551   char buffer2[64];
00552   char *buffert[MAX_NINPUTS] =
00553     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00554   xmlDoc *doc;
00555   xmlNode *node, *child;
00556   xmlChar *buffer;
00557   char *msg;
00558   int error_code;
00559   unsigned int i;
00560
00561 #if DEBUG
00562   fprintf (stderr, "input_open: start\n");
00563 #endif
00564
00565   // Resetting input data
00566   buffer = NULL;
00567   input_new ();
00568
00569   // Parsing the input file
00570 #if DEBUG
00571   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00572 #endif
00573   doc = xmlParseFile (filename);
00574   if (!doc)
00575     {
00576       msg = gettext ("Unable to parse the input file");
00577       goto exit_on_error;
00578     }
00579
00580   // Getting the root node
00581 #if DEBUG
00582   fprintf (stderr, "input_open: getting the root node\n");
00583 #endif
00584   node = xmlDocGetRootElement (doc);
00585   if (xmlStrcmp (node->name, XML_CALIBRATE))
00586     {
00587       msg = gettext ("Bad root XML node");
00588       goto exit_on_error;
00589     }
00590
00591   // Getting results file names
00592   input->result = (char *) xmlGetProp (node, XML_RESULT);
00593   if (!input->result)
00594     input->result = (char *) xmlStrdup (result_name);
00595   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00596   if (!input->variables)
00597     input->variables = (char *) xmlStrdup (variables_name);
00598
00599   // Opening simulator program name
00600   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00601   if (!input->simulator)
00602     {
00603       msg = gettext ("Bad simulator program");
00604       goto exit_on_error;
00605     }
00606
00607   // Opening evaluator program name
00608   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00609
00610   // Obtaining pseudo-random numbers generator seed
00611   input->seed
00612     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00613                                       &error_code);
00614   if (error_code)
00615     {
00616       msg = gettext ("Bad pseudo-random numbers generator seed");
00617       goto exit_on_error;
00618     }
00619
00620   // Opening algorithm
00621   buffer = xmlGetProp (node, XML_ALGORITHM);
00622   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00623     {
00624       input->algorithm = ALGORITHM_MONTE_CARLO;
00625
00626       // Obtaining simulations number
```

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

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

```
00798                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00799               if (error_code || !input->nestimates)
00800                 {
00801                   msg = gettext ("Invalid estimates number");
00802                   goto exit_on_error;
00803                 }
00804             }
00805           else
00806             {
00807               msg = gettext ("Unknown method to estimate the gradient");
00808               goto exit_on_error;
00809             }
00810           xmlFree (buffer);
00811           buffer = NULL;
00812           input->relaxation
00813             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00814                                                DEFAULT_RELAXATION, &error_code);
00815           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00816             {
00817               msg = gettext ("Invalid relaxation parameter");
00818               goto exit_on_error;
00819             }
00820         }
00821       else
00822         input->nsteps = 0;
00823     }
00824
00825   // Reading the experimental data
00826   for (child = node->children; child; child = child->next)
00827     {
00828       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833       if (xmlHasProp (child, XML_NAME))
00834         buffer = xmlGetProp (child, XML_NAME);
00835       else
00836         {
00837           snprintf (buffer2, 64, "%s %u: %s",
00838                     gettext ("Experiment"),
00839                     input->nexperiments + 1, gettext ("no data file name"));
00840           msg = buffer2;
00841           goto exit_on_error;
00842         }
00843 #if DEBUG
00844       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845 #endif
00846       input->weight = g_realloc (input->weight,
00847                                  (1 + input->nexperiments) * sizeof (double));
00848       input->weight[input->nexperiments]
00849         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00850       if (error_code)
00851         {
00852           snprintf (buffer2, 64, "%s %s: %s",
00853                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00854           msg = buffer2;
00855           goto exit_on_error;
00856         }
00857 #if DEBUG
00858       fprintf (stderr, "input_open: weight=%lg\n",
00859                input->weight[input->nexperiments]);
00860 #endif
00861       if (!input->nexperiments)
00862         input->ninputs = 0;
00863 #if DEBUG
00864       fprintf (stderr, "input_open: template[0]\n");
00865 #endif
00866       if (xmlHasProp (child, XML_TEMPLATE1))
00867         {
00868           input->template[0]
00869             = (char **) g_realloc (input->template[0],
00870                                    (1 + input->nexperiments) * sizeof (char *));
00871           buffert[0] = (char *) xmlGetProp (child, template[0]);
00872 #if DEBUG
00873           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00874                    input->nexperiments, buffert[0]);
00875 #endif
00876           if (!input->nexperiments)
00877             ++input->ninputs;
00878 #if DEBUG
00879           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00880 #endif
00881         }
```

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

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

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

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

```
01236 #endif
01237         }
01238       else
01239         {
01240           length = strlen (buffer3);
01241           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01242                                              calibrate->label[i], 0, NULL);
01243           g_free (buffer3);
01244         }
01245       g_regex_unref (regex);
01246       length = strlen (buffer2);
01247       snprintf (buffer, 32, "@value%u@", i + 1);
01248       regex = g_regex_new (buffer, 0, 0, NULL);
01249       snprintf (value, 32, format[calibrate->precision[i]],
01250                 calibrate->value[simulation * calibrate->nvariables + i]);
01251
01252 #if DEBUG
01253       fprintf (stderr, "calibrate_input: value=%s\n", value);
01254 #endif
01255       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01256                                          0, NULL);
01257       g_free (buffer2);
01258       g_regex_unref (regex);
01259     }
01260
01261   // Saving input file
01262   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01263   g_free (buffer3);
01264   fclose (file);
01265
01266 calibrate_input_end:
01267 #if DEBUG
01268   fprintf (stderr, "calibrate_input: end\n");
01269 #endif
01270   return;
01271 }
01272
01283 double
01284 calibrate_parse (unsigned int simulation, unsigned int experiment)
01285 {
01286   unsigned int i;
01287   double e;
01288   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01289     *buffer3, *buffer4;
01290   FILE *file_result;
01291
01292 #if DEBUG
01293   fprintf (stderr, "calibrate_parse: start\n");
01294   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01295            experiment);
01296 #endif
01297
01298   // Opening input files
01299   for (i = 0; i < calibrate->ninputs; ++i)
01300     {
01301       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01302 #if DEBUG
01303       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01304 #endif
01305       calibrate_input (simulation, &input[i][0],
01306                        calibrate->file[i][experiment]);
01307     }
01308   for (; i < MAX_NINPUTS; ++i)
01309     strcpy (&input[i][0], "");
01310 #if DEBUG
01311   fprintf (stderr, "calibrate_parse: parsing end\n");
01312 #endif
01313
01314   // Performing the simulation
01315   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01316   buffer2 = g_path_get_dirname (calibrate->simulator);
01317   buffer3 = g_path_get_basename (calibrate->simulator);
01318   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01319   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01320             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01321             input[6], input[7], output);
01322   g_free (buffer4);
01323   g_free (buffer3);
01324   g_free (buffer2);
01325 #if DEBUG
01326   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01327 #endif
01328   system (buffer);
01329
01330   // Checking the objective value function
01331   if (calibrate->evaluator)
01332     {
```

```
01333       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01334       buffer2 = g_path_get_dirname (calibrate->evaluator);
01335       buffer3 = g_path_get_basename (calibrate->evaluator);
01336       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01337       snprintf (buffer, 512, "\"%s\" %s %s %s",
01338                 buffer4, output, calibrate->experiment[experiment], result);
01339       g_free (buffer4);
01340       g_free (buffer3);
01341       g_free (buffer2);
01342 #if DEBUG
01343       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01344 #endif
01345       system (buffer);
01346       file_result = g_fopen (result, "r");
01347       e = atof (fgets (buffer, 512, file_result));
01348       fclose (file_result);
01349     }
01350   else
01351     {
01352       strcpy (result, "");
01353       file_result = g_fopen (output, "r");
01354       e = atof (fgets (buffer, 512, file_result));
01355       fclose (file_result);
01356     }
01357
01358   // Removing files
01359 #if !DEBUG
01360   for (i = 0; i < calibrate->ninputs; ++i)
01361     {
01362       if (calibrate->file[i][0])
01363         {
01364           snprintf (buffer, 512, RM " %s", &input[i][0]);
01365           system (buffer);
01366         }
01367     }
01368   snprintf (buffer, 512, RM " %s %s", output, result);
01369   system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373   fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376   // Returning the objective function
01377   return e * calibrate->weight[experiment];
01378 }
01379
01384 void
01385 calibrate_print ()
01386 {
01387   unsigned int i;
01388   char buffer[512];
01389 #if HAVE_MPI
01390   if (calibrate->mpi_rank)
01391     return;
01392 #endif
01393   printf ("%s\n", gettext ("Best result"));
01394   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01395   printf ("error = %.15le\n", calibrate->error_old[0]);
01396   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
     error_old[0]);
01397   for (i = 0; i < calibrate->nvariables; ++i)
01398     {
01399       snprintf (buffer, 512, "%s = %s\n",
01400                 calibrate->label[i], format[calibrate->precision[i]]);
01401       printf (buffer, calibrate->value_old[i]);
01402       fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01403     }
01404   fflush (calibrate->file_result);
01405 }
01406
01415 void
01416 calibrate_save_variables (unsigned int simulation, double error)
01417 {
01418   unsigned int i;
01419   char buffer[64];
01420 #if DEBUG
01421   fprintf (stderr, "calibrate_save_variables: start\n");
01422 #endif
01423   for (i = 0; i < calibrate->nvariables; ++i)
01424     {
01425       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01426       fprintf (calibrate->file_variables, buffer,
01427                calibrate->value[simulation * calibrate->nvariables + i]);
01428     }
01429   fprintf (calibrate->file_variables, "%.14le\n", error);
01430 #if DEBUG
```

```
01431   fprintf (stderr, "calibrate_save_variables: end\n");
01432 #endif
01433 }
01434
01443 void
01444 calibrate_best (unsigned int simulation, double value)
01445 {
01446   unsigned int i, j;
01447   double e;
01448 #if DEBUG
01449   fprintf (stderr, "calibrate_best: start\n");
01450   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01451            calibrate->nsaveds, calibrate->nbest);
01452 #endif
01453   if (calibrate->nsaveds < calibrate->nbest
01454       || value < calibrate->error_best[calibrate->nsaveds - 1])
01455     {
01456       if (calibrate->nsaveds < calibrate->nbest)
01457         ++calibrate->nsaveds;
01458       calibrate->error_best[calibrate->nsaveds - 1] = value;
01459       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01460       for (i = calibrate->nsaveds; --i;)
01461         {
01462           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01463             {
01464               j = calibrate->simulation_best[i];
01465               e = calibrate->error_best[i];
01466               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01467               calibrate->error_best[i] = calibrate->error_best[i - 1];
01468               calibrate->simulation_best[i - 1] = j;
01469               calibrate->error_best[i - 1] = e;
01470             }
01471           else
01472             break;
01473         }
01474     }
01475 #if DEBUG
01476   fprintf (stderr, "calibrate_best: end\n");
01477 #endif
01478 }
01479
01484 void
01485 calibrate_sequential ()
01486 {
01487   unsigned int i, j;
01488   double e;
01489 #if DEBUG
01490   fprintf (stderr, "calibrate_sequential: start\n");
01491   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01492            calibrate->nstart, calibrate->nend);
01493 #endif
01494   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01495     {
01496       e = 0.;
01497       for (j = 0; j < calibrate->nexperiments; ++j)
01498         e += calibrate_parse (i, j);
01499       calibrate_best (i, e);
01500       calibrate_save_variables (i, e);
01501 #if DEBUG
01502       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01503 #endif
01504     }
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_sequential: end\n");
01507 #endif
01508 }
01509
01517 void *
01518 calibrate_thread (ParallelData * data)
01519 {
01520   unsigned int i, j, thread;
01521   double e;
01522 #if DEBUG
01523   fprintf (stderr, "calibrate_thread: start\n");
01524 #endif
01525   thread = data->thread;
01526 #if DEBUG
01527   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01528            calibrate->thread[thread], calibrate->thread[thread + 1]);
01529 #endif
01530   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01531     {
01532       e = 0.;
01533       for (j = 0; j < calibrate->nexperiments; ++j)
01534         e += calibrate_parse (i, j);
01535       g_mutex_lock (mutex);
```

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

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

```
01733             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01734         }
01735       for (i = 0; i < nthreads; ++i)
01736         g_thread_join (thread[i]);
01737     }
01738 #if HAVE_MPI
01739   // Communicating tasks results
01740   calibrate_synchronise ();
01741 #endif
01742 #if DEBUG
01743   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01744 #endif
01745 }
01746
01756 void
01757 calibrate_best_gradient (unsigned int simulation, double value)
01758 {
01759 #if DEBUG
01760   fprintf (stderr, "calibrate_best_gradient: start\n");
01761   fprintf (stderr,
01762            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01763            simulation, value, calibrate->error_best[0]);
01764 #endif
01765   if (value < calibrate->error_best[0])
01766     {
01767       calibrate->error_best[0] = value;
01768       calibrate->simulation_best[0] = simulation;
01769 #if DEBUG
01770       fprintf (stderr,
01771                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01772                simulation, value);
01773 #endif
01774     }
01775 #if DEBUG
01776   fprintf (stderr, "calibrate_best_gradient: end\n");
01777 #endif
01778 }
01779
01786 void
01787 calibrate_gradient_sequential (unsigned int simulation)
01788 {
01789   unsigned int i, j, k;
01790   double e;
01791 #if DEBUG
01792   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01793   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01794            "nend_gradient=%u\n",
01795            calibrate->nstart_gradient, calibrate->nend_gradient);
01796 #endif
01797   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01798     {
01799       k = simulation + i;
01800       e = 0.;
01801       for (j = 0; j < calibrate->nexperiments; ++j)
01802         e += calibrate_parse (k, j);
01803       calibrate_best_gradient (k, e);
01804       calibrate_save_variables (k, e);
01805 #if DEBUG
01806       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01807 #endif
01808     }
01809 #if DEBUG
01810   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01811 #endif
01812 }
01813
01821 void *
01822 calibrate_gradient_thread (ParallelData * data)
01823 {
01824   unsigned int i, j, thread;
01825   double e;
01826 #if DEBUG
01827   fprintf (stderr, "calibrate_gradient_thread: start\n");
01828 #endif
01829   thread = data->thread;
01830 #if DEBUG
01831   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01832            thread,
01833            calibrate->thread_gradient[thread],
01834            calibrate->thread_gradient[thread + 1]);
01835 #endif
01836   for (i = calibrate->thread_gradient[thread];
01837        i < calibrate->thread_gradient[thread + 1]; ++i)
01838     {
01839       e = 0.;
01840       for (j = 0; j < calibrate->nexperiments; ++j)
01841         e += calibrate_parse (i, j);
```

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

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

```
02043             }
02044           adjust = 0;
02045         }
02046     }
02047 #if DEBUG
02048   fprintf (stderr, "calibrate_gradient: end\n");
02049 #endif
02050 }
02051
02059 double
02060 calibrate_genetic_objective (Entity * entity)
02061 {
02062   unsigned int j;
02063   double objective;
02064   char buffer[64];
02065 #if DEBUG
02066   fprintf (stderr, "calibrate_genetic_objective: start\n");
02067 #endif
02068   for (j = 0; j < calibrate->nvariables; ++j)
02069     {
02070       calibrate->value[entity->id * calibrate->nvariables + j]
02071         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02072     }
02073   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02074     objective += calibrate_parse (entity->id, j);
02075   g_mutex_lock (mutex);
02076   for (j = 0; j < calibrate->nvariables; ++j)
02077     {
02078       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02079       fprintf (calibrate->file_variables, buffer,
02080                genetic_get_variable (entity, calibrate->genetic_variable + j));
02081     }
02082   fprintf (calibrate->file_variables, "%.14le\n", objective);
02083   g_mutex_unlock (mutex);
02084 #if DEBUG
02085   fprintf (stderr, "calibrate_genetic_objective: end\n");
02086 #endif
02087   return objective;
02088 }
02089
02094 void
02095 calibrate_genetic ()
02096 {
02097   char *best_genome;
02098   double best_objective, *best_variable;
02099 #if DEBUG
02100   fprintf (stderr, "calibrate_genetic: start\n");
02101   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02102            nthreads);
02103   fprintf (stderr,
02104            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02105            calibrate->nvariables, calibrate->nsimulations,
02106            calibrate->niterations);
02107   fprintf (stderr,
02108            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02109            calibrate->mutation_ratio, calibrate->
    reproduction_ratio,
02110            calibrate->adaptation_ratio);
02111 #endif
02112   genetic_algorithm_default (calibrate->nvariables,
02113                              calibrate->genetic_variable,
02114                              calibrate->nsimulations,
02115                              calibrate->niterations,
02116                              calibrate->mutation_ratio,
02117                              calibrate->reproduction_ratio,
02118                              calibrate->adaptation_ratio,
02119                              &calibrate_genetic_objective,
02120                              &best_genome, &best_variable, &best_objective);
02121 #if DEBUG
02122   fprintf (stderr, "calibrate_genetic: the best\n");
02123 #endif
02124   calibrate->error_old = (double *) g_malloc (sizeof (double));
02125   calibrate->value_old
02126     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02127   calibrate->error_old[0] = best_objective;
02128   memcpy (calibrate->value_old, best_variable,
02129           calibrate->nvariables * sizeof (double));
02130   g_free (best_genome);
02131   g_free (best_variable);
02132   calibrate_print ();
02133 #if DEBUG
02134   fprintf (stderr, "calibrate_genetic: end\n");
02135 #endif
02136 }
02137
02142 void
02143 calibrate_save_old ()
```

```
02144 {
02145   unsigned int i, j;
02146 #if DEBUG
02147   fprintf (stderr, "calibrate_save_old: start\n");
02148   fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02149 #endif
02150   memcpy (calibrate->error_old, calibrate->error_best,
02151           calibrate->nbest * sizeof (double));
02152   for (i = 0; i < calibrate->nbest; ++i)
02153     {
02154       j = calibrate->simulation_best[i];
02155 #if DEBUG
02156       fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02157 #endif
02158       memcpy (calibrate->value_old + i * calibrate->nvariables,
02159               calibrate->value + j * calibrate->nvariables,
02160               calibrate->nvariables * sizeof (double));
02161     }
02162 #if DEBUG
02163   for (i = 0; i < calibrate->nvariables; ++i)
02164     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02165              i, calibrate->value_old[i]);
02166   fprintf (stderr, "calibrate_save_old: end\n");
02167 #endif
02168 }
02169
02175 void
02176 calibrate_merge_old ()
02177 {
02178   unsigned int i, j, k;
02179   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
02180     *enew, *eold;
02181 #if DEBUG
02182   fprintf (stderr, "calibrate_merge_old: start\n");
02183 #endif
02184   enew = calibrate->error_best;
02185   eold = calibrate->error_old;
02186   i = j = k = 0;
02187   do
02188     {
02189       if (*enew < *eold)
02190         {
02191           memcpy (v + k * calibrate->nvariables,
02192                   calibrate->value
02193                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
02194                   calibrate->nvariables * sizeof (double));
02195           e[k] = *enew;
02196           ++k;
02197           ++enew;
02198           ++i;
02199         }
02200       else
02201         {
02202           memcpy (v + k * calibrate->nvariables,
02203                   calibrate->value_old + j * calibrate->nvariables,
02204                   calibrate->nvariables * sizeof (double));
02205           e[k] = *eold;
02206           ++k;
02207           ++eold;
02208           ++j;
02209         }
02210     }
02211   while (k < calibrate->nbest);
02212   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02213   memcpy (calibrate->error_old, e, k * sizeof (double));
02214 #if DEBUG
02215   fprintf (stderr, "calibrate_merge_old: end\n");
02216 #endif
02217 }
02218
02224 void
02225 calibrate_refine ()
02226 {
02227   unsigned int i, j;
02228   double d;
02229 #if HAVE_MPI
02230   MPI_Status mpi_stat;
02231 #endif
02232 #if DEBUG
02233   fprintf (stderr, "calibrate_refine: start\n");
02234 #endif
02235 #if HAVE_MPI
02236   if (!calibrate->mpi_rank)
02237     {
02238 #endif
```

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

```
02334   calibrate->error_old
02335     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02336   calibrate->value_old = (double *)
02337     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02338   calibrate_step ();
02339   calibrate_save_old ();
02340   calibrate_refine ();
02341   calibrate_print ();
02342   for (i = 1; i < calibrate->niterations; ++i)
02343     {
02344       calibrate_step ();
02345       calibrate_merge_old ();
02346       calibrate_refine ();
02347       calibrate_print ();
02348     }
02349 #if DEBUG
02350   fprintf (stderr, "calibrate_iterate: end\n");
02351 #endif
02352 }
02353
02358 void
02359 calibrate_free ()
02360 {
02361   unsigned int i, j;
02362 #if DEBUG
02363   fprintf (stderr, "calibrate_free: start\n");
02364 #endif
02365   for (j = 0; j < calibrate->ninputs; ++j)
02366     {
02367       for (i = 0; i < calibrate->nexperiments; ++i)
02368         g_mapped_file_unref (calibrate->file[j][i]);
02369       g_free (calibrate->file[j]);
02370     }
02371   g_free (calibrate->error_old);
02372   g_free (calibrate->value_old);
02373   g_free (calibrate->value);
02374   g_free (calibrate->genetic_variable);
02375   g_free (calibrate->rangemax);
02376   g_free (calibrate->rangemin);
02377 #if DEBUG
02378   fprintf (stderr, "calibrate_free: end\n");
02379 #endif
02380 }
02381
02386 void
02387 calibrate_open ()
02388 {
02389   GTimeZone *tz;
02390   GDateTime *t0, *t;
02391   unsigned int i, j, *nbits;
02392
02393 #if DEBUG
02394   char *buffer;
02395   fprintf (stderr, "calibrate_open: start\n");
02396 #endif
02397
02398   // Getting initial time
02399 #if DEBUG
02400   fprintf (stderr, "calibrate_open: getting initial time\n");
02401 #endif
02402   tz = g_time_zone_new_utc ();
02403   t0 = g_date_time_new_now (tz);
02404
02405   // Obtaining and initing the pseudo-random numbers generator seed
02406 #if DEBUG
02407   fprintf (stderr, "calibrate_open: getting initial seed\n");
02408 #endif
02409   calibrate->seed = input->seed;
02410   gsl_rng_set (calibrate->rng, calibrate->seed);
02411
02412   // Replacing the working directory
02413 #if DEBUG
02414   fprintf (stderr, "calibrate_open: replacing the working directory\n");
02415 #endif
02416   g_chdir (input->directory);
02417
02418   // Getting results file names
02419   calibrate->result = input->result;
02420   calibrate->variables = input->variables;
02421
02422   // Obtaining the simulator file
02423   calibrate->simulator = input->simulator;
02424
02425   // Obtaining the evaluator file
02426   calibrate->evaluator = input->evaluator;
02427
02428   // Reading the algorithm
```

```
02429   calibrate->algorithm = input->algorithm;
02430   switch (calibrate->algorithm)
02431     {
02432     case ALGORITHM_MONTE_CARLO:
02433       calibrate_algorithm = calibrate_MonteCarlo;
02434       break;
02435     case ALGORITHM_SWEEP:
02436       calibrate_algorithm = calibrate_sweep;
02437       break;
02438     default:
02439       calibrate_algorithm = calibrate_genetic;
02440       calibrate->mutation_ratio = input->mutation_ratio;
02441       calibrate->reproduction_ratio = input->
      reproduction_ratio;
02442       calibrate->adaptation_ratio = input->adaptation_ratio;
02443     }
02444   calibrate->nvariables = input->nvariables;
02445   calibrate->nsimulations = input->nsimulations;
02446   calibrate->niterations = input->niterations;
02447   calibrate->nbest = input->nbest;
02448   calibrate->tolerance = input->tolerance;
02449   calibrate->nsteps = input->nsteps;
02450   calibrate->nestimates = 0;
02451   if (input->nsteps)
02452     {
02453       calibrate->gradient_method = input->gradient_method;
02454       calibrate->relaxation = input->relaxation;
02455       switch (input->gradient_method)
02456         {
02457         case GRADIENT_METHOD_COORDINATES:
02458           calibrate->nestimates = 2 * calibrate->nvariables;
02459           calibrate_estimate_gradient =
      calibrate_estimate_gradient_coordinates;
02460           break;
02461         default:
02462           calibrate->nestimates = input->nestimates;
02463           calibrate_estimate_gradient =
      calibrate_estimate_gradient_random;
02464         }
02465     }
02466
02467 #if DEBUG
02468   fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02469 #endif
02470   calibrate->simulation_best
02471     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02472   calibrate->error_best
02473     = (double *) alloca (calibrate->nbest * sizeof (double));
02474
02475   // Reading the experimental data
02476 #if DEBUG
02477   buffer = g_get_current_dir ();
02478   fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02479   g_free (buffer);
02480 #endif
02481   calibrate->nexperiments = input->nexperiments;
02482   calibrate->ninputs = input->ninputs;
02483   calibrate->experiment = input->experiment;
02484   calibrate->weight = input->weight;
02485   for (i = 0; i < input->ninputs; ++i)
02486     {
02487       calibrate->template[i] = input->template[i];
02488       calibrate->file[i]
02489         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02490     }
02491   for (i = 0; i < input->nexperiments; ++i)
02492     {
02493 #if DEBUG
02494       fprintf (stderr, "calibrate_open: i=%u\n", i);
02495       fprintf (stderr, "calibrate_open: experiment=%s\n",
02496                calibrate->experiment[i]);
02497       fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
02498 #endif
02499       for (j = 0; j < input->ninputs; ++j)
02500         {
02501 #if DEBUG
02502           fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02503           fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
02504                    i, j + 1, calibrate->template[j][i]);
02505 #endif
02506           calibrate->file[j][i]
02507             = g_mapped_file_new (input->template[j][i], 0, NULL);
02508         }
02509     }
02510
02511   // Reading the variables data
02512 #if DEBUG
```

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

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

```
02685           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
02686         }
02687     }
02688 #if DEBUG
02689   fprintf (stderr, "input_save_gradient: end\n");
02690 #endif
02691 }
02692
02699 void
02700 input_save (char *filename)
02701 {
02702   unsigned int i, j;
02703   char *buffer;
02704   xmlDoc *doc;
02705   xmlNode *node, *child;
02706   GFile *file, *file2;
02707
02708 #if DEBUG
02709   fprintf (stderr, "input_save: start\n");
02710 #endif
02711
02712   // Getting the input file directory
02713   input->name = g_path_get_basename (filename);
02714   input->directory = g_path_get_dirname (filename);
02715   file = g_file_new_for_path (input->directory);
02716
02717   // Opening the input file
02718   doc = xmlNewDoc ((const xmlChar *) "1.0");
02719
02720   // Setting root XML node
02721   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02722   xmlDocSetRootElement (doc, node);
02723
02724   // Adding properties to the root XML node
02725   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02726     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02727   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02728     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02729   file2 = g_file_new_for_path (input->simulator);
02730   buffer = g_file_get_relative_path (file, file2);
02731   g_object_unref (file2);
02732   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02733   g_free (buffer);
02734   if (input->evaluator)
02735     {
02736       file2 = g_file_new_for_path (input->evaluator);
02737       buffer = g_file_get_relative_path (file, file2);
02738       g_object_unref (file2);
02739       if (xmlStrlen ((xmlChar *) buffer))
02740         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02741       g_free (buffer);
02742     }
02743   if (input->seed != DEFAULT_RANDOM_SEED)
02744     xml_node_set_uint (node, XML_SEED, input->seed);
02745
02746   // Setting the algorithm
02747   buffer = (char *) g_malloc (64);
02748   switch (input->algorithm)
02749     {
02750     case ALGORITHM_MONTE_CARLO:
02751       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02752       snprintf (buffer, 64, "%u", input->nsimulations);
02753       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02754       snprintf (buffer, 64, "%u", input->niterations);
02755       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02756       snprintf (buffer, 64, "%.3lg", input->tolerance);
02757       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02758       snprintf (buffer, 64, "%u", input->nbest);
02759       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02760       input_save_gradient (node);
02761       break;
02762     case ALGORITHM_SWEEP:
02763       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02764       snprintf (buffer, 64, "%u", input->niterations);
02765       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02766       snprintf (buffer, 64, "%.3lg", input->tolerance);
02767       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02768       snprintf (buffer, 64, "%u", input->nbest);
02769       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02770       input_save_gradient (node);
02771       break;
02772     default:
02773       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02774       snprintf (buffer, 64, "%u", input->nsimulations);
02775       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02776       snprintf (buffer, 64, "%u", input->niterations);
```

```
02777       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02778       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02779       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02780       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02781       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02782       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02783       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02784       break;
02785     }
02786   g_free (buffer);
02787
02788   // Setting the experimental data
02789   for (i = 0; i < input->nexperiments; ++i)
02790     {
02791       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02792       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02793       if (input->weight[i] != 1.)
02794         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02795       for (j = 0; j < input->ninputs; ++j)
02796         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02797     }
02798
02799   // Setting the variables data
02800   for (i = 0; i < input->nvariables; ++i)
02801     {
02802       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02803       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02804       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02805       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02806         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02807       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02808       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02809         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02810       if (input->precision[i] != DEFAULT_PRECISION)
02811         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02812       if (input->algorithm == ALGORITHM_SWEEP)
02813         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02814       else if (input->algorithm == ALGORITHM_GENETIC)
02815         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02816       if (input->nsteps)
02817         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02818     }
02819
02820   // Saving the XML file
02821   xmlSaveFormatFile (filename, doc, 1);
02822
02823   // Freeing memory
02824   xmlFreeDoc (doc);
02825
02826 #if DEBUG
02827   fprintf (stderr, "input_save: end\n");
02828 #endif
02829 }
02830
02835 void
02836 options_new ()
02837 {
02838 #if DEBUG
02839   fprintf (stderr, "options_new: start\n");
02840 #endif
02841   options->label_seed = (GtkLabel *)
02842     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02843   options->spin_seed = (GtkSpinButton *)
02844     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02845   gtk_widget_set_tooltip_text
02846     (GTK_WIDGET (options->spin_seed),
02847      gettext ("Seed to init the pseudo-random numbers generator"));
02848   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02849   options->label_threads = (GtkLabel *)
02850     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02851   options->spin_threads
02852     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02853   gtk_widget_set_tooltip_text
02854     (GTK_WIDGET (options->spin_threads),
02855      gettext ("Number of threads to perform the calibration/optimization for "
02856               "the stochastic algorithm"));
02857   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
```

```
02858   options->label_gradient = (GtkLabel *)
02859     gtk_label_new (gettext ("Threads number for the gradient based method"));
02860   options->spin_gradient
02861     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02862   gtk_widget_set_tooltip_text
02863     (GTK_WIDGET (options->spin_gradient),
02864      gettext ("Number of threads to perform the calibration/optimization for "
02865               "the gradient based method"));
02866   gtk_spin_button_set_value (options->spin_gradient,
02867                              (gdouble) nthreads_gradient);
02868   options->grid = (GtkGrid *) gtk_grid_new ();
02869   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
02870   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
02871   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
02872                    0, 1, 1, 1);
02873   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
02874                    1, 1, 1, 1);
02875   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
02876                    0, 2, 1, 1);
02877   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
02878                    1, 2, 1, 1);
02879   gtk_widget_show_all (GTK_WIDGET (options->grid));
02880   options->dialog = (GtkDialog *)
02881     gtk_dialog_new_with_buttons (gettext ("Options"),
02882                                  window->window,
02883                                  GTK_DIALOG_MODAL,
02884                                  gettext ("_OK"), GTK_RESPONSE_OK,
02885                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02886                                  NULL);
02887   gtk_container_add
02888     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02889      GTK_WIDGET (options->grid));
02890   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02891     {
02892       input->seed
02893         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02894       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
02895       nthreads_gradient
02896         = gtk_spin_button_get_value_as_int (options->spin_gradient);
02897     }
02898   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02899 #if DEBUG
02900   fprintf (stderr, "options_new: end\n");
02901 #endif
02902 }
02903
02908 void
02909 running_new ()
02910 {
02911 #if DEBUG
02912   fprintf (stderr, "running_new: start\n");
02913 #endif
02914   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02915   running->dialog = (GtkDialog *)
02916     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02917                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02918   gtk_container_add
02919     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02920      GTK_WIDGET (running->label));
02921   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02922 #if DEBUG
02923   fprintf (stderr, "running_new: end\n");
02924 #endif
02925 }
02926
02932 int
02933 window_get_algorithm ()
02934 {
02935   unsigned int i;
02936 #if DEBUG
02937   fprintf (stderr, "window_get_algorithm: start\n");
02938 #endif
02939   for (i = 0; i < NALGORITHMS; ++i)
02940     if (gtk_toggle_button_get_active
02941         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02942       break;
02943 #if DEBUG
02944   fprintf (stderr, "window_get_algorithm: %u\n", i);
02945   fprintf (stderr, "window_get_algorithm: end\n");
02946 #endif
02947   return i;
02948 }
02949
02955 int
02956 window_get_gradient ()
02957 {
02958   unsigned int i;
```

```
02959 #if DEBUG
02960   fprintf (stderr, "window_get_gradient: start\n");
02961 #endif
02962   for (i = 0; i < NGRADIENTS; ++i)
02963     if (gtk_toggle_button_get_active
02964         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02965       break;
02966 #if DEBUG
02967   fprintf (stderr, "window_get_gradient: %u\n", i);
02968   fprintf (stderr, "window_get_gradient: end\n");
02969 #endif
02970   return i;
02971 }
02972
02977 void
02978 window_save_gradient ()
02979 {
02980 #if DEBUG
02981   fprintf (stderr, "window_save_gradient: start\n");
02982 #endif
02983   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02984     {
02985       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02986       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
02987       switch (window_get_gradient ())
02988         {
02989         case GRADIENT_METHOD_COORDINATES:
02990           input->gradient_method = GRADIENT_METHOD_COORDINATES;
02991           break;
02992         default:
02993           input->gradient_method = GRADIENT_METHOD_RANDOM;
02994           input->nestimates
02995             = gtk_spin_button_get_value_as_int (window->spin_estimates);
02996         }
02997     }
02998   else
02999     input->nsteps = 0;
03000 #if DEBUG
03001   fprintf (stderr, "window_save_gradient: end\n");
03002 #endif
03003 }
03004
03010 int
03011 window_save ()
03012 {
03013   GtkFileChooserDialog *dlg;
03014   GtkFileFilter *filter;
03015   char *buffer;
03016
03017 #if DEBUG
03018   fprintf (stderr, "window_save: start\n");
03019 #endif
03020
03021   // Opening the saving dialog
03022   dlg = (GtkFileChooserDialog *)
03023     gtk_file_chooser_dialog_new (gettext ("Save file"),
03024                                  window->window,
03025                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03026                                  gettext ("_Cancel"),
03027                                  GTK_RESPONSE_CANCEL,
03028                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03029   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03030   buffer = g_build_filename (input->directory, input->name, NULL);
03031   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03032   g_free (buffer);
03033
03034   // Adding XML filter
03035   filter = (GtkFileFilter *) gtk_file_filter_new ();
03036   gtk_file_filter_set_name (filter, "XML");
03037   gtk_file_filter_add_pattern (filter, "*.xml");
03038   gtk_file_filter_add_pattern (filter, "*.XML");
03039   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
03040
03041   // If OK response then saving
03042   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03043     {
03044
03045       // Adding properties to the root XML node
03046       input->simulator = gtk_file_chooser_get_filename
03047         (GTK_FILE_CHOOSER (window->button_simulator));
03048       if (gtk_toggle_button_get_active
03049           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03050         input->evaluator = gtk_file_chooser_get_filename
03051           (GTK_FILE_CHOOSER (window->button_evaluator));
03052       else
03053         input->evaluator = NULL;
```

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

```
03141   gtk_widget_destroy (GTK_WIDGET (running->dialog));
03142   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03143   msg2 = g_strdup (buffer);
03144   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03145     {
03146       snprintf (buffer, 64, "%s = %s\n",
03147                 calibrate->label[i], format[calibrate->precision[i]]);
03148       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03149       msg = g_strconcat (msg2, buffer2, NULL);
03150       g_free (msg2);
03151     }
03152   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03153             calibrate->calculation_time);
03154   msg = g_strconcat (msg2, buffer, NULL);
03155   g_free (msg2);
03156   show_message (gettext ("Best result"), msg, INFO_TYPE);
03157   g_free (msg);
03158   calibrate_free ();
03159 #if DEBUG
03160   fprintf (stderr, "window_run: end\n");
03161 #endif
03162 }
03163
03168 void
03169 window_help ()
03170 {
03171   char *buffer, *buffer2;
03172 #if DEBUG
03173   fprintf (stderr, "window_help: start\n");
03174 #endif
03175   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03176                               gettext ("user-manual.pdf"), NULL);
03177   buffer = g_filename_to_uri (buffer2, NULL, NULL);
03178   g_free (buffer2);
03179   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03180 #if DEBUG
03181   fprintf (stderr, "window_help: uri=%s\n", buffer);
03182 #endif
03183   g_free (buffer);
03184 #if DEBUG
03185   fprintf (stderr, "window_help: end\n");
03186 #endif
03187 }
03188
03193 void
03194 window_about ()
03195 {
03196   static const gchar *authors[] = {
03197     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03198     "Borja Latorre Garcés <borja.latorre@csic.es>",
03199     NULL
03200   };
03201 #if DEBUG
03202   fprintf (stderr, "window_about: start\n");
03203 #endif
03204   gtk_show_about_dialog
03205     (window->window,
03206      "program_name", "MPCOTool",
03207      "comments",
03208      gettext ("A software to perform calibrations/optimizations of empirical "
03209               "parameters"),
03210      "authors", authors,
03211      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03212      "version", "1.2.4",
03213      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03214      "logo", window->logo,
03215      "website", "https://github.com/jburguete/mpcotool",
03216      "license-type", GTK_LICENSE_BSD, NULL);
03217 #if DEBUG
03218   fprintf (stderr, "window_about: end\n");
03219 #endif
03220 }
03221
03227 void
03228 window_update_gradient ()
03229 {
03230 #if DEBUG
03231   fprintf (stderr, "window_update_gradient: start\n");
03232 #endif
03233   gtk_widget_show (GTK_WIDGET (window->check_gradient));
03234   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03235     {
03236       gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03237       gtk_widget_show (GTK_WIDGET (window->label_step));
03238       gtk_widget_show (GTK_WIDGET (window->spin_step));
03239     }
03240   switch (window_get_gradient ())
```

```
03241     {
03242     case GRADIENT_METHOD_COORDINATES:
03243       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03244       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03245       break;
03246     default:
03247       gtk_widget_show (GTK_WIDGET (window->label_estimates));
03248       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03249     }
03250 #if DEBUG
03251   fprintf (stderr, "window_update_gradient: end\n");
03252 #endif
03253 }
03254
03259 void
03260 window_update ()
03261 {
03262   unsigned int i;
03263 #if DEBUG
03264   fprintf (stderr, "window_update: start\n");
03265 #endif
03266   gtk_widget_set_sensitive
03267     (GTK_WIDGET (window->button_evaluator),
03268      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03269                                    (window->check_evaluator)));
03270   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03271   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03272   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03273   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03274   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03275   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03276   gtk_widget_hide (GTK_WIDGET (window->label_bests));
03277   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03278   gtk_widget_hide (GTK_WIDGET (window->label_population));
03279   gtk_widget_hide (GTK_WIDGET (window->spin_population));
03280   gtk_widget_hide (GTK_WIDGET (window->label_generations));
03281   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03282   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03283   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03284   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03285   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03286   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03287   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03288   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03289   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03290   gtk_widget_hide (GTK_WIDGET (window->label_bits));
03291   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03292   gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03293   gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03294   gtk_widget_hide (GTK_WIDGET (window->label_step));
03295   gtk_widget_hide (GTK_WIDGET (window->spin_step));
03296   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03297   switch (window_get_algorithm ())
03298     {
03299     case ALGORITHM_MONTE_CARLO:
03300       gtk_widget_show (GTK_WIDGET (window->label_simulations));
03301       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03302       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03303       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03304       if (i > 1)
03305         {
03306           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03307           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03308           gtk_widget_show (GTK_WIDGET (window->label_bests));
03309           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03310         }
03311       window_update_gradient ();
03312       break;
03313     case ALGORITHM_SWEEP:
03314       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03315       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03316       if (i > 1)
03317         {
03318           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03319           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03320           gtk_widget_show (GTK_WIDGET (window->label_bests));
03321           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03322         }
03323       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03324       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03325       gtk_widget_show (GTK_WIDGET (window->check_gradient));
03326       window_update_gradient ();
03327       break;
03328     default:
03329       gtk_widget_show (GTK_WIDGET (window->label_population));
03330       gtk_widget_show (GTK_WIDGET (window->spin_population));
03331       gtk_widget_show (GTK_WIDGET (window->label_generations));
```

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

```
03419     case ALGORITHM_SWEEP:
03420       input->nsweeps = (unsigned int *) g_realloc
03421         (input->nsweeps, input->nvariables * sizeof (unsigned int));
03422       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03423       if (i < 0)
03424         i = 0;
03425       gtk_spin_button_set_value (window->spin_sweeps,
03426                                  (gdouble) input->nsweeps[i]);
03427       break;
03428     case ALGORITHM_GENETIC:
03429       input->nbits = (unsigned int *) g_realloc
03430         (input->nbits, input->nvariables * sizeof (unsigned int));
03431       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03432       if (i < 0)
03433         i = 0;
03434       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03435     }
03436   window_update ();
03437 #if DEBUG
03438   fprintf (stderr, "window_set_algorithm: end\n");
03439 #endif
03440 }
03441
03446 void
03447 window_set_experiment ()
03448 {
03449   unsigned int i, j;
03450   char *buffer1, *buffer2;
03451 #if DEBUG
03452   fprintf (stderr, "window_set_experiment: start\n");
03453 #endif
03454   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03455   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03456   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03457   buffer2 = g_build_filename (input->directory, buffer1, NULL);
03458   g_free (buffer1);
03459   g_signal_handler_block
03460     (window->button_experiment, window->id_experiment_name);
03461   gtk_file_chooser_set_filename
03462     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03463   g_signal_handler_unblock
03464     (window->button_experiment, window->id_experiment_name);
03465   g_free (buffer2);
03466   for (j = 0; j < input->ninputs; ++j)
03467     {
03468       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03469       buffer2
03470         = g_build_filename (input->directory, input->template[j][i], NULL);
03471       gtk_file_chooser_set_filename
03472         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03473       g_free (buffer2);
03474       g_signal_handler_unblock
03475         (window->button_template[j], window->id_input[j]);
03476     }
03477 #if DEBUG
03478   fprintf (stderr, "window_set_experiment: end\n");
03479 #endif
03480 }
03481
03486 void
03487 window_remove_experiment ()
03488 {
03489   unsigned int i, j;
03490 #if DEBUG
03491   fprintf (stderr, "window_remove_experiment: start\n");
03492 #endif
03493   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03494   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03495   gtk_combo_box_text_remove (window->combo_experiment, i);
03496   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03497   xmlFree (input->experiment[i]);
03498   --input->nexperiments;
03499   for (j = i; j < input->nexperiments; ++j)
03500     {
03501       input->experiment[j] = input->experiment[j + 1];
03502       input->weight[j] = input->weight[j + 1];
03503     }
03504   j = input->nexperiments - 1;
03505   if (i > j)
03506     i = j;
03507   for (j = 0; j < input->ninputs; ++j)
03508     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
```

```
03509   g_signal_handler_block
03510     (window->button_experiment, window->id_experiment_name);
03511   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03512   g_signal_handler_unblock
03513     (window->button_experiment, window->id_experiment_name);
03514   for (j = 0; j < input->ninputs; ++j)
03515     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03516   window_update ();
03517 #if DEBUG
03518   fprintf (stderr, "window_remove_experiment: end\n");
03519 #endif
03520 }
03521
03526 void
03527 window_add_experiment ()
03528 {
03529   unsigned int i, j;
03530 #if DEBUG
03531   fprintf (stderr, "window_add_experiment: start\n");
03532 #endif
03533   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03534   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03535   gtk_combo_box_text_insert_text
03536     (window->combo_experiment, i, input->experiment[i]);
03537   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03538   input->experiment = (char **) g_realloc
03539     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03540   input->weight = (double *) g_realloc
03541     (input->weight, (input->nexperiments + 1) * sizeof (double));
03542   for (j = input->nexperiments - 1; j > i; --j)
03543     {
03544       input->experiment[j + 1] = input->experiment[j];
03545       input->weight[j + 1] = input->weight[j];
03546     }
03547   input->experiment[j + 1]
03548     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03549   input->weight[j + 1] = input->weight[j];
03550   ++input->nexperiments;
03551   for (j = 0; j < input->ninputs; ++j)
03552     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03553   g_signal_handler_block
03554     (window->button_experiment, window->id_experiment_name);
03555   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03556   g_signal_handler_unblock
03557     (window->button_experiment, window->id_experiment_name);
03558   for (j = 0; j < input->ninputs; ++j)
03559     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03560   window_update ();
03561 #if DEBUG
03562   fprintf (stderr, "window_add_experiment: end\n");
03563 #endif
03564 }
03565
03570 void
03571 window_name_experiment ()
03572 {
03573   unsigned int i;
03574   char *buffer;
03575   GFile *file1, *file2;
03576 #if DEBUG
03577   fprintf (stderr, "window_name_experiment: start\n");
03578 #endif
03579   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03580   file1
03581     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03582   file2 = g_file_new_for_path (input->directory);
03583   buffer = g_file_get_relative_path (file2, file1);
03584   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03585   gtk_combo_box_text_remove (window->combo_experiment, i);
03586   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03587   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03588   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03589   g_free (buffer);
03590   g_object_unref (file2);
03591   g_object_unref (file1);
03592 #if DEBUG
03593   fprintf (stderr, "window_name_experiment: end\n");
03594 #endif
03595 }
03596
```

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

```
03701         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03702     }
03703   else
03704     {
03705       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03706       gtk_toggle_button_set_active
03707         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03708     }
03709   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03710     {
03711       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
03712       gtk_toggle_button_set_active
03713         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03714     }
03715   else
03716     {
03717       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03718       gtk_toggle_button_set_active
03719         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03720     }
03721   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
03722   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
03723   if (input->nsteps)
03724     gtk_spin_button_set_value (window->spin_step, input->step[i]);
03725 #if DEBUG
03726   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03727            input->precision[i]);
03728 #endif
03729   switch (window_get_algorithm ())
03730     {
03731     case ALGORITHM_SWEEP:
03732       gtk_spin_button_set_value (window->spin_sweeps,
03733                                  (gdouble) input->nsweeps[i]);
03734 #if DEBUG
03735       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03736                input->nsweeps[i]);
03737 #endif
03738       break;
03739     case ALGORITHM_GENETIC:
03740       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03741 #if DEBUG
03742       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03743                input->nbits[i]);
03744 #endif
03745       break;
03746     }
03747   window_update ();
03748 #if DEBUG
03749   fprintf (stderr, "window_set_variable: end\n");
03750 #endif
03751 }
03752
03757 void
03758 window_remove_variable ()
03759 {
03760   unsigned int i, j;
03761 #if DEBUG
03762   fprintf (stderr, "window_remove_variable: start\n");
03763 #endif
03764   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03765   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03766   gtk_combo_box_text_remove (window->combo_variable, i);
03767   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03768   xmlFree (input->label[i]);
03769   --input->nvariables;
03770   for (j = i; j < input->nvariables; ++j)
03771     {
03772       input->label[j] = input->label[j + 1];
03773       input->rangemin[j] = input->rangemin[j + 1];
03774       input->rangemax[j] = input->rangemax[j + 1];
03775       input->rangeminabs[j] = input->rangeminabs[j + 1];
03776       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03777       input->precision[j] = input->precision[j + 1];
03778       input->step[j] = input->step[j + 1];
03779       switch (window_get_algorithm ())
03780         {
03781         case ALGORITHM_SWEEP:
03782           input->nsweeps[j] = input->nsweeps[j + 1];
03783           break;
03784         case ALGORITHM_GENETIC:
03785           input->nbits[j] = input->nbits[j + 1];
```

```
03786         }
03787     }
03788   j = input->nvariables - 1;
03789   if (i > j)
03790     i = j;
03791   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03792   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03793   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03794   window_update ();
03795 #if DEBUG
03796   fprintf (stderr, "window_remove_variable: end\n");
03797 #endif
03798 }
03799
03804 void
03805 window_add_variable ()
03806 {
03807   unsigned int i, j;
03808 #if DEBUG
03809   fprintf (stderr, "window_add_variable: start\n");
03810 #endif
03811   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03812   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03813   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03814   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03815   input->label = (char **) g_realloc
03816     (input->label, (input->nvariables + 1) * sizeof (char *));
03817   input->rangemin = (double *) g_realloc
03818     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03819   input->rangemax = (double *) g_realloc
03820     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03821   input->rangeminabs = (double *) g_realloc
03822     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03823   input->rangemaxabs = (double *) g_realloc
03824     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03825   input->precision = (unsigned int *) g_realloc
03826     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03827   input->step = (double *) g_realloc
03828     (input->step, (input->nvariables + 1) * sizeof (double));
03829   for (j = input->nvariables - 1; j > i; --j)
03830     {
03831       input->label[j + 1] = input->label[j];
03832       input->rangemin[j + 1] = input->rangemin[j];
03833       input->rangemax[j + 1] = input->rangemax[j];
03834       input->rangeminabs[j + 1] = input->rangeminabs[j];
03835       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03836       input->precision[j + 1] = input->precision[j];
03837       input->step[j + 1] = input->step[j];
03838     }
03839   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03840   input->rangemin[j + 1] = input->rangemin[j];
03841   input->rangemax[j + 1] = input->rangemax[j];
03842   input->rangeminabs[j + 1] = input->rangeminabs[j];
03843   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03844   input->precision[j + 1] = input->precision[j];
03845   input->step[j + 1] = input->step[j];
03846   switch (window_get_algorithm ())
03847     {
03848     case ALGORITHM_SWEEP:
03849       input->nsweeps = (unsigned int *) g_realloc
03850         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03851       for (j = input->nvariables - 1; j > i; --j)
03852         input->nsweeps[j + 1] = input->nsweeps[j];
03853       input->nsweeps[j + 1] = input->nsweeps[j];
03854       break;
03855     case ALGORITHM_GENETIC:
03856       input->nbits = (unsigned int *) g_realloc
03857         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03858       for (j = input->nvariables - 1; j > i; --j)
03859         input->nbits[j + 1] = input->nbits[j];
03860       input->nbits[j + 1] = input->nbits[j];
03861     }
03862   ++input->nvariables;
03863   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03864   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03865   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03866   window_update ();
03867 #if DEBUG
03868   fprintf (stderr, "window_add_variable: end\n");
03869 #endif
```

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

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

```
04079                                     (window->button_evaluator), buffer);
04080       g_free (buffer);
04081     }
04082   gtk_toggle_button_set_active
04083     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04084   switch (input->algorithm)
04085     {
04086     case ALGORITHM_MONTE_CARLO:
04087       gtk_spin_button_set_value (window->spin_simulations,
04088                                  (gdouble) input->nsimulations);
04089     case ALGORITHM_SWEEP:
04090       gtk_spin_button_set_value (window->spin_iterations,
04091                                  (gdouble) input->niterations);
04092       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
04093       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
04094       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
04095                                     input->nsteps);
04096       if (input->nsteps)
04097         {
04098           gtk_toggle_button_set_active
04099             (GTK_TOGGLE_BUTTON (window->button_gradient
04100                                 [input->gradient_method]), TRUE);
04101           gtk_spin_button_set_value (window->spin_steps,
04102                                      (gdouble) input->nsteps);
04103           gtk_spin_button_set_value (window->spin_relaxation,
04104                                      (gdouble) input->relaxation);
04105           switch (input->gradient_method)
04106             {
04107             case GRADIENT_METHOD_RANDOM:
04108               gtk_spin_button_set_value (window->spin_estimates,
04109                                          (gdouble) input->nestimates);
04110             }
04111         }
04112       break;
04113     default:
04114       gtk_spin_button_set_value (window->spin_population,
04115                                  (gdouble) input->nsimulations);
04116       gtk_spin_button_set_value (window->spin_generations,
04117                                  (gdouble) input->niterations);
04118       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04119       gtk_spin_button_set_value (window->spin_reproduction,
04120                                  input->reproduction_ratio);
04121       gtk_spin_button_set_value (window->spin_adaptation,
04122                                  input->adaptation_ratio);
04123     }
04124   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04125   g_signal_handler_block (window->button_experiment,
04126                           window->id_experiment_name);
04127   gtk_combo_box_text_remove_all (window->combo_experiment);
04128   for (i = 0; i < input->nexperiments; ++i)
04129     gtk_combo_box_text_append_text (window->combo_experiment,
04130                                     input->experiment[i]);
04131   g_signal_handler_unblock
04132     (window->button_experiment, window->id_experiment_name);
04133   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
04134   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04135   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04136   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04137   gtk_combo_box_text_remove_all (window->combo_variable);
04138   for (i = 0; i < input->nvariables; ++i)
04139     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
04140   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04141   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04142   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04143   window_set_variable ();
04144   window_update ();
04145
04146 #if DEBUG
04147   fprintf (stderr, "window_read: end\n");
04148 #endif
04149   return 1;
04150 }
04151
04156 void
04157 window_open ()
04158 {
```

```
04159   GtkFileChooserDialog *dlg;
04160   GtkFileFilter *filter;
04161   char *buffer, *directory, *name;
04162
04163 #if DEBUG
04164   fprintf (stderr, "window_open: start\n");
04165 #endif
04166
04167   // Saving a backup of the current input file
04168   directory = g_strdup (input->directory);
04169   name = g_strdup (input->name);
04170
04171   // Opening dialog
04172   dlg = (GtkFileChooserDialog *)
04173     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04174                                  window->window,
04175                                  GTK_FILE_CHOOSER_ACTION_OPEN,
04176                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04177                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04178
04179   // Adding XML filter
04180   filter = (GtkFileFilter *) gtk_file_filter_new ();
04181   gtk_file_filter_set_name (filter, "XML");
04182   gtk_file_filter_add_pattern (filter, "*.xml");
04183   gtk_file_filter_add_pattern (filter, "*.XML");
04184   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
04185
04186   // If OK saving
04187   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04188     {
04189
04190       // Traying to open the input file
04191       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04192       if (!window_read (buffer))
04193         {
04194 #if DEBUG
04195           fprintf (stderr, "window_open: error reading input file\n");
04196 #endif
04197           g_free (buffer);
04198
04199           // Reading backup file on error
04200           buffer = g_build_filename (directory, name, NULL);
04201           if (!input_open (buffer))
04202             {
04203
04204               // Closing on backup file reading error
04205 #if DEBUG
04206               fprintf (stderr, "window_read: error reading backup file\n");
04207 #endif
04208               g_free (buffer);
04209               break;
04210             }
04211           g_free (buffer);
04212         }
04213       else
04214         {
04215           g_free (buffer);
04216           break;
04217         }
04218     }
04219
04220   // Freeing and closing
04221   g_free (name);
04222   g_free (directory);
04223   gtk_widget_destroy (GTK_WIDGET (dlg));
04224 #if DEBUG
04225   fprintf (stderr, "window_open: end\n");
04226 #endif
04227 }
04228
04233 void
04234 window_new ()
04235 {
04236   unsigned int i;
04237   char *buffer, *buffer2, buffer3[64];
04238   char *label_algorithm[NALGORITHMS] = {
04239     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04240   };
04241   char *tip_algorithm[NALGORITHMS] = {
04242     gettext ("Monte-Carlo brute force algorithm"),
04243     gettext ("Sweep brute force algorithm"),
04244     gettext ("Genetic algorithm")
04245   };
04246   char *label_gradient[NGRADIENTS] = {
04247     gettext ("_Coordinates descent"), gettext ("_Random")
04248   };
04249   char *tip_gradient[NGRADIENTS] = {
```

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

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

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

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

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

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

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

```
04836                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04837   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
04838
04839   // Setting the window logo
04840   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04841   gtk_window_set_icon (window->window, window->logo);
04842
04843   // Showing the window
04844   gtk_widget_show_all (GTK_WIDGET (window->window));
04845
04846   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04847 #if GTK_MINOR_VERSION >= 16
04848   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04849   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04850   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04851   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04852   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
04853 #endif
04854
04855   // Reading initial example
04856   input_new ();
04857   buffer2 = g_get_current_dir ();
04858   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04859   g_free (buffer2);
04860   window_read (buffer);
04861   g_free (buffer);
04862
04863 #if DEBUG
04864   fprintf (stderr, "window_new: start\n");
04865 #endif
04866 }
04867
04868 #endif
04869
04875 int
04876 cores_number ()
04877 {
04878 #ifdef G_OS_WIN32
04879   SYSTEM_INFO sysinfo;
04880   GetSystemInfo (&sysinfo);
04881   return sysinfo.dwNumberOfProcessors;
04882 #else
04883   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04884 #endif
04885 }
04886
04896 int
04897 main (int argn, char **argc)
04898 {
04899 #if HAVE_GTK
04900   char *buffer;
04901 #endif
04902
04903   // Starting pseudo-random numbers generator
04904   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04905   calibrate->seed = DEFAULT_RANDOM_SEED;
04906
04907   // Allowing spaces in the XML data file
04908   xmlKeepBlanksDefault (0);
04909
04910   // Starting MPI
04911 #if HAVE_MPI
04912   MPI_Init (&argn, &argc);
04913   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04914   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04915   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04916 #else
04917   ntasks = 1;
04918 #endif
04919
04920 #if HAVE_GTK
04921
04922   // Getting threads number
04923   nthreads_gradient = nthreads = cores_number ();
04924
04925   // Setting local language and international floating point numbers notation
04926   setlocale (LC_ALL, "");
04927   setlocale (LC_NUMERIC, "C");
04928   window->application_directory = g_get_current_dir ();
04929   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04930   bindtextdomain (PROGRAM_INTERFACE, buffer);
04931   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04932   textdomain (PROGRAM_INTERFACE);
04933
04934   // Initing GTK+
```

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

5.7 mpcotool.h File Reference

Header file of the mpcotool.

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

Data Structures

- struct Input

  Struct to define the calibration input file.

- struct Calibrate

  Struct to define the calibration data.

- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

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

  Enum to define the methods to estimate the gradient.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to calibrate sequentially.

- void * calibrate_thread (ParallelData *data)

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

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

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

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

  Function to calibrate with the Monte-Carlo algorithm.
- void calibrate_best_gradient (unsigned int simulation, double value)

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

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

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

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

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

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_open ()

  Function to open and perform a calibration.

5.7.1 Detailed Description

Header file of the mpcotool.

Authors

Javier Burguete.

Copyright

Definition in file mpcotool.h.

5.7.2 Enumeration Type Documentation

5.7.2.1 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

Definition at line 43 of file mpcotool.h.

```
00044 {
00045   ALGORITHM_MONTE_CARLO = 0,
00046   ALGORITHM_SWEEP = 1,
00047   ALGORITHM_GENETIC = 2
00048 };
```

5.7.2.2 enum GradientMethod

Enum to define the methods to estimate the gradient.

Enumerator

GRADIENT_METHOD_COORDINATES Coordinates descent method.

GRADIENT_METHOD_RANDOM Random method.

Definition at line 54 of file mpcotool.h.

```
00055 {
00056   GRADIENT_METHOD_COORDINATES = 0,
00057   GRADIENT_METHOD_RANDOM = 1,
00058 };
```

5.7.3 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1444 of file mpcotool.c.

```
01445 {
01446   unsigned int i, j;
01447   double e;
01448 #if DEBUG
01449   fprintf (stderr, "calibrate_best: start\n");
01450   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01451            calibrate->nsaveds, calibrate->nbest);
01452 #endif
01453   if (calibrate->nsaveds < calibrate->nbest
01454       || value < calibrate->error_best[calibrate->nsaveds - 1])
01455     {
01456       if (calibrate->nsaveds < calibrate->nbest)
01457         ++calibrate->nsaveds;
01458       calibrate->error_best[calibrate->nsaveds - 1] = value;
01459       calibrate->simulation_best[calibrate->
```

```
      nsaveds - 1] = simulation;
01460       for (i = calibrate->nsaveds; --i;)
01461         {
01462           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01463             {
01464               j = calibrate->simulation_best[i];
01465               e = calibrate->error_best[i];
01466               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01467               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01468               calibrate->simulation_best[i - 1] = j;
01469               calibrate->error_best[i - 1] = e;
01470             }
01471           else
01472             break;
01473         }
01474     }
01475 #if DEBUG
01476   fprintf (stderr, "calibrate_best: end\n");
01477 #endif
01478 }
```

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 1757 of file mpcotool.c.

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

5.7.3.3 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 2060 of file mpcotool.c.

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

Here is the call graph for this function:

5.7.3.4 void* calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 1822 of file mpcotool.c.

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

Here is the call graph for this function:

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

Function to write the simulation input file.

Parameters

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

Definition at line 1197 of file mpcotool.c.

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

```
01264   fclose (file);
01265
01266 calibrate_input_end:
01267 #if DEBUG
01268   fprintf (stderr, "calibrate_input: end\n");
01269 #endif
01270   return;
01271 }
```

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 1562 of file mpcotool.c.

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

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

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1284 of file mpcotool.c.

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

```
01356     }
01357
01358   // Removing files
01359 #if !DEBUG
01360   for (i = 0; i < calibrate->ninputs; ++i)
01361     {
01362       if (calibrate->file[i][0])
01363         {
01364           snprintf (buffer, 512, RM " %s", &input[i][0]);
01365           system (buffer);
01366         }
01367     }
01368   snprintf (buffer, 512, RM " %s %s", output, result);
01369   system (buffer);
01370 #endif
01371
01372 #if DEBUG
01373   fprintf (stderr, "calibrate_parse: end\n");
01374 #endif
01375
01376   // Returning the objective function
01377   return e * calibrate->weight[experiment];
01378 }
```

Here is the call graph for this function:

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

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

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1416 of file mpcotool.c.

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

5.7.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

simulation	Simulation number.

Definition at line 1924 of file mpcotool.c.

```
01925 {
01926   GThread *thread[nthreads_gradient];
01927   ParallelData data[nthreads_gradient];
01928   unsigned int i, j, k, b;
01929 #if DEBUG
01930   fprintf (stderr, "calibrate_step_gradient: start\n");
01931 #endif
01932   for (i = 0; i < calibrate->nestimates; ++i)
```

```
01933     {
01934       k = (simulation + i) * calibrate->nvariables;
01935       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01936 #if DEBUG
01937       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01938                simulation + i, calibrate->simulation_best[0]);
01939 #endif
01940       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01941         {
01942 #if DEBUG
01943           fprintf (stderr,
01944                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01945                    i, j, calibrate->value[b]);
01946 #endif
01947           calibrate->value[k]
01948             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01949           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01950                                             calibrate->rangeminabs[j]),
01951                                       calibrate->rangemaxabs[j]);
01952 #if DEBUG
01953           fprintf (stderr,
01954                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01955                    i, j, calibrate->value[k]);
01956 #endif
01957         }
01958     }
01959   if (nthreads_gradient == 1)
01960     calibrate_gradient_sequential (simulation);
01961   else
01962     {
01963       for (i = 0; i <= nthreads_gradient; ++i)
01964         {
01965           calibrate->thread_gradient[i]
01966             = simulation + calibrate->nstart_gradient
01967             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01968             / nthreads_gradient;
01969 #if DEBUG
01970           fprintf (stderr,
01971                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01972                    i, calibrate->thread_gradient[i]);
01973 #endif
01974         }
01975       for (i = 0; i < nthreads_gradient; ++i)
01976         {
01977           data[i].thread = i;
01978           thread[i] = g_thread_new
01979             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01980         }
01981       for (i = 0; i < nthreads_gradient; ++i)
01982         g_thread_join (thread[i]);
01983     }
01984 #if DEBUG
01985   fprintf (stderr, "calibrate_step_gradient: end\n");
01986 #endif
01987 }
```

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 1518 of file mpcotool.c.

```
01519 {
01520   unsigned int i, j, thread;
```

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

Here is the call graph for this function:

5.7.3.11 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 549 of file mpcotool.c.

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

```
00583 #endif
00584   node = xmlDocGetRootElement (doc);
00585   if (xmlStrcmp (node->name, XML_CALIBRATE))
00586     {
00587       msg = gettext ("Bad root XML node");
00588       goto exit_on_error;
00589     }
00590
00591   // Getting results file names
00592   input->result = (char *) xmlGetProp (node, XML_RESULT);
00593   if (!input->result)
00594     input->result = (char *) xmlStrdup (result_name);
00595   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00596   if (!input->variables)
00597     input->variables = (char *) xmlStrdup (variables_name);
00598
00599   // Opening simulator program name
00600   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00601   if (!input->simulator)
00602     {
00603       msg = gettext ("Bad simulator program");
00604       goto exit_on_error;
00605     }
00606
00607   // Opening evaluator program name
00608   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00609
00610   // Obtaining pseudo-random numbers generator seed
00611   input->seed
00612     = xml_node_get_uint_with_default (node,
      XML_SEED, DEFAULT_RANDOM_SEED,
00613                                       &error_code);
00614   if (error_code)
00615     {
00616       msg = gettext ("Bad pseudo-random numbers generator seed");
00617       goto exit_on_error;
00618     }
00619
00620   // Opening algorithm
00621   buffer = xmlGetProp (node, XML_ALGORITHM);
00622   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00623     {
00624       input->algorithm = ALGORITHM_MONTE_CARLO;
00625
00626       // Obtaining simulations number
00627       input->nsimulations
00628         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00629       if (error_code)
00630         {
00631           msg = gettext ("Bad simulations number");
00632           goto exit_on_error;
00633         }
00634     }
00635   else if (!xmlStrcmp (buffer, XML_SWEEP))
00636     input->algorithm = ALGORITHM_SWEEP;
00637   else if (!xmlStrcmp (buffer, XML_GENETIC))
00638     {
00639       input->algorithm = ALGORITHM_GENETIC;
00640
00641       // Obtaining population
00642       if (xmlHasProp (node, XML_NPOPULATION))
00643         {
00644           input->nsimulations
00645             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00646           if (error_code || input->nsimulations < 3)
00647             {
00648               msg = gettext ("Invalid population number");
00649               goto exit_on_error;
00650             }
00651         }
00652       else
00653         {
00654           msg = gettext ("No population number");
00655           goto exit_on_error;
00656         }
00657
00658       // Obtaining generations
00659       if (xmlHasProp (node, XML_NGENERATIONS))
00660         {
00661           input->niterations
00662             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00663           if (error_code || !input->niterations)
00664             {
00665               msg = gettext ("Invalid generations number");
00666               goto exit_on_error;
00667             }
00668         }
```

```
00669       else
00670         {
00671           msg = gettext ("No generations number");
00672           goto exit_on_error;
00673         }
00674
00675       // Obtaining mutation probability
00676       if (xmlHasProp (node, XML_MUTATION))
00677         {
00678           input->mutation_ratio
00679             = xml_node_get_float (node, XML_MUTATION, &error_code);
00680           if (error_code || input->mutation_ratio < 0.
00681               || input->mutation_ratio >= 1.)
00682             {
00683               msg = gettext ("Invalid mutation probability");
00684               goto exit_on_error;
00685             }
00686         }
00687       else
00688         {
00689           msg = gettext ("No mutation probability");
00690           goto exit_on_error;
00691         }
00692
00693       // Obtaining reproduction probability
00694       if (xmlHasProp (node, XML_REPRODUCTION))
00695         {
00696           input->reproduction_ratio
00697             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00698           if (error_code || input->reproduction_ratio < 0.
00699               || input->reproduction_ratio >= 1.0)
00700             {
00701               msg = gettext ("Invalid reproduction probability");
00702               goto exit_on_error;
00703             }
00704         }
00705       else
00706         {
00707           msg = gettext ("No reproduction probability");
00708           goto exit_on_error;
00709         }
00710
00711       // Obtaining adaptation probability
00712       if (xmlHasProp (node, XML_ADAPTATION))
00713         {
00714           input->adaptation_ratio
00715             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00716           if (error_code || input->adaptation_ratio < 0.
00717               || input->adaptation_ratio >= 1.)
00718             {
00719               msg = gettext ("Invalid adaptation probability");
00720               goto exit_on_error;
00721             }
00722         }
00723       else
00724         {
00725           msg = gettext ("No adaptation probability");
00726           goto exit_on_error;
00727         }
00728
00729       // Checking survivals
00730       i = input->mutation_ratio * input->nsimulations;
00731       i += input->reproduction_ratio * input->
     nsimulations;
00732       i += input->adaptation_ratio * input->
     nsimulations;
00733       if (i > input->nsimulations - 2)
00734         {
00735           msg = gettext
00736             ("No enough survival entities to reproduce the population");
00737           goto exit_on_error;
00738         }
00739     }
00740   else
00741     {
00742       msg = gettext ("Unknown algorithm");
00743       goto exit_on_error;
00744     }
00745   xmlFree (buffer);
00746   buffer = NULL;
00747
00748   if (input->algorithm == ALGORITHM_MONTE_CARLO
00749       || input->algorithm == ALGORITHM_SWEEP)
00750     {
00751
00752       // Obtaining iterations number
00753       input->niterations
```

```
00754         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00755       if (error_code == 1)
00756         input->niterations = 1;
00757       else if (error_code)
00758         {
00759           msg = gettext ("Bad iterations number");
00760           goto exit_on_error;
00761         }
00762
00763       // Obtaining best number
00764       input->nbest
00765         = xml_node_get_uint_with_default (node,
      XML_NBEST, 1, &error_code);
00766       if (error_code || !input->nbest)
00767         {
00768           msg = gettext ("Invalid best number");
00769           goto exit_on_error;
00770         }
00771
00772       // Obtaining tolerance
00773       input->tolerance
00774         = xml_node_get_float_with_default (node,
      XML_TOLERANCE, 0.,
00775                                            &error_code);
00776       if (error_code || input->tolerance < 0.)
00777         {
00778           msg = gettext ("Invalid tolerance");
00779           goto exit_on_error;
00780         }
00781
00782       // Getting gradient method parameters
00783       if (xmlHasProp (node, XML_NSTEPS))
00784         {
00785           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00786           if (error_code || !input->nsteps)
00787             {
00788               msg = gettext ("Invalid steps number");
00789               goto exit_on_error;
00790             }
00791           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00792           if (!xmlStrcmp (buffer, XML_COORDINATES))
00793             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00794           else if (!xmlStrcmp (buffer, XML_RANDOM))
00795             {
00796               input->gradient_method =
      GRADIENT_METHOD_RANDOM;
00797               input->nestimates
00798                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00799               if (error_code || !input->nestimates)
00800                 {
00801                   msg = gettext ("Invalid estimates number");
00802                   goto exit_on_error;
00803                 }
00804             }
00805           else
00806             {
00807               msg = gettext ("Unknown method to estimate the gradient");
00808               goto exit_on_error;
00809             }
00810           xmlFree (buffer);
00811           buffer = NULL;
00812           input->relaxation
00813             = xml_node_get_float_with_default (node,
      XML_RELAXATION,
00814                                                DEFAULT_RELAXATION, &error_code);
00815           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00816             {
00817               msg = gettext ("Invalid relaxation parameter");
00818               goto exit_on_error;
00819             }
00820         }
00821       else
00822         input->nsteps = 0;
00823     }
00824
00825   // Reading the experimental data
00826   for (child = node->children; child; child = child->next)
00827     {
00828       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00829         break;
00830 #if DEBUG
00831       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00832 #endif
00833       if (xmlHasProp (child, XML_NAME))
```

```
00834         buffer = xmlGetProp (child, XML_NAME);
00835       else
00836         {
00837           snprintf (buffer2, 64, "%s %u: %s",
00838                     gettext ("Experiment"),
00839                     input->nexperiments + 1, gettext ("no data file name"));
00840           msg = buffer2;
00841           goto exit_on_error;
00842         }
00843 #if DEBUG
00844       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00845 #endif
00846       input->weight = g_realloc (input->weight,
00847                                  (1 + input->nexperiments) * sizeof (double));
00848       input->weight[input->nexperiments]
00849         = xml_node_get_float_with_default (child,
      XML_WEIGHT, 1., &error_code);
00850       if (error_code)
00851         {
00852           snprintf (buffer2, 64, "%s %s: %s",
00853                     gettext ("Experiment"), buffer, gettext ("bad weight"));
00854           msg = buffer2;
00855           goto exit_on_error;
00856         }
00857 #if DEBUG
00858       fprintf (stderr, "input_open: weight=%lg\n",
00859                input->weight[input->nexperiments]);
00860 #endif
00861       if (!input->nexperiments)
00862         input->ninputs = 0;
00863 #if DEBUG
00864       fprintf (stderr, "input_open: template[0]\n");
00865 #endif
00866       if (xmlHasProp (child, XML_TEMPLATE1))
00867         {
00868           input->template[0]
00869             = (char **) g_realloc (input->template[0],
00870                                    (1 + input->nexperiments) * sizeof (char *));
00871           buffert[0] = (char *) xmlGetProp (child, template[0]);
00872 #if DEBUG
00873           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00874                    input->nexperiments, buffert[0]);
00875 #endif
00876           if (!input->nexperiments)
00877             ++input->ninputs;
00878 #if DEBUG
00879           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00880 #endif
00881         }
00882       else
00883         {
00884           snprintf (buffer2, 64, "%s %s: %s",
00885                     gettext ("Experiment"), buffer, gettext ("no template"));
00886           msg = buffer2;
00887           goto exit_on_error;
00888         }
00889       for (i = 1; i < MAX_NINPUTS; ++i)
00890         {
00891 #if DEBUG
00892           fprintf (stderr, "input_open: template%u\n", i + 1);
00893 #endif
00894           if (xmlHasProp (child, template[i]))
00895             {
00896               if (input->nexperiments && input->ninputs <= i)
00897                 {
00898                   snprintf (buffer2, 64, "%s %s: %s",
00899                             gettext ("Experiment"),
00900                             buffer, gettext ("bad templates number"));
00901                   msg = buffer2;
00902                   while (i-- > 0)
00903                     xmlFree (buffert[i]);
00904                   goto exit_on_error;
00905                 }
00906               input->template[i] = (char **)
00907                 g_realloc (input->template[i],
00908                            (1 + input->nexperiments) * sizeof (char *));
00909               buffert[i] = (char *) xmlGetProp (child, template[i]);
00910 #if DEBUG
00911               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00912                        input->nexperiments, i + 1,
00913                        input->template[i][input->nexperiments]);
00914 #endif
00915               if (!input->nexperiments)
00916                 ++input->ninputs;
00917 #if DEBUG
00918               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00919 #endif
```

```
00920             }
00921           else if (input->nexperiments && input->ninputs >= i)
00922             {
00923               snprintf (buffer2, 64, "%s %s: %s%u",
00924                         gettext ("Experiment"),
00925                         buffer, gettext ("no template"), i + 1);
00926               msg = buffer2;
00927               while (i-- > 0)
00928                 xmlFree (buffert[i]);
00929               goto exit_on_error;
00930             }
00931           else
00932             break;
00933         }
00934       input->experiment
00935         = g_realloc (input->experiment,
00936                      (1 + input->nexperiments) * sizeof (char *));
00937       input->experiment[input->nexperiments] = (char *) buffer;
00938       for (i = 0; i < input->ninputs; ++i)
00939         input->template[i][input->nexperiments] = buffert[i];
00940       ++input->nexperiments;
00941 #if DEBUG
00942       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00943 #endif
00944     }
00945   if (!input->nexperiments)
00946     {
00947       msg = gettext ("No calibration experiments");
00948       goto exit_on_error;
00949     }
00950   buffer = NULL;
00951
00952   // Reading the variables data
00953   for (; child; child = child->next)
00954     {
00955       if (xmlStrcmp (child->name, XML_VARIABLE))
00956         {
00957           snprintf (buffer2, 64, "%s %u: %s",
00958                     gettext ("Variable"),
00959                     input->nvariables + 1, gettext ("bad XML node"));
00960           msg = buffer2;
00961           goto exit_on_error;
00962         }
00963       if (xmlHasProp (child, XML_NAME))
00964         buffer = xmlGetProp (child, XML_NAME);
00965       else
00966         {
00967           snprintf (buffer2, 64, "%s %u: %s",
00968                     gettext ("Variable"),
00969                     input->nvariables + 1, gettext ("no name"));
00970           msg = buffer2;
00971           goto exit_on_error;
00972         }
00973       if (xmlHasProp (child, XML_MINIMUM))
00974         {
00975           input->rangemin = g_realloc
00976             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00977           input->rangeminabs = g_realloc
00978             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00979           input->rangemin[input->nvariables]
00980             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00981           if (error_code)
00982             {
00983               snprintf (buffer2, 64, "%s %s: %s",
00984                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00985               msg = buffer2;
00986               goto exit_on_error;
00987             }
00988           input->rangeminabs[input->nvariables]
00989             = xml_node_get_float_with_default (child,
    XML_ABSOLUTE_MINIMUM,
00990                                                -G_MAXDOUBLE, &error_code);
00991           if (error_code)
00992             {
00993               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
00994                         gettext ("bad absolute minimum"));
00995               msg = buffer2;
00996               goto exit_on_error;
00997             }
00998           if (input->rangemin[input->nvariables]
00999               < input->rangeminabs[input->nvariables])
01000             {
01001               snprintf (buffer2, 64, "%s %s: %s",
01002                         gettext ("Variable"),
01003                         buffer, gettext ("minimum range not allowed"));
01004               msg = buffer2;
01005               goto exit_on_error;
```

```
01006             }
01007         }
01008       else
01009         {
01010           snprintf (buffer2, 64, "%s %s: %s",
01011                     gettext ("Variable"), buffer, gettext ("no minimum range"));
01012           msg = buffer2;
01013           goto exit_on_error;
01014         }
01015       if (xmlHasProp (child, XML_MAXIMUM))
01016         {
01017           input->rangemax = g_realloc
01018             (input->rangemax, (1 + input->nvariables) * sizeof (double));
01019           input->rangemaxabs = g_realloc
01020             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
01021           input->rangemax[input->nvariables]
01022             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
01023           if (error_code)
01024             {
01025               snprintf (buffer2, 64, "%s %s: %s",
01026                         gettext ("Variable"), buffer, gettext ("bad maximum"));
01027               msg = buffer2;
01028               goto exit_on_error;
01029             }
01030           input->rangemaxabs[input->nvariables]
01031             = xml_node_get_float_with_default (child,
      XML_ABSOLUTE_MAXIMUM,
01032                                                G_MAXDOUBLE, &error_code);
01033           if (error_code)
01034             {
01035               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01036                         gettext ("bad absolute maximum"));
01037               msg = buffer2;
01038               goto exit_on_error;
01039             }
01040           if (input->rangemax[input->nvariables]
01041               > input->rangemaxabs[input->nvariables])
01042             {
01043               snprintf (buffer2, 64, "%s %s: %s",
01044                         gettext ("Variable"),
01045                         buffer, gettext ("maximum range not allowed"));
01046               msg = buffer2;
01047               goto exit_on_error;
01048             }
01049         }
01050       else
01051         {
01052           snprintf (buffer2, 64, "%s %s: %s",
01053                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01054           msg = buffer2;
01055           goto exit_on_error;
01056         }
01057       if (input->rangemax[input->nvariables]
01058           < input->rangemin[input->nvariables])
01059         {
01060           snprintf (buffer2, 64, "%s %s: %s",
01061                     gettext ("Variable"), buffer, gettext ("bad range"));
01062           msg = buffer2;
01063           goto exit_on_error;
01064         }
01065       input->precision = g_realloc
01066         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01067       input->precision[input->nvariables]
01068         = xml_node_get_uint_with_default (child,
      XML_PRECISION,
01069                                           DEFAULT_PRECISION, &error_code);
01070       if (error_code || input->precision[input->nvariables] >=
      NPRECISIONS)
01071         {
01072           snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01073                     gettext ("bad precision"));
01074           msg = buffer2;
01075           goto exit_on_error;
01076         }
01077       if (input->algorithm == ALGORITHM_SWEEP)
01078         {
01079           if (xmlHasProp (child, XML_NSWEEPS))
01080             {
01081               input->nsweeps = (unsigned int *)
01082                 g_realloc (input->nsweeps,
01083                            (1 + input->nvariables) * sizeof (unsigned int));
01084               input->nsweeps[input->nvariables]
01085                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01086               if (error_code || !input->nsweeps[input->
      nvariables])
01087                 {
01088                   snprintf (buffer2, 64, "%s %s: %s",
```

```
01089                           gettext ("Variable"),
01090                           buffer, gettext ("bad sweeps"));
01091                   msg = buffer2;
01092                   goto exit_on_error;
01093                 }
01094             }
01095           else
01096             {
01097               snprintf (buffer2, 64, "%s %s: %s", gettext ("Variable"), buffer,
01098                         gettext ("no sweeps number"));
01099               msg = buffer2;
01100               goto exit_on_error;
01101             }
01102 #if DEBUG
01103           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01104                    input->nsweeps[input->nvariables],
      input->nsimulations);
01105 #endif
01106         }
01107       if (input->algorithm == ALGORITHM_GENETIC)
01108         {
01109           // Obtaining bits representing each variable
01110           if (xmlHasProp (child, XML_NBITS))
01111             {
01112               input->nbits = (unsigned int *)
01113                 g_realloc (input->nbits,
01114                            (1 + input->nvariables) * sizeof (unsigned int));
01115               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01116               if (error_code || !i)
01117                 {
01118                   snprintf (buffer2, 64, "%s %s: %s",
01119                             gettext ("Variable"),
01120                             buffer, gettext ("invalid bits number"));
01121                   msg = buffer2;
01122                   goto exit_on_error;
01123                 }
01124               input->nbits[input->nvariables] = i;
01125             }
01126           else
01127             {
01128               snprintf (buffer2, 64, "%s %s: %s",
01129                         gettext ("Variable"),
01130                         buffer, gettext ("no bits number"));
01131               msg = buffer2;
01132               goto exit_on_error;
01133             }
01134         }
01135       else if (input->nsteps)
01136         {
01137           input->step = (double *)
01138             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01139           input->step[input->nvariables]
01140             = xml_node_get_float (child, XML_STEP, &error_code);
01141           if (error_code || input->step[input->nvariables] < 0.)
01142             {
01143               snprintf (buffer2, 64, "%s %s: %s",
01144                         gettext ("Variable"),
01145                         buffer, gettext ("bad step size"));
01146               msg = buffer2;
01147               goto exit_on_error;
01148             }
01149         }
01150       input->label = g_realloc
01151         (input->label, (1 + input->nvariables) * sizeof (char *));
01152       input->label[input->nvariables] = (char *) buffer;
01153       ++input->nvariables;
01154     }
01155   if (!input->nvariables)
01156     {
01157       msg = gettext ("No calibration variables");
01158       goto exit_on_error;
01159     }
01160   buffer = NULL;
01161
01162   // Getting the working directory
01163   input->directory = g_path_get_dirname (filename);
01164   input->name = g_path_get_basename (filename);
01165
01166   // Closing the XML document
01167   xmlFreeDoc (doc);
01168
01169 #if DEBUG
01170   fprintf (stderr, "input_open: end\n");
01171 #endif
01172   return 1;
01173
01174 exit_on_error:
```

```
01175   xmlFree (buffer);
01176   xmlFreeDoc (doc);
01177   show_error (msg);
01178   input_free ();
01179 #if DEBUG
01180   fprintf (stderr, "input_open: end\n");
01181 #endif
01182   return 0;
01183 }
```

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 257 of file mpcotool.c.

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 227 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 367 of file mpcotool.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 401 of file mpcotool.c.

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

Here is the call graph for this function:

5.7.3.16 int xml_node_get_int (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 275 of file mpcotool.c.

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

5.7.3.17 unsigned int xml_node_get_uint (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 306 of file mpcotool.c.

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

5.7.3.18 unsigned int xml_node_get_uint_with_default (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 340 of file mpcotool.c.

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

Here is the call graph for this function:

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

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

Parameters

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

Definition at line 464 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 426 of file mpcotool.c.

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

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

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

Parameters

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

Definition at line 445 of file mpcotool.c.

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

5.8 mpcotool.h

```
00001 /*
00002 MPCOTool: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier 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