Calibrator

1.2.3

Generated by Doxygen 1.8.9.1

Tue Jan 5 2016 08:42:31

Contents

Chapter 1

MPCOTool

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

VERSIONS

- 1.2.3: 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:

- 1.2.3/configure.ac: configure generator.
- 1.2.3/Makefile.in: Makefile generator.
- 1.2.3/config.h.in: config header generator.
- 1.2.3/mpcotool.c: main source code.
- 1.2.3/mpcotool.h: main header code.
- 1.2.3/interface.h: interface header code.
- 1.2.3/build: script to build all.
- 1.2.3/logo.png: logo figure.
- 1.2.3/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/1.2.3
   $ 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/1.2.3):

```
$ 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 ../../1.2.3
$ make tests
```

USER INSTRUCTIONS

- Command line in sequential mode:

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

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

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

- The syntax of the simulator has to be:

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

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

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

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

```
$ ./mpcotool
```

INPUT FILE FORMAT

The format of the main input file is as:

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

with:

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

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

Implemented algorithms are:

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

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

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

and for each variable:

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

The total number of simulations to run is:

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

SOME EXAMPLES OF INPUT FILES

Example 1

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

  $ ./pivot input_file output_file

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

  $ ./compare simulated_file data_file result_file

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

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

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

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

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

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

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

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

- A template file as *template1.js*:

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

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

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

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

`#include <mpcotool.h>`

Data Fields

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

Array of maximum variable values.

- double * rangeminabs

  Array of absolute minimum variable values.
- double * rangemaxabs

  Array of absolute maximum variable values.
- double * error_best

  Array of the best minimum errors.
- double * weight

  Array of the experiment weights.
- double * step

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

  Vector of gradient estimation.
- double * value_old

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

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

  Array of variable precisions.
- unsigned int * nsweeps

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

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

  Array of best simulation numbers.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- double calculation_time

  Calculation time.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int gradient_method

  Method to estimate the gradient.
- unsigned int nsteps

Number of steps for the gradient based method.

- unsigned int nestimates

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

  Algorithm type.
- unsigned int nstart

  Beginning simulation number of the task.
- unsigned int nend

  Ending simulation number of the task.
- unsigned int nstart_gradient

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

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 111 of file mpcotool.h.

4.1.2 Field Documentation

4.1.2.1 unsigned int* Calibrate::thread_gradient

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

Definition at line 144 of file mpcotool.h.

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

- mpcotool.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

- char * template [MAX_NINPUTS]

  Array of input template names.
- char * name

  File name.
- double weight

  Weight to calculate the objective function value.

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <mpcotool.h>`

Data Fields

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

- unsigned int * nbits

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

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- double relaxation

  Relaxation parameter.
- unsigned long int seed

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

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int ninputs

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

  Simulations number per experiment.
- unsigned int algorithm

  Algorithm type.
- unsigned int nsteps

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

  Method to estimate the gradient.
- unsigned int nestimates

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

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 64 of file mpcotool.h.

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

- mpcotool.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_seed

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

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

  Threads number GtkLabel.
- GtkSpinButton * spin_threads

  Threads number GtkSpinButton.
- GtkLabel * label_gradient

  Gradient threads number GtkLabel.
- GtkSpinButton * spin_gradient

  Gradient threads number GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 76 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <mpcotool.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 184 of file mpcotool.h.

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

- mpcotool.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkLabel * label

  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 94 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label

  Variable label.
- double rangemin

  Minimum value.
- double rangemax

  Maximum value.
- double rangeminabs

  Minimum allowed value.
- double rangemaxabs

  Maximum allowed value.
- double step

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

  Precision digits.
- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file interface.h.

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

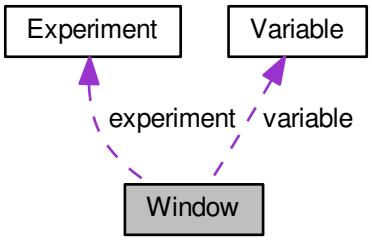
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

- GtkWindow * window
  Main GtkWindow.
- GtkGrid * grid
  Main GtkGrid.
- GtkToolbar * bar_buttons
  GtkToolbar to store the main buttons.
- GtkToolButton * button_open
  Open GtkToolButton.
- GtkToolButton * button_save
  Save GtkToolButton.
- GtkToolButton * button_run
  Run GtkToolButton.
- GtkToolButton * button_options
  Options GtkToolButton.
- GtkToolButton * button_help
  Help GtkToolButton.
- GtkToolButton * button_about
  Help GtkToolButton.
- GtkToolButton * button_exit

Exit GtkToolButton.

- GtkGrid * grid_files

  Files GtkGrid.
- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkLabel * label_result

  Result file GtkLabel.
- GtkEntry * entry_result

  Result file GtkEntry.
- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

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

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

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.

- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkCheckButton * check_gradient

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

  GtkGrid to pack the gradient based method widgets.
- GtkRadioButton * button_gradient [NGRADIENTS]

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

  GtkLabel to set the steps number.
- GtkSpinButton * spin_steps

  GtkSpinButton to set the steps number.
- GtkLabel * label_estimates

  GtkLabel to set the estimates number.
- GtkSpinButton * spin_estimates

  GtkSpinButton to set the estimates number.
- GtkLabel * label_relaxation

  GtkLabel to set the relaxation parameter.
- GtkSpinButton * spin_relaxation

  GtkSpinButton to set the relaxation parameter.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.
- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

Maximum GtkSpinButton.

- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkLabel * label_step

  GtkLabel to set the step.
- GtkSpinButton * spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow * scrolled_step

  step GtkScrolledWindow.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

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

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.

- GtkCheckButton * check_template [MAX_NINPUTS]

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

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

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.
- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

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

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

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

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 104 of file interface.h.

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

- interface.h

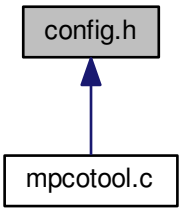
Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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



Macros

- #define MAX_NINPUTS 8

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

  Number of stochastic algorithms.
- #define NGRADIENTS 2

  Number of gradient estimate methods.
- #define NPRECISIONS 15

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

  Default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

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

  Default relaxation parameter.
- #define LOCALE_DIR "locales"

  Locales directory.

- #define PROGRAM_INTERFACE "mpcotool"

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

precision XML label.

- #define XML_RANDOM (const xmlChar*)"random"
  random XML label.
- #define XML_RELAXATION (const xmlChar*)"relaxation"
  relaxation XML label.
- #define XML_REPRODUCTION (const xmlChar*)"reproduction"
  reproduction XML label.
- #define XML_RESULT (const xmlChar*)"result"
  result XML label.
- #define XML_SIMULATOR (const xmlChar*)"simulator"
  simulator XML label.
- #define XML_SEED (const xmlChar*)"seed"
  seed XML label.
- #define XML_STEP (const xmlChar*)"step"
  step XML label.
- #define XML_SWEEP (const xmlChar*)"sweep"
  sweep XML label.
- #define XML_TEMPLATE1 (const xmlChar*)"template1"
  template1 XML label.
- #define XML_TEMPLATE2 (const xmlChar*)"template2"
  template2 XML label.
- #define XML_TEMPLATE3 (const xmlChar*)"template3"
  template3 XML label.
- #define XML_TEMPLATE4 (const xmlChar*)"template4"
  template4 XML label.
- #define XML_TEMPLATE5 (const xmlChar*)"template5"
  template5 XML label.
- #define XML_TEMPLATE6 (const xmlChar*)"template6"
  template6 XML label.
- #define XML_TEMPLATE7 (const xmlChar*)"template7"
  template7 XML label.
- #define XML_TEMPLATE8 (const xmlChar*)"template8"
  template8 XML label.
- #define XML_TOLERANCE (const xmlChar*)"tolerance"
  tolerance XML label.
- #define XML_VARIABLE (const xmlChar*)"variable"
  variable XML label.
- #define XML_VARIABLES (const xmlChar*)"variables"
  variables XML label.
- #define XML_WEIGHT (const xmlChar*)"weight"
  weight XML label.

5.1.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file config.h.

5.2 config.h

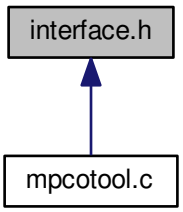
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
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

Copyright 2012-2015, all rights reserved.

Definition in file interface.h.

5.3.2 Function Documentation

5.3.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4835 of file mpcotool.c.

```
04836 {
04837 #ifdef G_OS_WIN32
04838   SYSTEM_INFO sysinfo;
04839   GetSystemInfo (&sysinfo);
04840   return sysinfo.dwNumberOfProcessors;
04841 #else
04842   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04843 #endif
04844 }
```

5.3.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

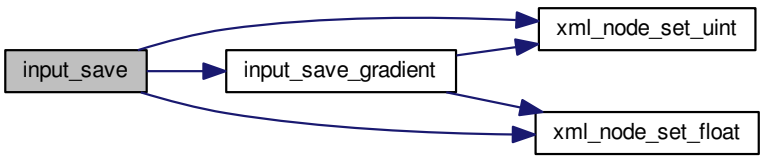
filename	Input file name.

Definition at line 2677 of file mpcotool.c.

```
02678 {
02679   unsigned int i, j;
02680   char *buffer;
02681   xmlDoc *doc;
02682   xmlNode *node, *child;
02683   GFile *file, *file2;
02684
02685 #if DEBUG
02686   fprintf (stderr, "input_save: start\n");
02687 #endif
02688
02689   // Getting the input file directory
02690   input->name = g_path_get_basename (filename);
02691   input->directory = g_path_get_dirname (filename);
02692   file = g_file_new_for_path (input->directory);
02693
02694   // Opening the input file
02695   doc = xmlNewDoc ((const xmlChar *) "1.0");
02696
02697   // Setting root XML node
02698   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02699   xmlDocSetRootElement (doc, node);
02700
02701   // Adding properties to the root XML node
02702   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02703     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02704   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02705     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02706   file2 = g_file_new_for_path (input->simulator);
02707   buffer = g_file_get_relative_path (file, file2);
02708   g_object_unref (file2);
02709   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02710   g_free (buffer);
02711   if (input->evaluator)
02712     {
02713       file2 = g_file_new_for_path (input->evaluator);
02714       buffer = g_file_get_relative_path (file, file2);
02715       g_object_unref (file2);
02716       if (xmlStrlen ((xmlChar *) buffer))
02717         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02718       g_free (buffer);
02719     }
02720   if (input->seed != DEFAULT_RANDOM_SEED)
02721     xml_node_set_uint (node, XML_SEED, input->seed);
02722
02723   // Setting the algorithm
02724   buffer = (char *) g_malloc (64);
02725   switch (input->algorithm)
02726     {
02727     case ALGORITHM_MONTE_CARLO:
02728       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02729       snprintf (buffer, 64, "%u", input->nsimulations);
02730       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02731       snprintf (buffer, 64, "%u", input->niterations);
02732       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02733       snprintf (buffer, 64, "%.3lg", input->tolerance);
02734       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02735       snprintf (buffer, 64, "%u", input->nbest);
02736       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02737       input_save_gradient (node);
02738       break;
02739     case ALGORITHM_SWEEP:
02740       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02741       snprintf (buffer, 64, "%u", input->niterations);
02742       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02743       snprintf (buffer, 64, "%.3lg", input->tolerance);
02744       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02745       snprintf (buffer, 64, "%u", input->nbest);
02746       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02747       input_save_gradient (node);
02748       break;
02749     default:
02750       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02751       snprintf (buffer, 64, "%u", input->nsimulations);
02752       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02753       snprintf (buffer, 64, "%u", input->niterations);
02754       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02755       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
```

```
02756       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02757       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02758       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02759       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02760       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02761       break;
02762     }
02763   g_free (buffer);
02764
02765   // Setting the experimental data
02766   for (i = 0; i < input->nexperiments; ++i)
02767     {
02768       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02769       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02770       if (input->weight[i] != 1.)
02771         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02772       for (j = 0; j < input->ninputs; ++j)
02773         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02774     }
02775
02776   // Setting the variables data
02777   for (i = 0; i < input->nvariables; ++i)
02778     {
02779       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02780       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02781       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02782       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02783         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02784       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02785       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02786         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02787       if (input->precision[i] != DEFAULT_PRECISION)
02788         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02789       if (input->algorithm == ALGORITHM_SWEEP)
02790         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02791       else if (input->algorithm == ALGORITHM_GENETIC)
02792         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02793       if (input->nsteps)
02794         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02795     }
02796
02797   // Saving the XML file
02798   xmlSaveFormatFile (filename, doc, 1);
02799
02800   // Freeing memory
02801   xmlFreeDoc (doc);
02802
02803 #if DEBUG
02804   fprintf (stderr, "input_save: end\n");
02805 #endif
02806 }
```

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 2910 of file mpcotool.c.

```
02911 {
02912   unsigned int i;
02913 #if DEBUG
02914   fprintf (stderr, "window_get_algorithm: start\n");
02915 #endif
02916   for (i = 0; i < NALGORITHMS; ++i)
02917     if (gtk_toggle_button_get_active
02918         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02919       break;
02920 #if DEBUG
02921   fprintf (stderr, "window_get_algorithm: %u\n", i);
02922   fprintf (stderr, "window_get_algorithm: end\n");
02923 #endif
02924   return i;
02925 }
```

5.3.2.4 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2933 of file mpcotool.c.

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

5.3.2.5 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

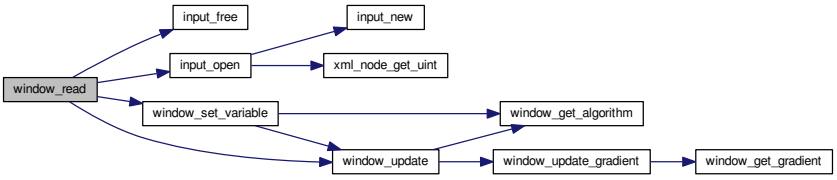
1 on succes, 0 on error.

Definition at line 4022 of file mpcotool.c.

```
04023 {
04024   unsigned int i;
04025   char *buffer;
04026 #if DEBUG
04027   fprintf (stderr, "window_read: start\n");
04028 #endif
04029
04030   // Reading new input file
04031   input_free ();
04032   if (!input_open (filename))
04033     return 0;
04034
04035   // Setting GTK+ widgets data
04036   gtk_entry_set_text (window->entry_result, input->result);
04037   gtk_entry_set_text (window->entry_variables, input->
      variables);
04038   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04039   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04040                                  (window->button_simulator), buffer);
04041   g_free (buffer);
04042   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04043                                 (size_t) input->evaluator);
04044   if (input->evaluator)
04045     {
04046       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04047       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04048                                      (window->button_evaluator), buffer);
04049       g_free (buffer);
04050     }
04051   gtk_toggle_button_set_active
04052     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04053   switch (input->algorithm)
04054     {
04055     case ALGORITHM_MONTE_CARLO:
04056       gtk_spin_button_set_value (window->spin_simulations,
04057                                  (gdouble) input->nsimulations);
04058     case ALGORITHM_SWEEP:
04059       gtk_spin_button_set_value (window->spin_iterations,
04060                                  (gdouble) input->niterations);
04061       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04062       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04063       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04064                                     input->nsteps);
04065       if (input->nsteps)
04066         {
04067           gtk_toggle_button_set_active
04068             (GTK_TOGGLE_BUTTON (window->button_gradient
04069                                 [input->gradient_method]), TRUE);
04070           gtk_spin_button_set_value (window->spin_steps,
04071                                      (gdouble) input->nsteps);
04072           gtk_spin_button_set_value (window->spin_relaxation,
04073                                      (gdouble) input->relaxation);
04074           switch (input->gradient_method)
04075             {
04076             case GRADIENT_METHOD_RANDOM:
04077               gtk_spin_button_set_value (window->spin_estimates,
04078                                          (gdouble) input->nestimates);
04079             }
04080         }
04081       break;
04082     default:
04083       gtk_spin_button_set_value (window->spin_population,
04084                                  (gdouble) input->nsimulations);
04085       gtk_spin_button_set_value (window->spin_generations,
04086                                  (gdouble) input->niterations);
04087       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04088       gtk_spin_button_set_value (window->spin_reproduction,
04089                                  input->reproduction_ratio);
04090       gtk_spin_button_set_value (window->spin_adaptation,
04091                                  input->adaptation_ratio);
04092     }
04093   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04094   g_signal_handler_block (window->button_experiment,
04095                           window->id_experiment_name);
04096   gtk_combo_box_text_remove_all (window->combo_experiment);
04097   for (i = 0; i < input->nexperiments; ++i)
04098     gtk_combo_box_text_append_text (window->combo_experiment,
04099                                     input->experiment[i]);
04100   g_signal_handler_unblock
```

```
04101     (window->button_experiment, window->
      id_experiment_name);
04102   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04103   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04104   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04105   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04106   gtk_combo_box_text_remove_all (window->combo_variable);
04107   for (i = 0; i < input->nvariables; ++i)
04108     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
04109   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04110   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04111   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04112   window_set_variable ();
04113   window_update ();
04114
04115 #if DEBUG
04116   fprintf (stderr, "window_read: end\n");
04117 #endif
04118   return 1;
04119 }
```

Here is the call graph for this function:



5.3.2.6 int window_save ( )

Function to save the input file.

Returns

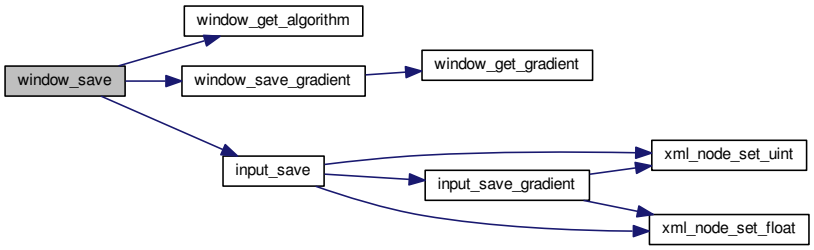
1 on OK, 0 on Cancel.

Definition at line 2988 of file mpcotool.c.

```
02989 {
02990   char *buffer;
02991   GtkFileChooserDialog *dlg;
02992
02993 #if DEBUG
02994   fprintf (stderr, "window_save: start\n");
02995 #endif
02996
02997   // Opening the saving dialog
02998   dlg = (GtkFileChooserDialog *)
02999     gtk_file_chooser_dialog_new (gettext ("Save file"),
03000                                  window->window,
03001                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03002                                  gettext ("_Cancel"),
03003                                  GTK_RESPONSE_CANCEL,
03004                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03005   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03006   buffer = g_build_filename (input->directory, input->name, NULL);
03007   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03008   g_free (buffer);
03009
03010   // If OK response then saving
03011   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
```

```
03012     {
03013
03014       // Adding properties to the root XML node
03015       input->simulator = gtk_file_chooser_get_filename
03016         (GTK_FILE_CHOOSER (window->button_simulator));
03017       if (gtk_toggle_button_get_active
03018           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03019         input->evaluator = gtk_file_chooser_get_filename
03020           (GTK_FILE_CHOOSER (window->button_evaluator));
03021       else
03022         input->evaluator = NULL;
03023       input->result
03024         = (char *) xmlStrdup ((const xmlChar *)
03025                               gtk_entry_get_text (window->entry_result));
03026       input->variables
03027         = (char *) xmlStrdup ((const xmlChar *)
03028                               gtk_entry_get_text (window->entry_variables));
03029
03030       // Setting the algorithm
03031       switch (window_get_algorithm ())
03032         {
03033         case ALGORITHM_MONTE_CARLO:
03034           input->algorithm = ALGORITHM_MONTE_CARLO;
03035           input->nsimulations
03036             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03037           input->niterations
03038             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03039           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03040           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03041           window_save_gradient ();
03042           break;
03043         case ALGORITHM_SWEEP:
03044           input->algorithm = ALGORITHM_SWEEP;
03045           input->niterations
03046             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03047           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03048           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03049           window_save_gradient ();
03050           break;
03051         default:
03052           input->algorithm = ALGORITHM_GENETIC;
03053           input->nsimulations
03054             = gtk_spin_button_get_value_as_int (window->spin_population);
03055           input->niterations
03056             = gtk_spin_button_get_value_as_int (window->spin_generations);
03057           input->mutation_ratio
03058             = gtk_spin_button_get_value (window->spin_mutation);
03059           input->reproduction_ratio
03060             = gtk_spin_button_get_value (window->spin_reproduction);
03061           input->adaptation_ratio
03062             = gtk_spin_button_get_value (window->spin_adaptation);
03063           break;
03064         }
03065
03066       // Saving the XML file
03067       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03068       input_save (buffer);
03069
03070       // Closing and freeing memory
03071       g_free (buffer);
03072       gtk_widget_destroy (GTK_WIDGET (dlg));
03073 #if DEBUG
03074       fprintf (stderr, "window_save: end\n");
03075 #endif
03076       return 1;
03077     }
03078
03079   // Closing and freeing memory
03080   gtk_widget_destroy (GTK_WIDGET (dlg));
03081 #if DEBUG
03082   fprintf (stderr, "window_save: end\n");
03083 #endif
03084   return 0;
03085 }
```

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 3626 of file mpcotool.c.

```
03627 {
03628   unsigned int i, j;
03629   char *buffer;
03630   GFile *file1, *file2;
03631 #if DEBUG
03632   fprintf (stderr, "window_template_experiment: start\n");
03633 #endif
03634   i = (size_t) data;
03635   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03636   file1
03637     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03638   file2 = g_file_new_for_path (input->directory);
03639   buffer = g_file_get_relative_path (file2, file1);
03640   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03641   g_free (buffer);
03642   g_object_unref (file2);
03643   g_object_unref (file1);
03644 #if DEBUG
03645   fprintf (stderr, "window_template_experiment: end\n");
03646 #endif
03647 }
```

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 ();
```

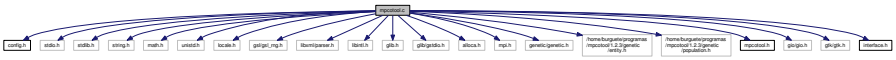
```
00247 void window_inputs_experiment ();
00248 void window_template_experiment (void *data);
00249 void window_set_variable ();
00250 void window_remove_variable ();
00251 void window_add_variable ();
00252 void window_label_variable ();
00253 void window_precision_variable ();
00254 void window_rangemin_variable ();
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.
- 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.
- 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 1421 of file mpcotool.c.

```
01422 {
01423   unsigned int i, j;
01424   double e;
01425 #if DEBUG
01426   fprintf (stderr, "calibrate_best: start\n");
01427   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01428            calibrate->nsaveds, calibrate->nbest);
01429 #endif
01430   if (calibrate->nsaveds < calibrate->nbest
01431       || value < calibrate->error_best[calibrate->nsaveds - 1])
01432     {
01433       if (calibrate->nsaveds < calibrate->nbest)
01434         ++calibrate->nsaveds;
01435       calibrate->error_best[calibrate->nsaveds - 1] = value;
01436       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01437       for (i = calibrate->nsaveds; --i;)
01438         {
01439           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01440             {
01441               j = calibrate->simulation_best[i];
01442               e = calibrate->error_best[i];
01443               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01444               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01445               calibrate->simulation_best[i - 1] = j;
01446               calibrate->error_best[i - 1] = e;
01447             }
01448           else
01449             break;
01450         }
01451     }
01452 #if DEBUG
01453   fprintf (stderr, "calibrate_best: end\n");
01454 #endif
01455 }
```

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 1734 of file mpcotool.c.

```
01735 {
01736 #if DEBUG
01737   fprintf (stderr, "calibrate_best_gradient: start\n");
01738   fprintf (stderr,
01739            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01740            simulation, value, calibrate->error_best[0]);
01741 #endif
01742   if (value < calibrate->error_best[0])
01743     {
01744       calibrate->error_best[0] = value;
01745       calibrate->simulation_best[0] = simulation;
01746 #if DEBUG
01747       fprintf (stderr,
01748                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01749                simulation, value);
01750 #endif
01751     }
01752 #if DEBUG
01753   fprintf (stderr, "calibrate_best_gradient: end\n");
01754 #endif
01755 }
```

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 1871 of file mpcotool.c.

```
01873 {
01874   double x;
01875 #if DEBUG
01876   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01877 #endif
01878   x = calibrate->gradient[variable];
01879   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01880     {
01881       if (estimate & 1)
01882         x += calibrate->step[variable];
01883       else
01884         x -= calibrate->step[variable];
01885     }
01886 #if DEBUG
01887   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01888            variable, x);
01889   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01890 #endif
01891   return x;
01892 }
```

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 1844 of file mpcotool.c.

```
01846 {
01847   double x;
01848 #if DEBUG
01849   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01850 #endif
01851   x = calibrate->gradient[variable]
01852     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
      step[variable];
01853 #if DEBUG
01854   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01855            variable, x);
01856   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01857 #endif
01858   return x;
01859 }
```

5.5.2.5 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

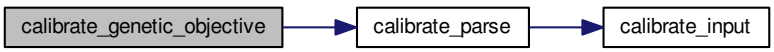
Returns

objective function value.

Definition at line 2037 of file mpcotool.c.

```
02038 {
02039   unsigned int j;
02040   double objective;
02041   char buffer[64];
02042 #if DEBUG
02043   fprintf (stderr, "calibrate_genetic_objective: start\n");
02044 #endif
02045   for (j = 0; j < calibrate->nvariables; ++j)
02046     {
02047       calibrate->value[entity->id * calibrate->nvariables + j]
02048         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02049     }
02050   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02051     objective += calibrate_parse (entity->id, j);
02052   g_mutex_lock (mutex);
02053   for (j = 0; j < calibrate->nvariables; ++j)
02054     {
02055       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02056       fprintf (calibrate->file_variables, buffer,
02057                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
02058     }
02059   fprintf (calibrate->file_variables, "%.14le\n", objective);
02060   g_mutex_unlock (mutex);
02061 #if DEBUG
02062   fprintf (stderr, "calibrate_genetic_objective: end\n");
02063 #endif
02064   return objective;
02065 }
```

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

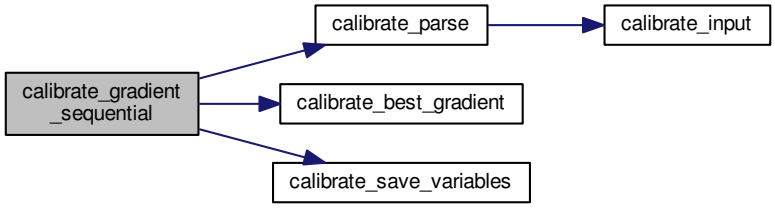
simulation	Simulation number.

Definition at line 1764 of file mpcotool.c.

```
01765 {
01766   unsigned int i, j, k;
01767   double e;
01768 #if DEBUG
01769   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01770   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01771            "nend_gradient=%u\n",
01772            calibrate->nstart_gradient, calibrate->
      nend_gradient);
01773 #endif
01774   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01775     {
```

```
01776       k = simulation + i;
01777       e = 0.;
01778       for (j = 0; j < calibrate->nexperiments; ++j)
01779         e += calibrate_parse (k, j);
01780       calibrate_best_gradient (k, e);
01781       calibrate_save_variables (k, e);
01782 #if DEBUG
01783       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01784 #endif
01785     }
01786 #if DEBUG
01787   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01788 #endif
01789 }
```

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

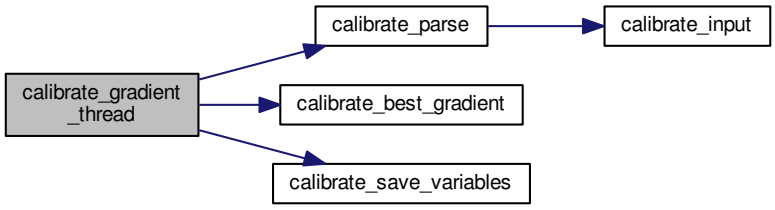
NULL

Definition at line 1799 of file mpcotool.c.

```
01800 {
01801   unsigned int i, j, thread;
01802   double e;
01803 #if DEBUG
01804   fprintf (stderr, "calibrate_gradient_thread: start\n");
01805 #endif
01806   thread = data->thread;
01807 #if DEBUG
01808   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01809            thread,
01810            calibrate->thread_gradient[thread],
01811            calibrate->thread_gradient[thread + 1]);
01812 #endif
01813   for (i = calibrate->thread_gradient[thread];
01814        i < calibrate->thread_gradient[thread + 1]; ++i)
01815     {
01816       e = 0.;
01817       for (j = 0; j < calibrate->nexperiments; ++j)
01818         e += calibrate_parse (i, j);
01819       g_mutex_lock (mutex);
01820       calibrate_best_gradient (i, e);
01821       calibrate_save_variables (i, e);
01822       g_mutex_unlock (mutex);
01823 #if DEBUG
01824       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
```

```
01825 #endif
01826     }
01827 #if DEBUG
01828   fprintf (stderr, "calibrate_gradient_thread: end\n");
01829 #endif
01830   g_thread_exit (NULL);
01831   return NULL;
01832 }
```

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 1174 of file mpcotool.c.

```
01175 {
01176   unsigned int i;
01177   char buffer[32], value[32], *buffer2, *buffer3, *content;
01178   FILE *file;
01179   gsize length;
01180   GRegex *regex;
01181
01182 #if DEBUG
01183   fprintf (stderr, "calibrate_input: start\n");
01184 #endif
01185
01186   // Checking the file
01187   if (!template)
01188     goto calibrate_input_end;
01189
01190   // Opening template
01191   content = g_mapped_file_get_contents (template);
01192   length = g_mapped_file_get_length (template);
01193 #if DEBUG
01194   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01195            content);
01196 #endif
01197   file = g_fopen (input, "w");
01198
01199   // Parsing template
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
01202 #if DEBUG
01203       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01204 #endif
01205       snprintf (buffer, 32, "@variable%u@", i + 1);
01206       regex = g_regex_new (buffer, 0, 0, NULL);
01207       if (i == 0)
01208         {
01209           buffer2 = g_regex_replace_literal (regex, content, length, 0,
```

```
01210                                           calibrate->label[i], 0, NULL);
01211 #if DEBUG
01212           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01213 #endif
01214         }
01215       else
01216         {
01217           length = strlen (buffer3);
01218           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01219                                              calibrate->label[i], 0, NULL);
01220           g_free (buffer3);
01221         }
01222       g_regex_unref (regex);
01223       length = strlen (buffer2);
01224       snprintf (buffer, 32, "@value%u@", i + 1);
01225       regex = g_regex_new (buffer, 0, 0, NULL);
01226       snprintf (value, 32, format[calibrate->precision[i]],
01227                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01228
01229 #if DEBUG
01230       fprintf (stderr, "calibrate_input: value=%s\n", value);
01231 #endif
01232       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01233                                          0, NULL);
01234       g_free (buffer2);
01235       g_regex_unref (regex);
01236     }
01237
01238   // Saving input file
01239   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01240   g_free (buffer3);
01241   fclose (file);
01242
01243 calibrate_input_end:
01244 #if DEBUG
01245   fprintf (stderr, "calibrate_input: end\n");
01246 #endif
01247   return;
01248 }
```

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 1539 of file mpcotool.c.

```
01541 {
01542   unsigned int i, j, k, s[calibrate->nbest];
01543   double e[calibrate->nbest];
01544 #if DEBUG
01545   fprintf (stderr, "calibrate_merge: start\n");
01546 #endif
01547   i = j = k = 0;
01548   do
01549     {
01550       if (i == calibrate->nsaveds)
01551         {
01552           s[k] = simulation_best[j];
01553           e[k] = error_best[j];
01554           ++j;
01555           ++k;
01556           if (j == nsaveds)
01557             break;
01558         }
01559       else if (j == nsaveds)
01560         {
01561           s[k] = calibrate->simulation_best[i];
01562           e[k] = calibrate->error_best[i];
01563           ++i;
01564           ++k;
01565           if (i == calibrate->nsaveds)
01566             break;
01567         }
```

```
01568       else if (calibrate->error_best[i] > error_best[j])
01569         {
01570           s[k] = simulation_best[j];
01571           e[k] = error_best[j];
01572           ++j;
01573           ++k;
01574         }
01575       else
01576         {
01577           s[k] = calibrate->simulation_best[i];
01578           e[k] = calibrate->error_best[i];
01579           ++i;
01580           ++k;
01581         }
01582     }
01583   while (k < calibrate->nbest);
01584   calibrate->nsaveds = k;
01585   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01586   memcpy (calibrate->error_best, e, k * sizeof (double));
01587 #if DEBUG
01588   fprintf (stderr, "calibrate_merge: end\n");
01589 #endif
01590 }
```

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 1261 of file mpcotool.c.

```
01262 {
01263   unsigned int i;
01264   double e;
01265   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01266     *buffer3, *buffer4;
01267   FILE *file_result;
01268
01269 #if DEBUG
01270   fprintf (stderr, "calibrate_parse: start\n");
01271   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01272            experiment);
01273 #endif
01274
01275   // Opening input files
01276   for (i = 0; i < calibrate->ninputs; ++i)
01277     {
01278       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01279 #if DEBUG
01280       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01281 #endif
01282       calibrate_input (simulation, &input[i][0],
01283                        calibrate->file[i][experiment]);
01284     }
01285   for (; i < MAX_NINPUTS; ++i)
01286     strcpy (&input[i][0], "");
01287 #if DEBUG
01288   fprintf (stderr, "calibrate_parse: parsing end\n");
01289 #endif
01290
01291   // Performing the simulation
01292   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01293   buffer2 = g_path_get_dirname (calibrate->simulator);
01294   buffer3 = g_path_get_basename (calibrate->simulator);
01295   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01296   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01297             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01298             input[6], input[7], output);
01299   g_free (buffer4);
01300   g_free (buffer3);
```

```
01301   g_free (buffer2);
01302 #if DEBUG
01303   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01304 #endif
01305   system (buffer);
01306
01307   // Checking the objective value function
01308   if (calibrate->evaluator)
01309     {
01310       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01311       buffer2 = g_path_get_dirname (calibrate->evaluator);
01312       buffer3 = g_path_get_basename (calibrate->evaluator);
01313       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01314       snprintf (buffer, 512, "\"%s\" %s %s %s",
01315                 buffer4, output, calibrate->experiment[experiment], result);
01316       g_free (buffer4);
01317       g_free (buffer3);
01318       g_free (buffer2);
01319 #if DEBUG
01320       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01321 #endif
01322       system (buffer);
01323       file_result = g_fopen (result, "r");
01324       e = atof (fgets (buffer, 512, file_result));
01325       fclose (file_result);
01326     }
01327   else
01328     {
01329       strcpy (result, "");
01330       file_result = g_fopen (output, "r");
01331       e = atof (fgets (buffer, 512, file_result));
01332       fclose (file_result);
01333     }
01334
01335   // Removing files
01336 #if !DEBUG
01337   for (i = 0; i < calibrate->ninputs; ++i)
01338     {
01339       if (calibrate->file[i][0])
01340         {
01341           snprintf (buffer, 512, RM " %s", &input[i][0]);
01342           system (buffer);
01343         }
01344     }
01345   snprintf (buffer, 512, RM " %s %s", output, result);
01346   system (buffer);
01347 #endif
01348
01349 #if DEBUG
01350   fprintf (stderr, "calibrate_parse: end\n");
01351 #endif
01352
01353   // Returning the objective function
01354   return e * calibrate->weight[experiment];
01355 }
```

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 1393 of file mpcotool.c.

```
01394 {
01395   unsigned int i;
01396   char buffer[64];
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_save_variables: start\n");
01399 #endif
01400   for (i = 0; i < calibrate->nvariables; ++i)
01401     {
01402       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01403       fprintf (calibrate->file_variables, buffer,
01404                calibrate->value[simulation * calibrate->
      nvariables + i]);
01405     }
01406   fprintf (calibrate->file_variables, "%.14le\n", error);
01407 #if DEBUG
01408   fprintf (stderr, "calibrate_save_variables: end\n");
01409 #endif
01410 }
```

5.5.2.12 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

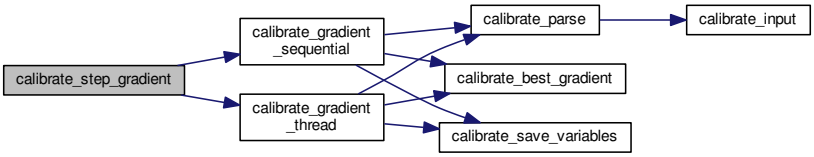
simulation	Simulation number.

Definition at line 1901 of file mpcotool.c.

```
01902 {
01903   GThread *thread[nthreads_gradient];
01904   ParallelData data[nthreads_gradient];
01905   unsigned int i, j, k, b;
01906 #if DEBUG
01907   fprintf (stderr, "calibrate_step_gradient: start\n");
01908 #endif
01909   for (i = 0; i < calibrate->nestimates; ++i)
01910     {
01911       k = (simulation + i) * calibrate->nvariables;
01912       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01913 #if DEBUG
01914       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01915                simulation + i, calibrate->simulation_best[0]);
01916 #endif
01917       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01918         {
01919 #if DEBUG
01920           fprintf (stderr,
01921                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01922                    i, j, calibrate->value[b]);
01923 #endif
01924           calibrate->value[k]
01925             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01926           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01927                                             calibrate->rangeminabs[j]),
01928                                       calibrate->rangemaxabs[j]);
01929 #if DEBUG
01930           fprintf (stderr,
01931                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01932                    i, j, calibrate->value[k]);
01933 #endif
01934         }
01935     }
01936   if (nthreads_gradient == 1)
01937     calibrate_gradient_sequential (simulation);
01938   else
01939     {
01940       for (i = 0; i <= nthreads_gradient; ++i)
01941         {
```

```
01942           calibrate->thread_gradient[i]
01943             = simulation + calibrate->nstart_gradient
01944             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01945               / nthreads_gradient;
01946 #if DEBUG
01947           fprintf (stderr,
01948                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01949                    i, calibrate->thread_gradient[i]);
01950 #endif
01951         }
01952       for (i = 0; i < nthreads_gradient; ++i)
01953         {
01954           data[i].thread = i;
01955           thread[i] = g_thread_new
01956             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01957         }
01958       for (i = 0; i < nthreads_gradient; ++i)
01959         g_thread_join (thread[i]);
01960     }
01961 #if DEBUG
01962   fprintf (stderr, "calibrate_step_gradient: end\n");
01963 #endif
01964 }
```

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

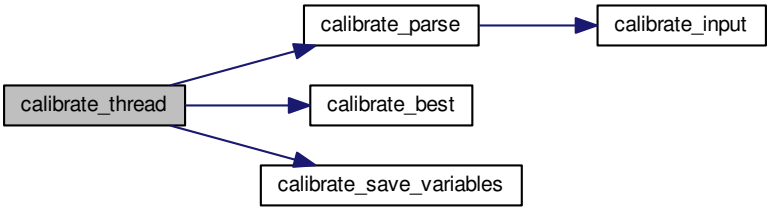
NULL

Definition at line 1495 of file mpcotool.c.

```
01496 {
01497   unsigned int i, j, thread;
01498   double e;
01499 #if DEBUG
01500   fprintf (stderr, "calibrate_thread: start\n");
01501 #endif
01502   thread = data->thread;
01503 #if DEBUG
01504   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01505            calibrate->thread[thread], calibrate->thread[thread + 1]);
01506 #endif
01507   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01508     {
01509       e = 0.;
01510       for (j = 0; j < calibrate->nexperiments; ++j)
01511         e += calibrate_parse (i, j);
01512       g_mutex_lock (mutex);
01513       calibrate_best (i, e);
01514       calibrate_save_variables (i, e);
01515       g_mutex_unlock (mutex);
```

```
01516 #if DEBUG
01517       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01518 #endif
01519     }
01520 #if DEBUG
01521   fprintf (stderr, "calibrate_thread: end\n");
01522 #endif
01523   g_thread_exit (NULL);
01524   return NULL;
01525 }
```

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 4835 of file mpcotool.c.

```
04836 {
04837 #ifdef G_OS_WIN32
04838   SYSTEM_INFO sysinfo;
04839   GetSystemInfo (&sysinfo);
04840   return sysinfo.dwNumberOfProcessors;
04841 #else
04842   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04843 #endif
04844 }
```

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 489 of file mpcotool.c.

```
00490 {
00491   char buffer2[64];
00492   char *buffert[MAX_NINPUTS] =
00493     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00494   xmlDoc *doc;
00495   xmlNode *node, *child;
00496   xmlChar *buffer;
00497   char *msg;
00498   int error_code;
00499   unsigned int i;
00500
00501 #if DEBUG
00502   fprintf (stderr, "input_open: start\n");
00503 #endif
00504
00505   // Resetting input data
00506   buffer = NULL;
00507   input_new ();
00508
00509   // Parsing the input file
00510 #if DEBUG
00511   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00512 #endif
00513   doc = xmlParseFile (filename);
00514   if (!doc)
00515     {
00516       msg = gettext ("Unable to parse the input file");
00517       goto exit_on_error;
00518     }
00519
00520   // Getting the root node
00521 #if DEBUG
00522   fprintf (stderr, "input_open: getting the root node\n");
00523 #endif
00524   node = xmlDocGetRootElement (doc);
00525   if (xmlStrcmp (node->name, XML_CALIBRATE))
00526     {
00527       msg = gettext ("Bad root XML node");
00528       goto exit_on_error;
00529     }
00530
00531   // Getting results file names
00532   input->result = (char *) xmlGetProp (node, XML_RESULT);
00533   if (!input->result)
00534     input->result = (char *) xmlStrdup (result_name);
00535   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00536   if (!input->variables)
00537     input->variables = (char *) xmlStrdup (variables_name);
00538
00539   // Opening simulator program name
00540   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00541   if (!input->simulator)
00542     {
00543       msg = gettext ("Bad simulator program");
00544       goto exit_on_error;
00545     }
00546
00547   // Opening evaluator program name
00548   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00549
00550   // Obtaining pseudo-random numbers generator seed
00551   if (!xmlHasProp (node, XML_SEED))
00552     input->seed = DEFAULT_RANDOM_SEED;
00553   else
00554     {
00555       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00556       if (error_code)
00557         {
00558           msg = gettext ("Bad pseudo-random numbers generator seed");
00559           goto exit_on_error;
00560         }
00561     }
00562
00563   // Opening algorithm
00564   buffer = xmlGetProp (node, XML_ALGORITHM);
00565   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00566     {
00567       input->algorithm = ALGORITHM_MONTE_CARLO;
00568
00569       // Obtaining simulations number
00570       input->nsimulations
00571         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00572       if (error_code)
00573         {
00574           msg = gettext ("Bad simulations number");
00575           goto exit_on_error;
00576         }
```

```
00577     }
00578   else if (!xmlStrcmp (buffer, XML_SWEEP))
00579     input->algorithm = ALGORITHM_SWEEP;
00580   else if (!xmlStrcmp (buffer, XML_GENETIC))
00581     {
00582       input->algorithm = ALGORITHM_GENETIC;
00583
00584       // Obtaining population
00585       if (xmlHasProp (node, XML_NPOPULATION))
00586         {
00587           input->nsimulations
00588             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00589           if (error_code || input->nsimulations < 3)
00590             {
00591               msg = gettext ("Invalid population number");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No population number");
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining generations
00602       if (xmlHasProp (node, XML_NGENERATIONS))
00603         {
00604           input->niterations
00605             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00606           if (error_code || !input->niterations)
00607             {
00608               msg = gettext ("Invalid generations number");
00609               goto exit_on_error;
00610             }
00611         }
00612       else
00613         {
00614           msg = gettext ("No generations number");
00615           goto exit_on_error;
00616         }
00617
00618       // Obtaining mutation probability
00619       if (xmlHasProp (node, XML_MUTATION))
00620         {
00621           input->mutation_ratio
00622             = xml_node_get_float (node, XML_MUTATION, &error_code);
00623           if (error_code || input->mutation_ratio < 0.
00624               || input->mutation_ratio >= 1.)
00625             {
00626               msg = gettext ("Invalid mutation probability");
00627               goto exit_on_error;
00628             }
00629         }
00630       else
00631         {
00632           msg = gettext ("No mutation probability");
00633           goto exit_on_error;
00634         }
00635
00636       // Obtaining reproduction probability
00637       if (xmlHasProp (node, XML_REPRODUCTION))
00638         {
00639           input->reproduction_ratio
00640             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00641           if (error_code || input->reproduction_ratio < 0.
00642               || input->reproduction_ratio >= 1.0)
00643             {
00644               msg = gettext ("Invalid reproduction probability");
00645               goto exit_on_error;
00646             }
00647         }
00648       else
00649         {
00650           msg = gettext ("No reproduction probability");
00651           goto exit_on_error;
00652         }
00653
00654       // Obtaining adaptation probability
00655       if (xmlHasProp (node, XML_ADAPTATION))
00656         {
00657           input->adaptation_ratio
00658             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00659           if (error_code || input->adaptation_ratio < 0.
00660               || input->adaptation_ratio >= 1.)
00661             {
00662               msg = gettext ("Invalid adaptation probability");
00663               goto exit_on_error;
```

```
00664             }
00665         }
00666       else
00667         {
00668           msg = gettext ("No adaptation probability");
00669           goto exit_on_error;
00670         }
00671
00672       // Checking survivals
00673       i = input->mutation_ratio * input->nsimulations;
00674       i += input->reproduction_ratio * input->
      nsimulations;
00675       i += input->adaptation_ratio * input->
      nsimulations;
00676       if (i > input->nsimulations - 2)
00677         {
00678           msg = gettext
00679             ("No enough survival entities to reproduce the population");
00680           goto exit_on_error;
00681         }
00682     }
00683   else
00684     {
00685       msg = gettext ("Unknown algorithm");
00686       goto exit_on_error;
00687     }
00688   xmlFree (buffer);
00689   buffer = NULL;
00690
00691   if (input->algorithm == ALGORITHM_MONTE_CARLO
00692       || input->algorithm == ALGORITHM_SWEEP)
00693     {
00694
00695       // Obtaining iterations number
00696       input->niterations
00697         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00698       if (error_code == 1)
00699         input->niterations = 1;
00700       else if (error_code)
00701         {
00702           msg = gettext ("Bad iterations number");
00703           goto exit_on_error;
00704         }
00705
00706       // Obtaining best number
00707       if (xmlHasProp (node, XML_NBEST))
00708         {
00709           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00710           if (error_code || !input->nbest)
00711             {
00712               msg = gettext ("Invalid best number");
00713               goto exit_on_error;
00714             }
00715         }
00716       else
00717         input->nbest = 1;
00718
00719       // Obtaining tolerance
00720       if (xmlHasProp (node, XML_TOLERANCE))
00721         {
00722           input->tolerance
00723             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00724           if (error_code || input->tolerance < 0.)
00725             {
00726               msg = gettext ("Invalid tolerance");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         input->tolerance = 0.;
00732
00733       // Getting gradient method parameters
00734       if (xmlHasProp (node, XML_NSTEPS))
00735         {
00736           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00737           if (error_code || !input->nsteps)
00738             {
00739               msg = gettext ("Invalid steps number");
00740               goto exit_on_error;
00741             }
00742           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00743           if (!xmlStrcmp (buffer, XML_COORDINATES))
00744             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00745           else if (!xmlStrcmp (buffer, XML_RANDOM))
```

```
00746             {
00747               input->gradient_method =
    GRADIENT_METHOD_RANDOM;
00748               input->nestimates
00749                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00750               if (error_code || !input->nestimates)
00751                 {
00752                   msg = gettext ("Invalid estimates number");
00753                   goto exit_on_error;
00754                 }
00755             }
00756           else
00757             {
00758               msg = gettext ("Unknown method to estimate the gradient");
00759               goto exit_on_error;
00760             }
00761           xmlFree (buffer);
00762           buffer = NULL;
00763           if (xmlHasProp (node, XML_RELAXATION))
00764             {
00765               input->relaxation
00766                 = xml_node_get_float (node, XML_RELAXATION, &error_code);
00767               if (error_code || input->relaxation < 0.
00768                   || input->relaxation > 2.)
00769                 {
00770                   msg = gettext ("Invalid relaxation parameter");
00771                   goto exit_on_error;
00772                 }
00773             }
00774           else
00775             input->relaxation = DEFAULT_RELAXATION;
00776         }
00777       else
00778         input->nsteps = 0;
00779     }
00780
00781   // Reading the experimental data
00782   for (child = node->children; child; child = child->next)
00783     {
00784       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00785         break;
00786 #if DEBUG
00787       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00788 #endif
00789       if (xmlHasProp (child, XML_NAME))
00790         buffer = xmlGetProp (child, XML_NAME);
00791       else
00792         {
00793           snprintf (buffer2, 64, "%s %u: %s",
00794                     gettext ("Experiment"),
00795                     input->nexperiments + 1, gettext ("no data file name"));
00796           msg = buffer2;
00797           goto exit_on_error;
00798         }
00799 #if DEBUG
00800       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00801 #endif
00802       input->weight = g_realloc (input->weight,
00803                                  (1 + input->nexperiments) * sizeof (double));
00804       if (xmlHasProp (child, XML_WEIGHT))
00805         {
00806           input->weight[input->nexperiments]
00807             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00808           if (error_code)
00809             {
00810               snprintf (buffer2, 64, "%s %s: %s",
00811                         gettext ("Experiment"), buffer, gettext ("bad weight"));
00812               msg = buffer2;
00813               goto exit_on_error;
00814             }
00815         }
00816       else
00817         input->weight[input->nexperiments] = 1.;
00818 #if DEBUG
00819       fprintf (stderr, "input_open: weight=%lg\n",
00820                input->weight[input->nexperiments]);
00821 #endif
00822       if (!input->nexperiments)
00823         input->ninputs = 0;
00824 #if DEBUG
00825       fprintf (stderr, "input_open: template[0]\n");
00826 #endif
00827       if (xmlHasProp (child, XML_TEMPLATE1))
00828         {
00829           input->template[0]
00830             = (char **) g_realloc (input->template[0],
00831                                    (1 + input->nexperiments) * sizeof (char *));
```

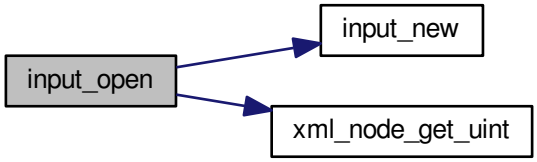
```
00832           buffert[0] = (char *) xmlGetProp (child, template[0]);
00833 #if DEBUG
00834           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00835                    input->nexperiments, buffert[0]);
00836 #endif
00837           if (!input->nexperiments)
00838             ++input->ninputs;
00839 #if DEBUG
00840           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00841 #endif
00842         }
00843       else
00844         {
00845           snprintf (buffer2, 64, "%s %s: %s",
00846                     gettext ("Experiment"), buffer, gettext ("no template"));
00847           msg = buffer2;
00848           goto exit_on_error;
00849         }
00850       for (i = 1; i < MAX_NINPUTS; ++i)
00851         {
00852 #if DEBUG
00853           fprintf (stderr, "input_open: template%u\n", i + 1);
00854 #endif
00855           if (xmlHasProp (child, template[i]))
00856             {
00857               if (input->nexperiments && input->ninputs <= i)
00858                 {
00859                   snprintf (buffer2, 64, "%s %s: %s",
00860                             gettext ("Experiment"),
00861                             buffer, gettext ("bad templates number"));
00862                   msg = buffer2;
00863                   while (i-- > 0)
00864                     xmlFree (buffert[i]);
00865                   goto exit_on_error;
00866                 }
00867               input->template[i] = (char **)
00868                 g_realloc (input->template[i],
00869                            (1 + input->nexperiments) * sizeof (char *));
00870               buffert[i] = (char *) xmlGetProp (child, template[i]);
00871 #if DEBUG
00872               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00873                        input->nexperiments, i + 1,
00874                        input->template[i][input->nexperiments]);
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 if (input->nexperiments && input->ninputs >= i)
00883             {
00884               snprintf (buffer2, 64, "%s %s: %s%u",
00885                         gettext ("Experiment"),
00886                         buffer, gettext ("no template"), i + 1);
00887               msg = buffer2;
00888               while (i-- > 0)
00889                 xmlFree (buffert[i]);
00890               goto exit_on_error;
00891             }
00892           else
00893             break;
00894         }
00895       input->experiment
00896         = g_realloc (input->experiment,
00897                      (1 + input->nexperiments) * sizeof (char *));
00898       input->experiment[input->nexperiments] = (char *) buffer;
00899       for (i = 0; i < input->ninputs; ++i)
00900         input->template[i][input->nexperiments] = buffert[i];
00901       ++input->nexperiments;
00902 #if DEBUG
00903       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00904 #endif
00905     }
00906   if (!input->nexperiments)
00907     {
00908       msg = gettext ("No calibration experiments");
00909       goto exit_on_error;
00910     }
00911   buffer = NULL;
00912
00913   // Reading the variables data
00914   for (; child; child = child->next)
00915     {
00916       if (xmlStrcmp (child->name, XML_VARIABLE))
00917         {
00918           snprintf (buffer2, 64, "%s %u: %s",
```

```
00919                     gettext ("Variable"),
00920                     input->nvariables + 1, gettext ("bad XML node"));
00921           msg = buffer2;
00922           goto exit_on_error;
00923         }
00924       if (xmlHasProp (child, XML_NAME))
00925         buffer = xmlGetProp (child, XML_NAME);
00926       else
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("no name"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       if (xmlHasProp (child, XML_MINIMUM))
00935         {
00936           input->rangemin = g_realloc
00937             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00938           input->rangeminabs = g_realloc
00939             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00940           input->rangemin[input->nvariables]
00941             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00942           if (error_code)
00943             {
00944               snprintf (buffer2, 64, "%s %s: %s",
00945                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00946               msg = buffer2;
00947               goto exit_on_error;
00948             }
00949           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00950             {
00951               input->rangeminabs[input->nvariables]
00952                 = xml_node_get_float (child,
     XML_ABSOLUTE_MINIMUM, &error_code);
00953               if (error_code)
00954                 {
00955                   snprintf (buffer2, 64, "%s %s: %s",
00956                             gettext ("Variable"),
00957                             buffer, gettext ("bad absolute minimum"));
00958                   msg = buffer2;
00959                   goto exit_on_error;
00960                 }
00961             }
00962           else
00963             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00964           if (input->rangemin[input->nvariables]
00965               < input->rangeminabs[input->nvariables])
00966             {
00967               snprintf (buffer2, 64, "%s %s: %s",
00968                         gettext ("Variable"),
00969                         buffer, gettext ("minimum range not allowed"));
00970               msg = buffer2;
00971               goto exit_on_error;
00972             }
00973         }
00974       else
00975         {
00976           snprintf (buffer2, 64, "%s %s: %s",
00977                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00978           msg = buffer2;
00979           goto exit_on_error;
00980         }
00981       if (xmlHasProp (child, XML_MAXIMUM))
00982         {
00983           input->rangemax = g_realloc
00984             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00985           input->rangemaxabs = g_realloc
00986             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00987           input->rangemax[input->nvariables]
00988             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00989           if (error_code)
00990             {
00991               snprintf (buffer2, 64, "%s %s: %s",
00992                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00993               msg = buffer2;
00994               goto exit_on_error;
00995             }
00996           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00997             {
00998               input->rangemaxabs[input->nvariables]
00999                 = xml_node_get_float (child,
     XML_ABSOLUTE_MAXIMUM, &error_code);
01000               if (error_code)
01001                 {
01002                   snprintf (buffer2, 64, "%s %s: %s",
01003                             gettext ("Variable"),
```

```
01004                           buffer, gettext ("bad absolute maximum"));
01005                 msg = buffer2;
01006                 goto exit_on_error;
01007               }
01008           }
01009         else
01010           input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01011         if (input->rangemax[input->nvariables]
01012             > input->rangemaxabs[input->nvariables])
01013           {
01014             snprintf (buffer2, 64, "%s %s: %s",
01015                       gettext ("Variable"),
01016                       buffer, gettext ("maximum range not allowed"));
01017             msg = buffer2;
01018             goto exit_on_error;
01019           }
01020       }
01021     else
01022       {
01023         snprintf (buffer2, 64, "%s %s: %s",
01024                   gettext ("Variable"), buffer, gettext ("no maximum range"));
01025         msg = buffer2;
01026         goto exit_on_error;
01027       }
01028     if (input->rangemax[input->nvariables]
01029         < input->rangemin[input->nvariables])
01030       {
01031         snprintf (buffer2, 64, "%s %s: %s",
01032                   gettext ("Variable"), buffer, gettext ("bad range"));
01033         msg = buffer2;
01034         goto exit_on_error;
01035       }
01036     input->precision = g_realloc
01037       (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01038     if (xmlHasProp (child, XML_PRECISION))
01039       {
01040         input->precision[input->nvariables]
01041           = xml_node_get_uint (child, XML_PRECISION, &error_code);
01042         if (error_code || input->precision[input->
    nvariables] >= NPRECISIONS)
01043           {
01044             snprintf (buffer2, 64, "%s %s: %s",
01045                       gettext ("Variable"),
01046                       buffer, gettext ("bad precision"));
01047             msg = buffer2;
01048             goto exit_on_error;
01049           }
01050       }
01051     else
01052       input->precision[input->nvariables] =
    DEFAULT_PRECISION;
01053     if (input->algorithm == ALGORITHM_SWEEP)
01054       {
01055         if (xmlHasProp (child, XML_NSWEEPS))
01056           {
01057             input->nsweeps = (unsigned int *)
01058               g_realloc (input->nsweeps,
01059                          (1 + input->nvariables) * sizeof (unsigned int));
01060             input->nsweeps[input->nvariables]
01061               = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01062             if (error_code || !input->nsweeps[input->
    nvariables])
01063               {
01064                 snprintf (buffer2, 64, "%s %s: %s",
01065                           gettext ("Variable"),
01066                           buffer, gettext ("bad sweeps"));
01067                 msg = buffer2;
01068                 goto exit_on_error;
01069               }
01070           }
01071         else
01072           {
01073             snprintf (buffer2, 64, "%s %s: %s",
01074                       gettext ("Variable"),
01075                       buffer, gettext ("no sweeps number"));
01076             msg = buffer2;
01077             goto exit_on_error;
01078           }
01079 #if DEBUG
01080         fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01081                  input->nsweeps[input->nvariables],
    input->nsimulations);
01082 #endif
01083       }
01084     if (input->algorithm == ALGORITHM_GENETIC)
01085       {
01086         // Obtaining bits representing each variable
```

```
01087           if (xmlHasProp (child, XML_NBITS))
01088             {
01089               input->nbits = (unsigned int *)
01090                 g_realloc (input->nbits,
01091                            (1 + input->nvariables) * sizeof (unsigned int));
01092               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01093               if (error_code || !i)
01094                 {
01095                   snprintf (buffer2, 64, "%s %s: %s",
01096                             gettext ("Variable"),
01097                             buffer, gettext ("invalid bits number"));
01098                   msg = buffer2;
01099                   goto exit_on_error;
01100                 }
01101               input->nbits[input->nvariables] = i;
01102             }
01103           else
01104             {
01105               snprintf (buffer2, 64, "%s %s: %s",
01106                         gettext ("Variable"),
01107                         buffer, gettext ("no bits number"));
01108               msg = buffer2;
01109               goto exit_on_error;
01110             }
01111         }
01112       else if (input->nsteps)
01113         {
01114           input->step = (double *)
01115             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01116           input->step[input->nvariables]
01117             = xml_node_get_float (child, XML_STEP, &error_code);
01118           if (error_code || input->step[input->nvariables] < 0.)
01119             {
01120               snprintf (buffer2, 64, "%s %s: %s",
01121                         gettext ("Variable"),
01122                         buffer, gettext ("bad step size"));
01123               msg = buffer2;
01124               goto exit_on_error;
01125             }
01126         }
01127       input->label = g_realloc
01128         (input->label, (1 + input->nvariables) * sizeof (char *));
01129       input->label[input->nvariables] = (char *) buffer;
01130       ++input->nvariables;
01131     }
01132   if (!input->nvariables)
01133     {
01134       msg = gettext ("No calibration variables");
01135       goto exit_on_error;
01136     }
01137   buffer = NULL;
01138
01139   // Getting the working directory
01140   input->directory = g_path_get_dirname (filename);
01141   input->name = g_path_get_basename (filename);
01142
01143   // Closing the XML document
01144   xmlFreeDoc (doc);
01145
01146 #if DEBUG
01147   fprintf (stderr, "input_open: end\n");
01148 #endif
01149   return 1;
01150
01151 exit_on_error:
01152   xmlFree (buffer);
01153   xmlFreeDoc (doc);
01154   show_error (msg);
01155   input_free ();
01156 #if DEBUG
01157   fprintf (stderr, "input_open: end\n");
01158 #endif
01159   return 0;
01160 }
```

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.

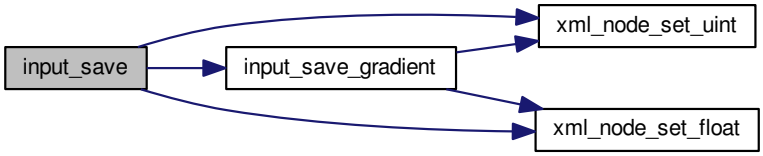
Definition at line 2677 of file mpcotool.c.

```
02678 {
02679   unsigned int i, j;
02680   char *buffer;
02681   xmlDoc *doc;
02682   xmlNode *node, *child;
02683   GFile *file, *file2;
02684
02685 #if DEBUG
02686   fprintf (stderr, "input_save: start\n");
02687 #endif
02688
02689   // Getting the input file directory
02690   input->name = g_path_get_basename (filename);
02691   input->directory = g_path_get_dirname (filename);
02692   file = g_file_new_for_path (input->directory);
02693
02694   // Opening the input file
02695   doc = xmlNewDoc ((const xmlChar *) "1.0");
02696
02697   // Setting root XML node
02698   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02699   xmlDocSetRootElement (doc, node);
02700
02701   // Adding properties to the root XML node
02702   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02703     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02704   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02705     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02706   file2 = g_file_new_for_path (input->simulator);
02707   buffer = g_file_get_relative_path (file, file2);
02708   g_object_unref (file2);
02709   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02710   g_free (buffer);
02711   if (input->evaluator)
02712     {
02713       file2 = g_file_new_for_path (input->evaluator);
02714       buffer = g_file_get_relative_path (file, file2);
02715       g_object_unref (file2);
02716       if (xmlStrlen ((xmlChar *) buffer))
02717         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02718       g_free (buffer);
02719     }
02720   if (input->seed != DEFAULT_RANDOM_SEED)
02721     xml_node_set_uint (node, XML_SEED, input->seed);
02722
02723   // Setting the algorithm
```

```
02724   buffer = (char *) g_malloc (64);
02725   switch (input->algorithm)
02726     {
02727     case ALGORITHM_MONTE_CARLO:
02728       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02729       snprintf (buffer, 64, "%u", input->nsimulations);
02730       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02731       snprintf (buffer, 64, "%u", input->niterations);
02732       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02733       snprintf (buffer, 64, "%.3lg", input->tolerance);
02734       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02735       snprintf (buffer, 64, "%u", input->nbest);
02736       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02737       input_save_gradient (node);
02738       break;
02739     case ALGORITHM_SWEEP:
02740       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02741       snprintf (buffer, 64, "%u", input->niterations);
02742       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02743       snprintf (buffer, 64, "%.3lg", input->tolerance);
02744       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02745       snprintf (buffer, 64, "%u", input->nbest);
02746       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02747       input_save_gradient (node);
02748       break;
02749     default:
02750       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02751       snprintf (buffer, 64, "%u", input->nsimulations);
02752       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02753       snprintf (buffer, 64, "%u", input->niterations);
02754       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02755       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02756       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02757       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02758       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02759       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02760       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02761       break;
02762     }
02763   g_free (buffer);
02764
02765   // Setting the experimental data
02766   for (i = 0; i < input->nexperiments; ++i)
02767     {
02768       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02769       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02770       if (input->weight[i] != 1.)
02771         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02772       for (j = 0; j < input->ninputs; ++j)
02773         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02774     }
02775
02776   // Setting the variables data
02777   for (i = 0; i < input->nvariables; ++i)
02778     {
02779       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02780       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02781       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02782       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02783         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02784       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02785       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02786         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02787       if (input->precision[i] != DEFAULT_PRECISION)
02788         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02789       if (input->algorithm == ALGORITHM_SWEEP)
02790         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02791       else if (input->algorithm == ALGORITHM_GENETIC)
02792         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02793       if (input->nsteps)
02794         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02795     }
02796
02797   // Saving the XML file
02798   xmlSaveFormatFile (filename, doc, 1);
02799
02800   // Freeing memory
02801   xmlFreeDoc (doc);
```

```
02802
02803 #if DEBUG
02804   fprintf (stderr, "input_save: end\n");
02805 #endif
02806 }
```

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.

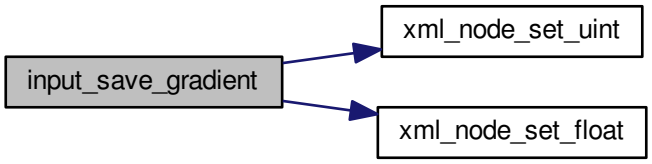
Parameters

node	XML node.

Definition at line 2645 of file mpcotool.c.

```
02646 {
02647 #if DEBUG
02648   fprintf (stderr, "input_save_gradient: start\n");
02649 #endif
02650   if (input->nsteps)
02651     {
02652       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
02653       if (input->relaxation != DEFAULT_RELAXATION)
02654         xml_node_set_float (node, XML_RELAXATION,
      input->relaxation);
02655       switch (input->gradient_method)
02656         {
02657         case GRADIENT_METHOD_COORDINATES:
02658           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02659           break;
02660         default:
02661           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02662           xml_node_set_uint (node, XML_NESTIMATES,
      input->nestimates);
02663         }
02664     }
02665 #if DEBUG
02666   fprintf (stderr, "input_save_gradient: end\n");
02667 #endif
02668 }
```

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

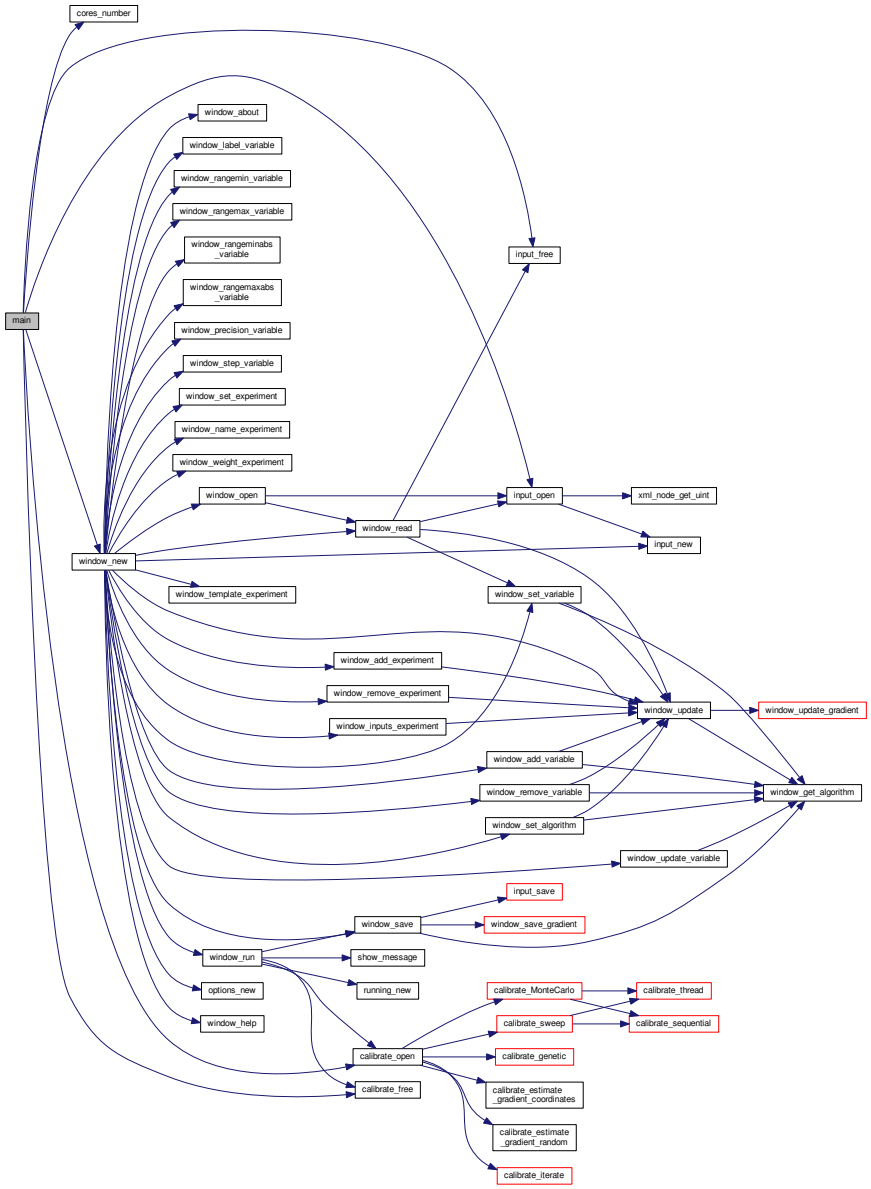
0 on success, >0 on error.

Definition at line 4856 of file mpcotool.c.

```
04857 {
04858 #if HAVE_GTK
04859   char *buffer;
04860 #endif
04861
04862   // Starting pseudo-random numbers generator
04863   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04864   calibrate->seed = DEFAULT_RANDOM_SEED;
04865
04866   // Allowing spaces in the XML data file
04867   xmlKeepBlanksDefault (0);
04868
04869   // Starting MPI
04870 #if HAVE_MPI
04871   MPI_Init (&argn, &argc);
04872   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04873   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04874   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04875 #else
04876   ntasks = 1;
04877 #endif
04878
04879 #if HAVE_GTK
04880
04881   // Getting threads number
04882   nthreads_gradient = nthreads = cores_number ();
04883
04884   // Setting local language and international floating point numbers notation
04885   setlocale (LC_ALL, "");
04886   setlocale (LC_NUMERIC, "C");
04887   window->application_directory = g_get_current_dir ();
04888   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04889   bindtextdomain (PROGRAM_INTERFACE, buffer);
04890   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04891   textdomain (PROGRAM_INTERFACE);
04892
04893   // Initing GTK+
04894   gtk_disable_setlocale ();
04895   gtk_init (&argn, &argc);
```

```
04896
04897   // Opening the main window
04898   window_new ();
04899   gtk_main ();
04900
04901   // Freeing memory
04902   input_free ();
04903   g_free (buffer);
04904   gtk_widget_destroy (GTK_WIDGET (window->window));
04905   g_free (window->application_directory);
04906
04907 #else
04908
04909   // Checking syntax
04910   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04911     {
04912       printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04913       return 1;
04914     }
04915
04916   // Getting threads number
04917   if (argn == 2)
04918     nthreads_gradient = nthreads = cores_number ();
04919   else
04920     {
04921       nthreads_gradient = nthreads = atoi (argc[2]);
04922       if (!nthreads)
04923         {
04924           printf ("Bad threads number\n");
04925           return 2;
04926         }
04927     }
04928   printf ("nthreads=%u\n", nthreads);
04929
04930   // Making calibration
04931   if (input_open (argc[argn - 1]))
04932     calibrate_open ();
04933
04934   // Freeing memory
04935   calibrate_free ();
04936
04937 #endif
04938
04939   // Closing MPI
04940 #if HAVE_MPI
04941   MPI_Finalize ();
04942 #endif
04943
04944   // Freeing memory
04945   gsl_rng_free (calibrate->rng);
04946
04947   // Closing
04948   return 0;
04949 }
```

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.

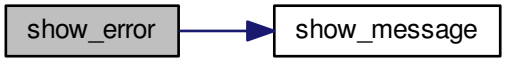
Parameters

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 2910 of file mpcotool.c.

```
02911 {
02912   unsigned int i;
02913 #if DEBUG
02914   fprintf (stderr, "window_get_algorithm: start\n");
02915 #endif
02916   for (i = 0; i < NALGORITHMS; ++i)
02917     if (gtk_toggle_button_get_active
02918         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02919       break;
02920 #if DEBUG
```

```
02921   fprintf (stderr, "window_get_algorithm: %u\n", i);
02922   fprintf (stderr, "window_get_algorithm: end\n");
02923 #endif
02924   return i;
02925 }
```

5.5.2.22 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2933 of file mpcotool.c.

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

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.

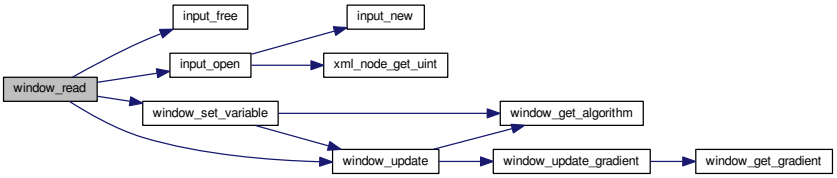
Definition at line 4022 of file mpcotool.c.

```
04023 {
04024   unsigned int i;
04025   char *buffer;
04026 #if DEBUG
04027   fprintf (stderr, "window_read: start\n");
04028 #endif
04029
04030   // Reading new input file
04031   input_free ();
04032   if (!input_open (filename))
04033     return 0;
04034
04035   // Setting GTK+ widgets data
04036   gtk_entry_set_text (window->entry_result, input->result);
04037   gtk_entry_set_text (window->entry_variables, input->
      variables);
04038   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
04039   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04040                                  (window->button_simulator), buffer);
04041   g_free (buffer);
04042   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04043                                 (size_t) input->evaluator);
04044   if (input->evaluator)
```

```
04045     {
04046       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
04047       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04048                                      (window->button_evaluator), buffer);
04049       g_free (buffer);
04050     }
04051   gtk_toggle_button_set_active
04052     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
04053   switch (input->algorithm)
04054     {
04055     case ALGORITHM_MONTE_CARLO:
04056       gtk_spin_button_set_value (window->spin_simulations,
04057                                  (gdouble) input->nsimulations);
04058     case ALGORITHM_SWEEP:
04059       gtk_spin_button_set_value (window->spin_iterations,
04060                                  (gdouble) input->niterations);
04061       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
04062       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
04063       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
04064                                     input->nsteps);
04065       if (input->nsteps)
04066         {
04067           gtk_toggle_button_set_active
04068             (GTK_TOGGLE_BUTTON (window->button_gradient
04069                                 [input->gradient_method]), TRUE);
04070           gtk_spin_button_set_value (window->spin_steps,
04071                                      (gdouble) input->nsteps);
04072           gtk_spin_button_set_value (window->spin_relaxation,
04073                                      (gdouble) input->relaxation);
04074           switch (input->gradient_method)
04075             {
04076             case GRADIENT_METHOD_RANDOM:
04077               gtk_spin_button_set_value (window->spin_estimates,
04078                                          (gdouble) input->nestimates);
04079             }
04080         }
04081       break;
04082     default:
04083       gtk_spin_button_set_value (window->spin_population,
04084                                  (gdouble) input->nsimulations);
04085       gtk_spin_button_set_value (window->spin_generations,
04086                                  (gdouble) input->niterations);
04087       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04088       gtk_spin_button_set_value (window->spin_reproduction,
04089                                  input->reproduction_ratio);
04090       gtk_spin_button_set_value (window->spin_adaptation,
04091                                  input->adaptation_ratio);
04092     }
04093   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04094   g_signal_handler_block (window->button_experiment,
04095                           window->id_experiment_name);
04096   gtk_combo_box_text_remove_all (window->combo_experiment);
04097   for (i = 0; i < input->nexperiments; ++i)
04098     gtk_combo_box_text_append_text (window->combo_experiment,
04099                                     input->experiment[i]);
04100   g_signal_handler_unblock
04101     (window->button_experiment, window->
      id_experiment_name);
04102   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
04103   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04104   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04105   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04106   gtk_combo_box_text_remove_all (window->combo_variable);
04107   for (i = 0; i < input->nvariables; ++i)
04108     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
04109   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04110   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04111   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04112   window_set_variable ();
04113   window_update ();
04114
04115 #if DEBUG
04116   fprintf (stderr, "window_read: end\n");
04117 #endif
```

```
04118   return 1;
04119 }
```

Here is the call graph for this function:



5.5.2.24 int window_save ( )

Function to save the input file.

Returns

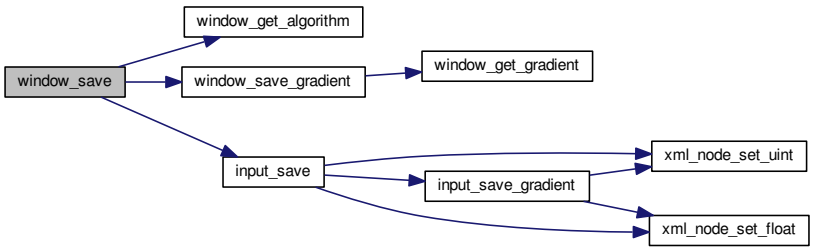
1 on OK, 0 on Cancel.

Definition at line 2988 of file mpcotool.c.

```
02989 {
02990   char *buffer;
02991   GtkFileChooserDialog *dlg;
02992
02993 #if DEBUG
02994   fprintf (stderr, "window_save: start\n");
02995 #endif
02996
02997   // Opening the saving dialog
02998   dlg = (GtkFileChooserDialog *)
02999     gtk_file_chooser_dialog_new (gettext ("Save file"),
03000                                  window->window,
03001                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03002                                  gettext ("_Cancel"),
03003                                  GTK_RESPONSE_CANCEL,
03004                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03005   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03006   buffer = g_build_filename (input->directory, input->name, NULL);
03007   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03008   g_free (buffer);
03009
03010   // If OK response then saving
03011   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03012     {
03013
03014       // Adding properties to the root XML node
03015       input->simulator = gtk_file_chooser_get_filename
03016         (GTK_FILE_CHOOSER (window->button_simulator));
03017       if (gtk_toggle_button_get_active
03018           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03019         input->evaluator = gtk_file_chooser_get_filename
03020           (GTK_FILE_CHOOSER (window->button_evaluator));
03021       else
03022         input->evaluator = NULL;
03023       input->result
03024         = (char *) xmlStrdup ((const xmlChar *)
03025                               gtk_entry_get_text (window->entry_result));
03026       input->variables
03027         = (char *) xmlStrdup ((const xmlChar *)
03028                               gtk_entry_get_text (window->entry_variables));
03029
03030       // Setting the algorithm
03031       switch (window_get_algorithm ())
03032         {
03033         case ALGORITHM_MONTE_CARLO:
03034           input->algorithm = ALGORITHM_MONTE_CARLO;
03035           input->nsimulations
```

```
03036             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03037           input->niterations
03038             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03039           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03040           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03041           window_save_gradient ();
03042           break;
03043         case ALGORITHM_SWEEP:
03044           input->algorithm = ALGORITHM_SWEEP;
03045           input->niterations
03046             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03047           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
03048           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
03049           window_save_gradient ();
03050           break;
03051         default:
03052           input->algorithm = ALGORITHM_GENETIC;
03053           input->nsimulations
03054             = gtk_spin_button_get_value_as_int (window->spin_population);
03055           input->niterations
03056             = gtk_spin_button_get_value_as_int (window->spin_generations);
03057           input->mutation_ratio
03058             = gtk_spin_button_get_value (window->spin_mutation);
03059           input->reproduction_ratio
03060             = gtk_spin_button_get_value (window->spin_reproduction);
03061           input->adaptation_ratio
03062             = gtk_spin_button_get_value (window->spin_adaptation);
03063           break;
03064         }
03065
03066       // Saving the XML file
03067       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03068       input_save (buffer);
03069
03070       // Closing and freeing memory
03071       g_free (buffer);
03072       gtk_widget_destroy (GTK_WIDGET (dlg));
03073 #if DEBUG
03074       fprintf (stderr, "window_save: end\n");
03075 #endif
03076       return 1;
03077     }
03078
03079   // Closing and freeing memory
03080   gtk_widget_destroy (GTK_WIDGET (dlg));
03081 #if DEBUG
03082   fprintf (stderr, "window_save: end\n");
03083 #endif
03084   return 0;
03085 }
```

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 3626 of file mpcotool.c.

```
03627 {
03628   unsigned int i, j;
03629   char *buffer;
03630   GFile *file1, *file2;
03631 #if DEBUG
03632   fprintf (stderr, "window_template_experiment: start\n");
03633 #endif
03634   i = (size_t) data;
03635   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03636   file1
03637     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03638   file2 = g_file_new_for_path (input->directory);
03639   buffer = g_file_get_relative_path (file2, file1);
03640   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03641   g_free (buffer);
03642   g_object_unref (file2);
03643   g_object_unref (file1);
03644 #if DEBUG
03645   fprintf (stderr, "window_template_experiment: end\n");
03646 #endif
03647 }
```

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 337 of file mpcotool.c.

```
00338 {
00339   double x = 0.;
00340   xmlChar *buffer;
00341   buffer = xmlGetProp (node, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350       xmlFree (buffer);
00351     }
00352   return x;
00353 }
```

5.5.2.27 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 }
```

5.5.2.28 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 }
```

5.5.2.29 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 404 of file mpcotool.c.

```
00405 {
00406   xmlChar buffer[64];
00407   snprintf ((char *) buffer, 64, "%.14lg", value);
00408   xmlSetProp (node, prop, buffer);
00409 }
```

5.5.2.30 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 366 of file mpcotool.c.

```
00367 {
00368   xmlChar buffer[64];
00369   snprintf ((char *) buffer, 64, "%d", value);
00370   xmlSetProp (node, prop, buffer);
00371 }
```

5.5.2.31 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 385 of file mpcotool.c.

```
00386 {
00387   xmlChar buffer[64];
00388   snprintf ((char *) buffer, 64, "%u", value);
00389   xmlSetProp (node, prop, buffer);
00390 }
```

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
00336 double
00337 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00338 {
00339   double x = 0.;
00340   xmlChar *buffer;
00341   buffer = xmlGetProp (node, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350       xmlFree (buffer);
00351     }
00352   return x;
00353 }
00354
00365 void
00366 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00367 {
00368   xmlChar buffer[64];
00369   snprintf ((char *) buffer, 64, "%d", value);
00370   xmlSetProp (node, prop, buffer);
00371 }
00372
00384 void
00385 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00386 {
00387   xmlChar buffer[64];
00388   snprintf ((char *) buffer, 64, "%u", value);
00389   xmlSetProp (node, prop, buffer);
00390 }
00391
00403 void
00404 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00405 {
00406   xmlChar buffer[64];
00407   snprintf ((char *) buffer, 64, "%.14lg", value);
00408   xmlSetProp (node, prop, buffer);
00409 }
00410
00415 void
00416 input_new ()
```

```
00417 {
00418   unsigned int i;
00419 #if DEBUG
00420   fprintf (stderr, "input_new: start\n");
00421 #endif
00422   input->nvariables = input->nexperiments = input->ninputs = input->
      nsteps = 0;
00423   input->simulator = input->evaluator = input->directory = input->
      name
00424     = input->result = input->variables = NULL;
00425   input->experiment = input->label = NULL;
00426   input->precision = input->nsweeps = input->nbits = NULL;
00427   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00428     = input->weight = input->step = NULL;
00429   for (i = 0; i < MAX_NINPUTS; ++i)
00430     input->template[i] = NULL;
00431 #if DEBUG
00432   fprintf (stderr, "input_new: end\n");
00433 #endif
00434 }
00435
00440 void
00441 input_free ()
00442 {
00443   unsigned int i, j;
00444 #if DEBUG
00445   fprintf (stderr, "input_free: start\n");
00446 #endif
00447   g_free (input->name);
00448   g_free (input->directory);
00449   for (i = 0; i < input->nexperiments; ++i)
00450     {
00451       xmlFree (input->experiment[i]);
00452       for (j = 0; j < input->ninputs; ++j)
00453         xmlFree (input->template[j][i]);
00454       g_free (input->template[j]);
00455     }
00456   g_free (input->experiment);
00457   for (i = 0; i < input->ninputs; ++i)
00458     g_free (input->template[i]);
00459   for (i = 0; i < input->nvariables; ++i)
00460     xmlFree (input->label[i]);
00461   g_free (input->label);
00462   g_free (input->precision);
00463   g_free (input->rangemin);
00464   g_free (input->rangemax);
00465   g_free (input->rangeminabs);
00466   g_free (input->rangemaxabs);
00467   g_free (input->weight);
00468   g_free (input->step);
00469   g_free (input->nsweeps);
00470   g_free (input->nbits);
00471   xmlFree (input->evaluator);
00472   xmlFree (input->simulator);
00473   xmlFree (input->result);
00474   xmlFree (input->variables);
00475   input->nexperiments = input->ninputs = input->nvariables = input->
      nsteps = 0;
00476 #if DEBUG
00477   fprintf (stderr, "input_free: end\n");
00478 #endif
00479 }
00480
00488 int
00489 input_open (char *filename)
00490 {
00491   char buffer2[64];
00492   char *buffert[MAX_NINPUTS] =
00493     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00494   xmlDoc *doc;
00495   xmlNode *node, *child;
00496   xmlChar *buffer;
00497   char *msg;
00498   int error_code;
00499   unsigned int i;
00500
00501 #if DEBUG
00502   fprintf (stderr, "input_open: start\n");
00503 #endif
00504
00505   // Resetting input data
00506   buffer = NULL;
00507   input_new ();
00508
00509   // Parsing the input file
00510 #if DEBUG
```

```
00511   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00512 #endif
00513   doc = xmlParseFile (filename);
00514   if (!doc)
00515     {
00516       msg = gettext ("Unable to parse the input file");
00517       goto exit_on_error;
00518     }
00519
00520   // Getting the root node
00521 #if DEBUG
00522   fprintf (stderr, "input_open: getting the root node\n");
00523 #endif
00524   node = xmlDocGetRootElement (doc);
00525   if (xmlStrcmp (node->name, XML_CALIBRATE))
00526     {
00527       msg = gettext ("Bad root XML node");
00528       goto exit_on_error;
00529     }
00530
00531   // Getting results file names
00532   input->result = (char *) xmlGetProp (node, XML_RESULT);
00533   if (!input->result)
00534     input->result = (char *) xmlStrdup (result_name);
00535   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00536   if (!input->variables)
00537     input->variables = (char *) xmlStrdup (variables_name);
00538
00539   // Opening simulator program name
00540   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00541   if (!input->simulator)
00542     {
00543       msg = gettext ("Bad simulator program");
00544       goto exit_on_error;
00545     }
00546
00547   // Opening evaluator program name
00548   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00549
00550   // Obtaining pseudo-random numbers generator seed
00551   if (!xmlHasProp (node, XML_SEED))
00552     input->seed = DEFAULT_RANDOM_SEED;
00553   else
00554     {
00555       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00556       if (error_code)
00557         {
00558           msg = gettext ("Bad pseudo-random numbers generator seed");
00559           goto exit_on_error;
00560         }
00561     }
00562
00563   // Opening algorithm
00564   buffer = xmlGetProp (node, XML_ALGORITHM);
00565   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00566     {
00567       input->algorithm = ALGORITHM_MONTE_CARLO;
00568
00569       // Obtaining simulations number
00570       input->nsimulations
00571         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00572       if (error_code)
00573         {
00574           msg = gettext ("Bad simulations number");
00575           goto exit_on_error;
00576         }
00577     }
00578   else if (!xmlStrcmp (buffer, XML_SWEEP))
00579     input->algorithm = ALGORITHM_SWEEP;
00580   else if (!xmlStrcmp (buffer, XML_GENETIC))
00581     {
00582       input->algorithm = ALGORITHM_GENETIC;
00583
00584       // Obtaining population
00585       if (xmlHasProp (node, XML_NPOPULATION))
00586         {
00587           input->nsimulations
00588             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00589           if (error_code || input->nsimulations < 3)
00590             {
00591               msg = gettext ("Invalid population number");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No population number");
```

```
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining generations
00602       if (xmlHasProp (node, XML_NGENERATIONS))
00603         {
00604           input->niterations
00605             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00606           if (error_code || !input->niterations)
00607             {
00608               msg = gettext ("Invalid generations number");
00609               goto exit_on_error;
00610             }
00611         }
00612       else
00613         {
00614           msg = gettext ("No generations number");
00615           goto exit_on_error;
00616         }
00617
00618       // Obtaining mutation probability
00619       if (xmlHasProp (node, XML_MUTATION))
00620         {
00621           input->mutation_ratio
00622             = xml_node_get_float (node, XML_MUTATION, &error_code);
00623           if (error_code || input->mutation_ratio < 0.
00624               || input->mutation_ratio >= 1.)
00625             {
00626               msg = gettext ("Invalid mutation probability");
00627               goto exit_on_error;
00628             }
00629         }
00630       else
00631         {
00632           msg = gettext ("No mutation probability");
00633           goto exit_on_error;
00634         }
00635
00636       // Obtaining reproduction probability
00637       if (xmlHasProp (node, XML_REPRODUCTION))
00638         {
00639           input->reproduction_ratio
00640             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00641           if (error_code || input->reproduction_ratio < 0.
00642               || input->reproduction_ratio >= 1.0)
00643             {
00644               msg = gettext ("Invalid reproduction probability");
00645               goto exit_on_error;
00646             }
00647         }
00648       else
00649         {
00650           msg = gettext ("No reproduction probability");
00651           goto exit_on_error;
00652         }
00653
00654       // Obtaining adaptation probability
00655       if (xmlHasProp (node, XML_ADAPTATION))
00656         {
00657           input->adaptation_ratio
00658             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00659           if (error_code || input->adaptation_ratio < 0.
00660               || input->adaptation_ratio >= 1.)
00661             {
00662               msg = gettext ("Invalid adaptation probability");
00663               goto exit_on_error;
00664             }
00665         }
00666       else
00667         {
00668           msg = gettext ("No adaptation probability");
00669           goto exit_on_error;
00670         }
00671
00672       // Checking survivals
00673       i = input->mutation_ratio * input->nsimulations;
00674       i += input->reproduction_ratio * input->nsimulations;
00675       i += input->adaptation_ratio * input->nsimulations;
00676       if (i > input->nsimulations - 2)
00677         {
00678           msg = gettext
00679             ("No enough survival entities to reproduce the population");
00680           goto exit_on_error;
00681         }
00682     }
00683   else
00684     {
```

```
00685       msg = gettext ("Unknown algorithm");
00686       goto exit_on_error;
00687     }
00688   xmlFree (buffer);
00689   buffer = NULL;
00690
00691   if (input->algorithm == ALGORITHM_MONTE_CARLO
00692       || input->algorithm == ALGORITHM_SWEEP)
00693     {
00694
00695       // Obtaining iterations number
00696       input->niterations
00697         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00698       if (error_code == 1)
00699         input->niterations = 1;
00700       else if (error_code)
00701         {
00702           msg = gettext ("Bad iterations number");
00703           goto exit_on_error;
00704         }
00705
00706       // Obtaining best number
00707       if (xmlHasProp (node, XML_NBEST))
00708         {
00709           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00710           if (error_code || !input->nbest)
00711             {
00712               msg = gettext ("Invalid best number");
00713               goto exit_on_error;
00714             }
00715         }
00716       else
00717         input->nbest = 1;
00718
00719       // Obtaining tolerance
00720       if (xmlHasProp (node, XML_TOLERANCE))
00721         {
00722           input->tolerance
00723             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00724           if (error_code || input->tolerance < 0.)
00725             {
00726               msg = gettext ("Invalid tolerance");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         input->tolerance = 0.;
00732
00733       // Getting gradient method parameters
00734       if (xmlHasProp (node, XML_NSTEPS))
00735         {
00736           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00737           if (error_code || !input->nsteps)
00738             {
00739               msg = gettext ("Invalid steps number");
00740               goto exit_on_error;
00741             }
00742           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00743           if (!xmlStrcmp (buffer, XML_COORDINATES))
00744             input->gradient_method = GRADIENT_METHOD_COORDINATES;
00745           else if (!xmlStrcmp (buffer, XML_RANDOM))
00746             {
00747               input->gradient_method = GRADIENT_METHOD_RANDOM;
00748               input->nestimates
00749                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00750               if (error_code || !input->nestimates)
00751                 {
00752                   msg = gettext ("Invalid estimates number");
00753                   goto exit_on_error;
00754                 }
00755             }
00756           else
00757             {
00758               msg = gettext ("Unknown method to estimate the gradient");
00759               goto exit_on_error;
00760             }
00761           xmlFree (buffer);
00762           buffer = NULL;
00763           if (xmlHasProp (node, XML_RELAXATION))
00764             {
00765               input->relaxation
00766                 = xml_node_get_float (node, XML_RELAXATION, &error_code);
00767               if (error_code || input->relaxation < 0.
00768                   || input->relaxation > 2.)
00769                 {
```

```
00770                   msg = gettext ("Invalid relaxation parameter");
00771                   goto exit_on_error;
00772                 }
00773             }
00774           else
00775             input->relaxation = DEFAULT_RELAXATION;
00776         }
00777       else
00778         input->nsteps = 0;
00779     }
00780
00781   // Reading the experimental data
00782   for (child = node->children; child; child = child->next)
00783     {
00784       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00785         break;
00786 #if DEBUG
00787       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00788 #endif
00789       if (xmlHasProp (child, XML_NAME))
00790         buffer = xmlGetProp (child, XML_NAME);
00791       else
00792         {
00793           snprintf (buffer2, 64, "%s %u: %s",
00794                     gettext ("Experiment"),
00795                     input->nexperiments + 1, gettext ("no data file name"));
00796           msg = buffer2;
00797           goto exit_on_error;
00798         }
00799 #if DEBUG
00800       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00801 #endif
00802       input->weight = g_realloc (input->weight,
00803                                  (1 + input->nexperiments) * sizeof (double));
00804       if (xmlHasProp (child, XML_WEIGHT))
00805         {
00806           input->weight[input->nexperiments]
00807             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00808           if (error_code)
00809             {
00810               snprintf (buffer2, 64, "%s %s: %s",
00811                         gettext ("Experiment"), buffer, gettext ("bad weight"));
00812               msg = buffer2;
00813               goto exit_on_error;
00814             }
00815         }
00816       else
00817         input->weight[input->nexperiments] = 1.;
00818 #if DEBUG
00819       fprintf (stderr, "input_open: weight=%lg\n",
00820                input->weight[input->nexperiments]);
00821 #endif
00822       if (!input->nexperiments)
00823         input->ninputs = 0;
00824 #if DEBUG
00825       fprintf (stderr, "input_open: template[0]\n");
00826 #endif
00827       if (xmlHasProp (child, XML_TEMPLATE1))
00828         {
00829           input->template[0]
00830             = (char **) g_realloc (input->template[0],
00831                                    (1 + input->nexperiments) * sizeof (char *));
00832           buffert[0] = (char *) xmlGetProp (child, template[0]);
00833 #if DEBUG
00834           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00835                    input->nexperiments, buffert[0]);
00836 #endif
00837           if (!input->nexperiments)
00838             ++input->ninputs;
00839 #if DEBUG
00840           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00841 #endif
00842         }
00843       else
00844         {
00845           snprintf (buffer2, 64, "%s %s: %s",
00846                     gettext ("Experiment"), buffer, gettext ("no template"));
00847           msg = buffer2;
00848           goto exit_on_error;
00849         }
00850       for (i = 1; i < MAX_NINPUTS; ++i)
00851         {
00852 #if DEBUG
00853           fprintf (stderr, "input_open: template%u\n", i + 1);
00854 #endif
00855           if (xmlHasProp (child, template[i]))
00856             {
```

```
00857               if (input->nexperiments && input->ninputs <= i)
00858                 {
00859                   snprintf (buffer2, 64, "%s %s: %s",
00860                             gettext ("Experiment"),
00861                             buffer, gettext ("bad templates number"));
00862                   msg = buffer2;
00863                   while (i-- > 0)
00864                     xmlFree (buffert[i]);
00865                   goto exit_on_error;
00866                 }
00867               input->template[i] = (char **)
00868                 g_realloc (input->template[i],
00869                            (1 + input->nexperiments) * sizeof (char *));
00870               buffert[i] = (char *) xmlGetProp (child, template[i]);
00871 #if DEBUG
00872               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00873                        input->nexperiments, i + 1,
00874                        input->template[i][input->nexperiments]);
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 if (input->nexperiments && input->ninputs >= i)
00883             {
00884               snprintf (buffer2, 64, "%s %s: %s%u",
00885                         gettext ("Experiment"),
00886                         buffer, gettext ("no template"), i + 1);
00887               msg = buffer2;
00888               while (i-- > 0)
00889                 xmlFree (buffert[i]);
00890               goto exit_on_error;
00891             }
00892           else
00893             break;
00894         }
00895       input->experiment
00896         = g_realloc (input->experiment,
00897                      (1 + input->nexperiments) * sizeof (char *));
00898       input->experiment[input->nexperiments] = (char *) buffer;
00899       for (i = 0; i < input->ninputs; ++i)
00900         input->template[i][input->nexperiments] = buffert[i];
00901       ++input->nexperiments;
00902 #if DEBUG
00903       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00904 #endif
00905     }
00906   if (!input->nexperiments)
00907     {
00908       msg = gettext ("No calibration experiments");
00909       goto exit_on_error;
00910     }
00911   buffer = NULL;
00912
00913   // Reading the variables data
00914   for (; child; child = child->next)
00915     {
00916       if (xmlStrcmp (child->name, XML_VARIABLE))
00917         {
00918           snprintf (buffer2, 64, "%s %u: %s",
00919                     gettext ("Variable"),
00920                     input->nvariables + 1, gettext ("bad XML node"));
00921           msg = buffer2;
00922           goto exit_on_error;
00923         }
00924       if (xmlHasProp (child, XML_NAME))
00925         buffer = xmlGetProp (child, XML_NAME);
00926       else
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("no name"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       if (xmlHasProp (child, XML_MINIMUM))
00935         {
00936           input->rangemin = g_realloc
00937             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00938           input->rangeminabs = g_realloc
00939             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00940           input->rangemin[input->nvariables]
00941             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00942           if (error_code)
00943             {
```

```
00944               snprintf (buffer2, 64, "%s %s: %s",
00945                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00946               msg = buffer2;
00947               goto exit_on_error;
00948             }
00949           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00950             {
00951               input->rangeminabs[input->nvariables]
00952                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00953               if (error_code)
00954                 {
00955                   snprintf (buffer2, 64, "%s %s: %s",
00956                             gettext ("Variable"),
00957                             buffer, gettext ("bad absolute minimum"));
00958                   msg = buffer2;
00959                   goto exit_on_error;
00960                 }
00961             }
00962           else
00963             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00964           if (input->rangemin[input->nvariables]
00965               < input->rangeminabs[input->nvariables])
00966             {
00967               snprintf (buffer2, 64, "%s %s: %s",
00968                         gettext ("Variable"),
00969                         buffer, gettext ("minimum range not allowed"));
00970               msg = buffer2;
00971               goto exit_on_error;
00972             }
00973         }
00974       else
00975         {
00976           snprintf (buffer2, 64, "%s %s: %s",
00977                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00978           msg = buffer2;
00979           goto exit_on_error;
00980         }
00981       if (xmlHasProp (child, XML_MAXIMUM))
00982         {
00983           input->rangemax = g_realloc
00984             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00985           input->rangemaxabs = g_realloc
00986             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00987           input->rangemax[input->nvariables]
00988             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00989           if (error_code)
00990             {
00991               snprintf (buffer2, 64, "%s %s: %s",
00992                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00993               msg = buffer2;
00994               goto exit_on_error;
00995             }
00996           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00997             {
00998               input->rangemaxabs[input->nvariables]
00999                 = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
01000               if (error_code)
01001                 {
01002                   snprintf (buffer2, 64, "%s %s: %s",
01003                             gettext ("Variable"),
01004                             buffer, gettext ("bad absolute maximum"));
01005                   msg = buffer2;
01006                   goto exit_on_error;
01007                 }
01008             }
01009           else
01010             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01011           if (input->rangemax[input->nvariables]
01012               > input->rangemaxabs[input->nvariables])
01013             {
01014               snprintf (buffer2, 64, "%s %s: %s",
01015                         gettext ("Variable"),
01016                         buffer, gettext ("maximum range not allowed"));
01017               msg = buffer2;
01018               goto exit_on_error;
01019             }
01020         }
01021       else
01022         {
01023           snprintf (buffer2, 64, "%s %s: %s",
01024                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01025           msg = buffer2;
01026           goto exit_on_error;
01027         }
01028       if (input->rangemax[input->nvariables]
```

```
01029           < input->rangemin[input->nvariables])
01030         {
01031           snprintf (buffer2, 64, "%s %s: %s",
01032                     gettext ("Variable"), buffer, gettext ("bad range"));
01033           msg = buffer2;
01034           goto exit_on_error;
01035         }
01036       input->precision = g_realloc
01037         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01038       if (xmlHasProp (child, XML_PRECISION))
01039         {
01040           input->precision[input->nvariables]
01041             = xml_node_get_uint (child, XML_PRECISION, &error_code);
01042           if (error_code || input->precision[input->nvariables] >=
      NPRECISIONS)
01043             {
01044               snprintf (buffer2, 64, "%s %s: %s",
01045                         gettext ("Variable"),
01046                         buffer, gettext ("bad precision"));
01047               msg = buffer2;
01048               goto exit_on_error;
01049             }
01050         }
01051       else
01052         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
01053       if (input->algorithm == ALGORITHM_SWEEP)
01054         {
01055           if (xmlHasProp (child, XML_NSWEEPS))
01056             {
01057               input->nsweeps = (unsigned int *)
01058                 g_realloc (input->nsweeps,
01059                            (1 + input->nvariables) * sizeof (unsigned int));
01060               input->nsweeps[input->nvariables]
01061                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01062               if (error_code || !input->nsweeps[input->nvariables])
01063                 {
01064                   snprintf (buffer2, 64, "%s %s: %s",
01065                             gettext ("Variable"),
01066                             buffer, gettext ("bad sweeps"));
01067                   msg = buffer2;
01068                   goto exit_on_error;
01069                 }
01070             }
01071           else
01072             {
01073               snprintf (buffer2, 64, "%s %s: %s",
01074                         gettext ("Variable"),
01075                         buffer, gettext ("no sweeps number"));
01076               msg = buffer2;
01077               goto exit_on_error;
01078             }
01079 #if DEBUG
01080           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01081                    input->nsweeps[input->nvariables], input->
      nsimulations);
01082 #endif
01083         }
01084       if (input->algorithm == ALGORITHM_GENETIC)
01085         {
01086           // Obtaining bits representing each variable
01087           if (xmlHasProp (child, XML_NBITS))
01088             {
01089               input->nbits = (unsigned int *)
01090                 g_realloc (input->nbits,
01091                            (1 + input->nvariables) * sizeof (unsigned int));
01092               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01093               if (error_code || !i)
01094                 {
01095                   snprintf (buffer2, 64, "%s %s: %s",
01096                             gettext ("Variable"),
01097                             buffer, gettext ("invalid bits number"));
01098                   msg = buffer2;
01099                   goto exit_on_error;
01100                 }
01101               input->nbits[input->nvariables] = i;
01102             }
01103           else
01104             {
01105               snprintf (buffer2, 64, "%s %s: %s",
01106                         gettext ("Variable"),
01107                         buffer, gettext ("no bits number"));
01108               msg = buffer2;
01109               goto exit_on_error;
01110             }
01111         }
01112       else if (input->nsteps)
```

```
01113         {
01114           input->step = (double *)
01115             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01116           input->step[input->nvariables]
01117             = xml_node_get_float (child, XML_STEP, &error_code);
01118           if (error_code || input->step[input->nvariables] < 0.)
01119             {
01120               snprintf (buffer2, 64, "%s %s: %s",
01121                         gettext ("Variable"),
01122                         buffer, gettext ("bad step size"));
01123               msg = buffer2;
01124               goto exit_on_error;
01125             }
01126         }
01127       input->label = g_realloc
01128         (input->label, (1 + input->nvariables) * sizeof (char *));
01129       input->label[input->nvariables] = (char *) buffer;
01130       ++input->nvariables;
01131     }
01132   if (!input->nvariables)
01133     {
01134       msg = gettext ("No calibration variables");
01135       goto exit_on_error;
01136     }
01137   buffer = NULL;
01138
01139   // Getting the working directory
01140   input->directory = g_path_get_dirname (filename);
01141   input->name = g_path_get_basename (filename);
01142
01143   // Closing the XML document
01144   xmlFreeDoc (doc);
01145
01146 #if DEBUG
01147   fprintf (stderr, "input_open: end\n");
01148 #endif
01149   return 1;
01150
01151 exit_on_error:
01152   xmlFree (buffer);
01153   xmlFreeDoc (doc);
01154   show_error (msg);
01155   input_free ();
01156 #if DEBUG
01157   fprintf (stderr, "input_open: end\n");
01158 #endif
01159   return 0;
01160 }
01161
01173 void
01174 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01175 {
01176   unsigned int i;
01177   char buffer[32], value[32], *buffer2, *buffer3, *content;
01178   FILE *file;
01179   gsize length;
01180   GRegex *regex;
01181
01182 #if DEBUG
01183   fprintf (stderr, "calibrate_input: start\n");
01184 #endif
01185
01186   // Checking the file
01187   if (!template)
01188     goto calibrate_input_end;
01189
01190   // Opening template
01191   content = g_mapped_file_get_contents (template);
01192   length = g_mapped_file_get_length (template);
01193 #if DEBUG
01194   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01195            content);
01196 #endif
01197   file = g_fopen (input, "w");
01198
01199   // Parsing template
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
01202 #if DEBUG
01203       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01204 #endif
01205       snprintf (buffer, 32, "@variable%u@", i + 1);
01206       regex = g_regex_new (buffer, 0, 0, NULL);
01207       if (i == 0)
01208         {
01209           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01210                                              calibrate->label[i], 0, NULL);
```

```
01211 #if DEBUG
01212           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01213 #endif
01214         }
01215       else
01216         {
01217           length = strlen (buffer3);
01218           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01219                                              calibrate->label[i], 0, NULL);
01220           g_free (buffer3);
01221         }
01222       g_regex_unref (regex);
01223       length = strlen (buffer2);
01224       snprintf (buffer, 32, "@value%u@", i + 1);
01225       regex = g_regex_new (buffer, 0, 0, NULL);
01226       snprintf (value, 32, format[calibrate->precision[i]],
01227                 calibrate->value[simulation * calibrate->nvariables + i]);
01228
01229 #if DEBUG
01230       fprintf (stderr, "calibrate_input: value=%s\n", value);
01231 #endif
01232       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01233                                          0, NULL);
01234       g_free (buffer2);
01235       g_regex_unref (regex);
01236     }
01237
01238   // Saving input file
01239   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01240   g_free (buffer3);
01241   fclose (file);
01242
01243 calibrate_input_end:
01244 #if DEBUG
01245   fprintf (stderr, "calibrate_input: end\n");
01246 #endif
01247   return;
01248 }
01249
01260 double
01261 calibrate_parse (unsigned int simulation, unsigned int experiment)
01262 {
01263   unsigned int i;
01264   double e;
01265   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01266     *buffer3, *buffer4;
01267   FILE *file_result;
01268
01269 #if DEBUG
01270   fprintf (stderr, "calibrate_parse: start\n");
01271   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01272            experiment);
01273 #endif
01274
01275   // Opening input files
01276   for (i = 0; i < calibrate->ninputs; ++i)
01277     {
01278       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01279 #if DEBUG
01280       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01281 #endif
01282       calibrate_input (simulation, &input[i][0],
01283                        calibrate->file[i][experiment]);
01284     }
01285   for (; i < MAX_NINPUTS; ++i)
01286     strcpy (&input[i][0], "");
01287 #if DEBUG
01288   fprintf (stderr, "calibrate_parse: parsing end\n");
01289 #endif
01290
01291   // Performing the simulation
01292   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01293   buffer2 = g_path_get_dirname (calibrate->simulator);
01294   buffer3 = g_path_get_basename (calibrate->simulator);
01295   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01296   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01297             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01298             input[6], input[7], output);
01299   g_free (buffer4);
01300   g_free (buffer3);
01301   g_free (buffer2);
01302 #if DEBUG
01303   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01304 #endif
01305   system (buffer);
01306
01307   // Checking the objective value function
```

```
01308   if (calibrate->evaluator)
01309     {
01310       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01311       buffer2 = g_path_get_dirname (calibrate->evaluator);
01312       buffer3 = g_path_get_basename (calibrate->evaluator);
01313       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01314       snprintf (buffer, 512, "\"%s\" %s %s %s",
01315                 buffer4, output, calibrate->experiment[experiment], result);
01316       g_free (buffer4);
01317       g_free (buffer3);
01318       g_free (buffer2);
01319 #if DEBUG
01320       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01321 #endif
01322       system (buffer);
01323       file_result = g_fopen (result, "r");
01324       e = atof (fgets (buffer, 512, file_result));
01325       fclose (file_result);
01326     }
01327   else
01328     {
01329       strcpy (result, "");
01330       file_result = g_fopen (output, "r");
01331       e = atof (fgets (buffer, 512, file_result));
01332       fclose (file_result);
01333     }
01334
01335   // Removing files
01336 #if !DEBUG
01337   for (i = 0; i < calibrate->ninputs; ++i)
01338     {
01339       if (calibrate->file[i][0])
01340         {
01341           snprintf (buffer, 512, RM " %s", &input[i][0]);
01342           system (buffer);
01343         }
01344     }
01345   snprintf (buffer, 512, RM " %s %s", output, result);
01346   system (buffer);
01347 #endif
01348
01349 #if DEBUG
01350   fprintf (stderr, "calibrate_parse: end\n");
01351 #endif
01352
01353   // Returning the objective function
01354   return e * calibrate->weight[experiment];
01355 }
01356
01361 void
01362 calibrate_print ()
01363 {
01364   unsigned int i;
01365   char buffer[512];
01366 #if HAVE_MPI
01367   if (calibrate->mpi_rank)
01368     return;
01369 #endif
01370   printf ("%s\n", gettext ("Best result"));
01371   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01372   printf ("error = %.15le\n", calibrate->error_old[0]);
01373   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
      error_old[0]);
01374   for (i = 0; i < calibrate->nvariables; ++i)
01375     {
01376       snprintf (buffer, 512, "%s = %s\n",
01377                 calibrate->label[i], format[calibrate->precision[i]]);
01378       printf (buffer, calibrate->value_old[i]);
01379       fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01380     }
01381   fflush (calibrate->file_result);
01382 }
01383
01392 void
01393 calibrate_save_variables (unsigned int simulation, double error)
01394 {
01395   unsigned int i;
01396   char buffer[64];
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_save_variables: start\n");
01399 #endif
01400   for (i = 0; i < calibrate->nvariables; ++i)
01401     {
01402       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01403       fprintf (calibrate->file_variables, buffer,
01404                calibrate->value[simulation * calibrate->nvariables + i]);
01405     }
```

```
01406   fprintf (calibrate->file_variables, "%.14le\n", error);
01407 #if DEBUG
01408   fprintf (stderr, "calibrate_save_variables: end\n");
01409 #endif
01410 }
01411
01420 void
01421 calibrate_best (unsigned int simulation, double value)
01422 {
01423   unsigned int i, j;
01424   double e;
01425 #if DEBUG
01426   fprintf (stderr, "calibrate_best: start\n");
01427   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01428            calibrate->nsaveds, calibrate->nbest);
01429 #endif
01430   if (calibrate->nsaveds < calibrate->nbest
01431       || value < calibrate->error_best[calibrate->nsaveds - 1])
01432     {
01433       if (calibrate->nsaveds < calibrate->nbest)
01434         ++calibrate->nsaveds;
01435       calibrate->error_best[calibrate->nsaveds - 1] = value;
01436       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01437       for (i = calibrate->nsaveds; --i;)
01438         {
01439           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01440             {
01441               j = calibrate->simulation_best[i];
01442               e = calibrate->error_best[i];
01443               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01444               calibrate->error_best[i] = calibrate->error_best[i - 1];
01445               calibrate->simulation_best[i - 1] = j;
01446               calibrate->error_best[i - 1] = e;
01447             }
01448           else
01449             break;
01450         }
01451     }
01452 #if DEBUG
01453   fprintf (stderr, "calibrate_best: end\n");
01454 #endif
01455 }
01456
01461 void
01462 calibrate_sequential ()
01463 {
01464   unsigned int i, j;
01465   double e;
01466 #if DEBUG
01467   fprintf (stderr, "calibrate_sequential: start\n");
01468   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01469            calibrate->nstart, calibrate->nend);
01470 #endif
01471   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01472     {
01473       e = 0.;
01474       for (j = 0; j < calibrate->nexperiments; ++j)
01475         e += calibrate_parse (i, j);
01476       calibrate_best (i, e);
01477       calibrate_save_variables (i, e);
01478 #if DEBUG
01479       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01480 #endif
01481     }
01482 #if DEBUG
01483   fprintf (stderr, "calibrate_sequential: end\n");
01484 #endif
01485 }
01486
01494 void *
01495 calibrate_thread (ParallelData * data)
01496 {
01497   unsigned int i, j, thread;
01498   double e;
01499 #if DEBUG
01500   fprintf (stderr, "calibrate_thread: start\n");
01501 #endif
01502   thread = data->thread;
01503 #if DEBUG
01504   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01505            calibrate->thread[thread], calibrate->thread[thread + 1]);
01506 #endif
01507   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01508     {
01509       e = 0.;
01510       for (j = 0; j < calibrate->nexperiments; ++j)
```

```
01511         e += calibrate_parse (i, j);
01512       g_mutex_lock (mutex);
01513       calibrate_best (i, e);
01514       calibrate_save_variables (i, e);
01515       g_mutex_unlock (mutex);
01516 #if DEBUG
01517       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01518 #endif
01519     }
01520 #if DEBUG
01521   fprintf (stderr, "calibrate_thread: end\n");
01522 #endif
01523   g_thread_exit (NULL);
01524   return NULL;
01525 }
01526
01538 void
01539 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01540                  double *error_best)
01541 {
01542   unsigned int i, j, k, s[calibrate->nbest];
01543   double e[calibrate->nbest];
01544 #if DEBUG
01545   fprintf (stderr, "calibrate_merge: start\n");
01546 #endif
01547   i = j = k = 0;
01548   do
01549     {
01550       if (i == calibrate->nsaveds)
01551         {
01552           s[k] = simulation_best[j];
01553           e[k] = error_best[j];
01554           ++j;
01555           ++k;
01556           if (j == nsaveds)
01557             break;
01558         }
01559       else if (j == nsaveds)
01560         {
01561           s[k] = calibrate->simulation_best[i];
01562           e[k] = calibrate->error_best[i];
01563           ++i;
01564           ++k;
01565           if (i == calibrate->nsaveds)
01566             break;
01567         }
01568       else if (calibrate->error_best[i] > error_best[j])
01569         {
01570           s[k] = simulation_best[j];
01571           e[k] = error_best[j];
01572           ++j;
01573           ++k;
01574         }
01575       else
01576         {
01577           s[k] = calibrate->simulation_best[i];
01578           e[k] = calibrate->error_best[i];
01579           ++i;
01580           ++k;
01581         }
01582     }
01583   while (k < calibrate->nbest);
01584   calibrate->nsaveds = k;
01585   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01586   memcpy (calibrate->error_best, e, k * sizeof (double));
01587 #if DEBUG
01588   fprintf (stderr, "calibrate_merge: end\n");
01589 #endif
01590 }
01591
01596 #if HAVE_MPI
01597 void
01598 calibrate_synchronise ()
01599 {
01600   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01601   double error_best[calibrate->nbest];
01602   MPI_Status mpi_stat;
01603 #if DEBUG
01604   fprintf (stderr, "calibrate_synchronise: start\n");
01605 #endif
01606   if (calibrate->mpi_rank == 0)
01607     {
01608       for (i = 1; i < ntasks; ++i)
01609         {
01610           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01611           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01612                     MPI_COMM_WORLD, &mpi_stat);
```

```
01613           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01614                     MPI_COMM_WORLD, &mpi_stat);
01615           calibrate_merge (nsaveds, simulation_best, error_best);
01616         }
01617     }
01618   else
01619     {
01620       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01621       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01622                 MPI_COMM_WORLD);
01623       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01624                 MPI_COMM_WORLD);
01625     }
01626 #if DEBUG
01627   fprintf (stderr, "calibrate_synchronise: end\n");
01628 #endif
01629 }
01630 #endif
01631
01636 void
01637 calibrate_sweep ()
01638 {
01639   unsigned int i, j, k, l;
01640   double e;
01641   GThread *thread[nthreads];
01642   ParallelData data[nthreads];
01643 #if DEBUG
01644   fprintf (stderr, "calibrate_sweep: start\n");
01645 #endif
01646   for (i = 0; i < calibrate->nsimulations; ++i)
01647     {
01648       k = i;
01649       for (j = 0; j < calibrate->nvariables; ++j)
01650         {
01651           l = k % calibrate->nsweeps[j];
01652           k /= calibrate->nsweeps[j];
01653           e = calibrate->rangemin[j];
01654           if (calibrate->nsweeps[j] > 1)
01655             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01656               / (calibrate->nsweeps[j] - 1);
01657           calibrate->value[i * calibrate->nvariables + j] = e;
01658         }
01659     }
01660   calibrate->nsaveds = 0;
01661   if (nthreads <= 1)
01662     calibrate_sequential ();
01663   else
01664     {
01665       for (i = 0; i < nthreads; ++i)
01666         {
01667           data[i].thread = i;
01668           thread[i]
01669             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01670         }
01671       for (i = 0; i < nthreads; ++i)
01672         g_thread_join (thread[i]);
01673     }
01674 #if HAVE_MPI
01675   // Communicating tasks results
01676   calibrate_synchronise ();
01677 #endif
01678 #if DEBUG
01679   fprintf (stderr, "calibrate_sweep: end\n");
01680 #endif
01681 }
01682
01687 void
01688 calibrate_MonteCarlo ()
01689 {
01690   unsigned int i, j;
01691   GThread *thread[nthreads];
01692   ParallelData data[nthreads];
01693 #if DEBUG
01694   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01695 #endif
01696   for (i = 0; i < calibrate->nsimulations; ++i)
01697     for (j = 0; j < calibrate->nvariables; ++j)
01698       calibrate->value[i * calibrate->nvariables + j]
01699         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01700         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01701   calibrate->nsaveds = 0;
01702   if (nthreads <= 1)
01703     calibrate_sequential ();
01704   else
01705     {
01706       for (i = 0; i < nthreads; ++i)
01707         {
```

```
01708           data[i].thread = i;
01709           thread[i]
01710             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01711         }
01712       for (i = 0; i < nthreads; ++i)
01713         g_thread_join (thread[i]);
01714     }
01715 #if HAVE_MPI
01716   // Communicating tasks results
01717   calibrate_synchronise ();
01718 #endif
01719 #if DEBUG
01720   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01721 #endif
01722 }
01723
01733 void
01734 calibrate_best_gradient (unsigned int simulation, double value)
01735 {
01736 #if DEBUG
01737   fprintf (stderr, "calibrate_best_gradient: start\n");
01738   fprintf (stderr,
01739            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01740            simulation, value, calibrate->error_best[0]);
01741 #endif
01742   if (value < calibrate->error_best[0])
01743     {
01744       calibrate->error_best[0] = value;
01745       calibrate->simulation_best[0] = simulation;
01746 #if DEBUG
01747       fprintf (stderr,
01748                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01749                simulation, value);
01750 #endif
01751     }
01752 #if DEBUG
01753   fprintf (stderr, "calibrate_best_gradient: end\n");
01754 #endif
01755 }
01756
01763 void
01764 calibrate_gradient_sequential (unsigned int simulation)
01765 {
01766   unsigned int i, j, k;
01767   double e;
01768 #if DEBUG
01769   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01770   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01771            "nend_gradient=%u\n",
01772            calibrate->nstart_gradient, calibrate->nend_gradient);
01773 #endif
01774   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01775     {
01776       k = simulation + i;
01777       e = 0.;
01778       for (j = 0; j < calibrate->nexperiments; ++j)
01779         e += calibrate_parse (k, j);
01780       calibrate_best_gradient (k, e);
01781       calibrate_save_variables (k, e);
01782 #if DEBUG
01783       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01784 #endif
01785     }
01786 #if DEBUG
01787   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01788 #endif
01789 }
01790
01798 void *
01799 calibrate_gradient_thread (ParallelData * data)
01800 {
01801   unsigned int i, j, thread;
01802   double e;
01803 #if DEBUG
01804   fprintf (stderr, "calibrate_gradient_thread: start\n");
01805 #endif
01806   thread = data->thread;
01807 #if DEBUG
01808   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01809            thread,
01810            calibrate->thread_gradient[thread],
01811            calibrate->thread_gradient[thread + 1]);
01812 #endif
01813   for (i = calibrate->thread_gradient[thread];
01814        i < calibrate->thread_gradient[thread + 1]; ++i)
01815     {
01816       e = 0.;
```

```
01817       for (j = 0; j < calibrate->nexperiments; ++j)
01818         e += calibrate_parse (i, j);
01819       g_mutex_lock (mutex);
01820       calibrate_best_gradient (i, e);
01821       calibrate_save_variables (i, e);
01822       g_mutex_unlock (mutex);
01823 #if DEBUG
01824       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01825 #endif
01826     }
01827 #if DEBUG
01828   fprintf (stderr, "calibrate_gradient_thread: end\n");
01829 #endif
01830   g_thread_exit (NULL);
01831   return NULL;
01832 }
01833
01843 double
01844 calibrate_estimate_gradient_random (unsigned int variable,
01845                                     unsigned int estimate)
01846 {
01847   double x;
01848 #if DEBUG
01849   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01850 #endif
01851   x = calibrate->gradient[variable]
01852     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
01853 #if DEBUG
01854   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01855            variable, x);
01856   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01857 #endif
01858   return x;
01859 }
01860
01870 double
01871 calibrate_estimate_gradient_coordinates (unsigned int variable,
01872                                          unsigned int estimate)
01873 {
01874   double x;
01875 #if DEBUG
01876   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01877 #endif
01878   x = calibrate->gradient[variable];
01879   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01880     {
01881       if (estimate & 1)
01882         x += calibrate->step[variable];
01883       else
01884         x -= calibrate->step[variable];
01885     }
01886 #if DEBUG
01887   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01888            variable, x);
01889   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01890 #endif
01891   return x;
01892 }
01893
01900 void
01901 calibrate_step_gradient (unsigned int simulation)
01902 {
01903   GThread *thread[nthreads_gradient];
01904   ParallelData data[nthreads_gradient];
01905   unsigned int i, j, k, b;
01906 #if DEBUG
01907   fprintf (stderr, "calibrate_step_gradient: start\n");
01908 #endif
01909   for (i = 0; i < calibrate->nestimates; ++i)
01910     {
01911       k = (simulation + i) * calibrate->nvariables;
01912       b = calibrate->simulation_best[0] * calibrate->nvariables;
01913 #if DEBUG
01914       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01915                simulation + i, calibrate->simulation_best[0]);
01916 #endif
01917       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01918         {
01919 #if DEBUG
01920           fprintf (stderr,
01921                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01922                    i, j, calibrate->value[b]);
01923 #endif
01924           calibrate->value[k]
01925             = calibrate->value[b] + calibrate_estimate_gradient (j, i);
01926           calibrate->value[k] = fmin (fmax (calibrate->value[k],
01927                                             calibrate->rangeminabs[j]),
```

```
01928                                       calibrate->rangemaxabs[j]);
01929 #if DEBUG
01930           fprintf (stderr,
01931                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01932                    i, j, calibrate->value[k]);
01933 #endif
01934         }
01935     }
01936   if (nthreads_gradient == 1)
01937     calibrate_gradient_sequential (simulation);
01938   else
01939     {
01940       for (i = 0; i <= nthreads_gradient; ++i)
01941         {
01942           calibrate->thread_gradient[i]
01943             = simulation + calibrate->nstart_gradient
01944             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01945             / nthreads_gradient;
01946 #if DEBUG
01947           fprintf (stderr,
01948                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01949                    i, calibrate->thread_gradient[i]);
01950 #endif
01951         }
01952       for (i = 0; i < nthreads_gradient; ++i)
01953         {
01954           data[i].thread = i;
01955           thread[i] = g_thread_new
01956             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01957         }
01958       for (i = 0; i < nthreads_gradient; ++i)
01959         g_thread_join (thread[i]);
01960     }
01961 #if DEBUG
01962   fprintf (stderr, "calibrate_step_gradient: end\n");
01963 #endif
01964 }
01965
01970 void
01971 calibrate_gradient ()
01972 {
01973   unsigned int i, j, k, b, s, adjust;
01974 #if DEBUG
01975   fprintf (stderr, "calibrate_gradient: start\n");
01976 #endif
01977   for (i = 0; i < calibrate->nvariables; ++i)
01978     calibrate->gradient[i] = 0.;
01979   b = calibrate->simulation_best[0] * calibrate->nvariables;
01980   s = calibrate->nsimulations;
01981   adjust = 1;
01982   for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
01983     {
01984 #if DEBUG
01985       fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
01986                i, calibrate->simulation_best[0]);
01987 #endif
01988       calibrate_step_gradient (s);
01989       k = calibrate->simulation_best[0] * calibrate->nvariables;
01990 #if DEBUG
01991       fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
01992                i, calibrate->simulation_best[0]);
01993 #endif
01994       if (k == b)
01995         {
01996           if (adjust)
01997             for (j = 0; j < calibrate->nvariables; ++j)
01998               calibrate->step[j] *= 0.5;
01999           for (j = 0; j < calibrate->nvariables; ++j)
02000             calibrate->gradient[j] = 0.;
02001           adjust = 1;
02002         }
02003       else
02004         {
02005           for (j = 0; j < calibrate->nvariables; ++j)
02006             {
02007 #if DEBUG
02008               fprintf (stderr,
02009                        "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02010                        j, calibrate->value[k + j], j, calibrate->value[b + j]);
02011 #endif
02012               calibrate->gradient[j]
02013                 = (1. - calibrate->relaxation) * calibrate->gradient[j]
02014                 + calibrate->relaxation
02015                 * (calibrate->value[k + j] - calibrate->value[b + j]);
02016 #if DEBUG
02017               fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
```

```
02018                         j, calibrate->gradient[j]);
02019 #endif
02020             }
02021           adjust = 0;
02022         }
02023     }
02024 #if DEBUG
02025   fprintf (stderr, "calibrate_gradient: end\n");
02026 #endif
02027 }
02028
02036 double
02037 calibrate_genetic_objective (Entity * entity)
02038 {
02039   unsigned int j;
02040   double objective;
02041   char buffer[64];
02042 #if DEBUG
02043   fprintf (stderr, "calibrate_genetic_objective: start\n");
02044 #endif
02045   for (j = 0; j < calibrate->nvariables; ++j)
02046     {
02047       calibrate->value[entity->id * calibrate->nvariables + j]
02048         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02049     }
02050   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02051     objective += calibrate_parse (entity->id, j);
02052   g_mutex_lock (mutex);
02053   for (j = 0; j < calibrate->nvariables; ++j)
02054     {
02055       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02056       fprintf (calibrate->file_variables, buffer,
02057                genetic_get_variable (entity, calibrate->genetic_variable + j));
02058     }
02059   fprintf (calibrate->file_variables, "%.14le\n", objective);
02060   g_mutex_unlock (mutex);
02061 #if DEBUG
02062   fprintf (stderr, "calibrate_genetic_objective: end\n");
02063 #endif
02064   return objective;
02065 }
02066
02071 void
02072 calibrate_genetic ()
02073 {
02074   char *best_genome;
02075   double best_objective, *best_variable;
02076 #if DEBUG
02077   fprintf (stderr, "calibrate_genetic: start\n");
02078   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02079            nthreads);
02080   fprintf (stderr,
02081            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02082            calibrate->nvariables, calibrate->nsimulations,
02083            calibrate->niterations);
02084   fprintf (stderr,
02085            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02086            calibrate->mutation_ratio, calibrate->
    reproduction_ratio,
02087            calibrate->adaptation_ratio);
02088 #endif
02089   genetic_algorithm_default (calibrate->nvariables,
02090                              calibrate->genetic_variable,
02091                              calibrate->nsimulations,
02092                              calibrate->niterations,
02093                              calibrate->mutation_ratio,
02094                              calibrate->reproduction_ratio,
02095                              calibrate->adaptation_ratio,
02096                              &calibrate_genetic_objective,
02097                              &best_genome, &best_variable, &best_objective);
02098 #if DEBUG
02099   fprintf (stderr, "calibrate_genetic: the best\n");
02100 #endif
02101   calibrate->error_old = (double *) g_malloc (sizeof (double));
02102   calibrate->value_old
02103     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02104   calibrate->error_old[0] = best_objective;
02105   memcpy (calibrate->value_old, best_variable,
02106           calibrate->nvariables * sizeof (double));
02107   g_free (best_genome);
02108   g_free (best_variable);
02109   calibrate_print ();
02110 #if DEBUG
02111   fprintf (stderr, "calibrate_genetic: end\n");
02112 #endif
02113 }
02114
```

```
02119 void
02120 calibrate_save_old ()
02121 {
02122   unsigned int i, j;
02123 #if DEBUG
02124   fprintf (stderr, "calibrate_save_old: start\n");
02125   fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02126 #endif
02127   memcpy (calibrate->error_old, calibrate->error_best,
02128           calibrate->nbest * sizeof (double));
02129   for (i = 0; i < calibrate->nbest; ++i)
02130     {
02131       j = calibrate->simulation_best[i];
02132 #if DEBUG
02133       fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02134 #endif
02135       memcpy (calibrate->value_old + i * calibrate->nvariables,
02136               calibrate->value + j * calibrate->nvariables,
02137               calibrate->nvariables * sizeof (double));
02138     }
02139 #if DEBUG
02140   for (i = 0; i < calibrate->nvariables; ++i)
02141     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02142              i, calibrate->value_old[i]);
02143   fprintf (stderr, "calibrate_save_old: end\n");
02144 #endif
02145 }
02146
02152 void
02153 calibrate_merge_old ()
02154 {
02155   unsigned int i, j, k;
02156   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
     nbest],
02157     *enew, *eold;
02158 #if DEBUG
02159   fprintf (stderr, "calibrate_merge_old: start\n");
02160 #endif
02161   enew = calibrate->error_best;
02162   eold = calibrate->error_old;
02163   i = j = k = 0;
02164   do
02165     {
02166       if (*enew < *eold)
02167         {
02168           memcpy (v + k * calibrate->nvariables,
02169                   calibrate->value
02170                   + calibrate->simulation_best[i] * calibrate->
     nvariables,
02171                   calibrate->nvariables * sizeof (double));
02172           e[k] = *enew;
02173           ++k;
02174           ++enew;
02175           ++i;
02176         }
02177       else
02178         {
02179           memcpy (v + k * calibrate->nvariables,
02180                   calibrate->value_old + j * calibrate->nvariables,
02181                   calibrate->nvariables * sizeof (double));
02182           e[k] = *eold;
02183           ++k;
02184           ++eold;
02185           ++j;
02186         }
02187     }
02188   while (k < calibrate->nbest);
02189   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02190   memcpy (calibrate->error_old, e, k * sizeof (double));
02191 #if DEBUG
02192   fprintf (stderr, "calibrate_merge_old: end\n");
02193 #endif
02194 }
02195
02201 void
02202 calibrate_refine ()
02203 {
02204   unsigned int i, j;
02205   double d;
02206 #if HAVE_MPI
02207   MPI_Status mpi_stat;
02208 #endif
02209 #if DEBUG
02210   fprintf (stderr, "calibrate_refine: start\n");
02211 #endif
02212 #if HAVE_MPI
02213   if (!calibrate->mpi_rank)
```

```
02214     {
02215 #endif
02216       for (j = 0; j < calibrate->nvariables; ++j)
02217         {
02218           calibrate->rangemin[j] = calibrate->rangemax[j]
02219             = calibrate->value_old[j];
02220         }
02221       for (i = 0; ++i < calibrate->nbest;)
02222         {
02223           for (j = 0; j < calibrate->nvariables; ++j)
02224             {
02225               calibrate->rangemin[j]
02226                 = fmin (calibrate->rangemin[j],
02227                         calibrate->value_old[i * calibrate->nvariables + j]);
02228               calibrate->rangemax[j]
02229                 = fmax (calibrate->rangemax[j],
02230                         calibrate->value_old[i * calibrate->nvariables + j]);
02231             }
02232         }
02233       for (j = 0; j < calibrate->nvariables; ++j)
02234         {
02235           d = calibrate->tolerance
02236             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02237           switch (calibrate->algorithm)
02238             {
02239             case ALGORITHM_MONTE_CARLO:
02240               d *= 0.5;
02241               break;
02242             default:
02243               if (calibrate->nsweeps[j] > 1)
02244                 d /= calibrate->nsweeps[j] - 1;
02245               else
02246                 d = 0.;
02247             }
02248           calibrate->rangemin[j] -= d;
02249           calibrate->rangemin[j]
02250             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02251           calibrate->rangemax[j] += d;
02252           calibrate->rangemax[j]
02253             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02254           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02255                   calibrate->rangemin[j], calibrate->rangemax[j]);
02256           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02257                    calibrate->label[j], calibrate->rangemin[j],
02258                    calibrate->rangemax[j]);
02259         }
02260 #if HAVE_MPI
02261       for (i = 1; i < ntasks; ++i)
02262         {
02263           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02264                     1, MPI_COMM_WORLD);
02265           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02266                     1, MPI_COMM_WORLD);
02267         }
02268     }
02269   else
02270     {
02271       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02272                 MPI_COMM_WORLD, &mpi_stat);
02273       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02274                 MPI_COMM_WORLD, &mpi_stat);
02275     }
02276 #endif
02277 #if DEBUG
02278   fprintf (stderr, "calibrate_refine: end\n");
02279 #endif
02280 }
02281
02286 void
02287 calibrate_step ()
02288 {
02289 #if DEBUG
02290   fprintf (stderr, "calibrate_step: start\n");
02291 #endif
02292   calibrate_algorithm ();
02293   if (calibrate->nsteps)
02294     calibrate_gradient ();
02295 #if DEBUG
02296   fprintf (stderr, "calibrate_step: end\n");
02297 #endif
02298 }
02299
02304 void
02305 calibrate_iterate ()
02306 {
02307   unsigned int i;
02308 #if DEBUG
```

```
02309   fprintf (stderr, "calibrate_iterate: start\n");
02310 #endif
02311   calibrate->error_old
02312     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02313   calibrate->value_old = (double *)
02314     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02315   calibrate_step ();
02316   calibrate_save_old ();
02317   calibrate_refine ();
02318   calibrate_print ();
02319   for (i = 1; i < calibrate->niterations; ++i)
02320     {
02321       calibrate_step ();
02322       calibrate_merge_old ();
02323       calibrate_refine ();
02324       calibrate_print ();
02325     }
02326 #if DEBUG
02327   fprintf (stderr, "calibrate_iterate: end\n");
02328 #endif
02329 }
02330
02335 void
02336 calibrate_free ()
02337 {
02338   unsigned int i, j;
02339 #if DEBUG
02340   fprintf (stderr, "calibrate_free: start\n");
02341 #endif
02342   for (j = 0; j < calibrate->ninputs; ++j)
02343     {
02344       for (i = 0; i < calibrate->nexperiments; ++i)
02345         g_mapped_file_unref (calibrate->file[j][i]);
02346       g_free (calibrate->file[j]);
02347     }
02348   g_free (calibrate->error_old);
02349   g_free (calibrate->value_old);
02350   g_free (calibrate->value);
02351   g_free (calibrate->genetic_variable);
02352   g_free (calibrate->rangemax);
02353   g_free (calibrate->rangemin);
02354 #if DEBUG
02355   fprintf (stderr, "calibrate_free: end\n");
02356 #endif
02357 }
02358
02363 void
02364 calibrate_open ()
02365 {
02366   GTimeZone *tz;
02367   GDateTime *t0, *t;
02368   unsigned int i, j, *nbits;
02369
02370 #if DEBUG
02371   char *buffer;
02372   fprintf (stderr, "calibrate_open: start\n");
02373 #endif
02374
02375   // Getting initial time
02376 #if DEBUG
02377   fprintf (stderr, "calibrate_open: getting initial time\n");
02378 #endif
02379   tz = g_time_zone_new_utc ();
02380   t0 = g_date_time_new_now (tz);
02381
02382   // Obtaining and initing the pseudo-random numbers generator seed
02383 #if DEBUG
02384   fprintf (stderr, "calibrate_open: getting initial seed\n");
02385 #endif
02386   calibrate->seed = input->seed;
02387   gsl_rng_set (calibrate->rng, calibrate->seed);
02388
02389   // Replacing the working directory
02390 #if DEBUG
02391   fprintf (stderr, "calibrate_open: replacing the working directory\n");
02392 #endif
02393   g_chdir (input->directory);
02394
02395   // Getting results file names
02396   calibrate->result = input->result;
02397   calibrate->variables = input->variables;
02398
02399   // Obtaining the simulator file
02400   calibrate->simulator = input->simulator;
02401
02402   // Obtaining the evaluator file
02403   calibrate->evaluator = input->evaluator;
```

```
02404
02405   // Reading the algorithm
02406   calibrate->algorithm = input->algorithm;
02407   switch (calibrate->algorithm)
02408     {
02409     case ALGORITHM_MONTE_CARLO:
02410       calibrate_algorithm = calibrate_MonteCarlo;
02411       break;
02412     case ALGORITHM_SWEEP:
02413       calibrate_algorithm = calibrate_sweep;
02414       break;
02415     default:
02416       calibrate_algorithm = calibrate_genetic;
02417       calibrate->mutation_ratio = input->mutation_ratio;
02418       calibrate->reproduction_ratio = input->
      reproduction_ratio;
02419       calibrate->adaptation_ratio = input->adaptation_ratio;
02420     }
02421   calibrate->nvariables = input->nvariables;
02422   calibrate->nsimulations = input->nsimulations;
02423   calibrate->niterations = input->niterations;
02424   calibrate->nbest = input->nbest;
02425   calibrate->tolerance = input->tolerance;
02426   calibrate->nsteps = input->nsteps;
02427   calibrate->nestimates = 0;
02428   if (input->nsteps)
02429     {
02430       calibrate->gradient_method = input->gradient_method;
02431       calibrate->relaxation = input->relaxation;
02432       switch (input->gradient_method)
02433         {
02434         case GRADIENT_METHOD_COORDINATES:
02435           calibrate->nestimates = 2 * calibrate->nvariables;
02436           calibrate_estimate_gradient =
      calibrate_estimate_gradient_coordinates;
02437           break;
02438         default:
02439           calibrate->nestimates = input->nestimates;
02440           calibrate_estimate_gradient =
      calibrate_estimate_gradient_random;
02441         }
02442     }
02443
02444 #if DEBUG
02445   fprintf (stderr, "calibrate_open: nbest=%u\n", calibrate->nbest);
02446 #endif
02447   calibrate->simulation_best
02448     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02449   calibrate->error_best
02450     = (double *) alloca (calibrate->nbest * sizeof (double));
02451
02452   // Reading the experimental data
02453 #if DEBUG
02454   buffer = g_get_current_dir ();
02455   fprintf (stderr, "calibrate_open: current directory=%s\n", buffer);
02456   g_free (buffer);
02457 #endif
02458   calibrate->nexperiments = input->nexperiments;
02459   calibrate->ninputs = input->ninputs;
02460   calibrate->experiment = input->experiment;
02461   calibrate->weight = input->weight;
02462   for (i = 0; i < input->ninputs; ++i)
02463     {
02464       calibrate->template[i] = input->template[i];
02465       calibrate->file[i]
02466         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02467     }
02468   for (i = 0; i < input->nexperiments; ++i)
02469     {
02470 #if DEBUG
02471       fprintf (stderr, "calibrate_open: i=%u\n", i);
02472       fprintf (stderr, "calibrate_open: experiment=%s\n",
02473                calibrate->experiment[i]);
02474       fprintf (stderr, "calibrate_open: weight=%lg\n", calibrate->weight[i]);
02475 #endif
02476       for (j = 0; j < input->ninputs; ++j)
02477         {
02478 #if DEBUG
02479           fprintf (stderr, "calibrate_open: template%u\n", j + 1);
02480           fprintf (stderr, "calibrate_open: experiment=%u template%u=%s\n",
02481                    i, j + 1, calibrate->template[j][i]);
02482 #endif
02483           calibrate->file[j][i]
02484             = g_mapped_file_new (input->template[j][i], 0, NULL);
02485         }
02486     }
02487
```

```
02488   // Reading the variables data
02489 #if DEBUG
02490   fprintf (stderr, "calibrate_open: reading variables\n");
02491 #endif
02492   calibrate->label = input->label;
02493   j = input->nvariables * sizeof (double);
02494   calibrate->rangemin = (double *) g_malloc (j);
02495   calibrate->rangemax = (double *) g_malloc (j);
02496   memcpy (calibrate->rangemin, input->rangemin, j);
02497   memcpy (calibrate->rangemax, input->rangemax, j);
02498   calibrate->rangeminabs = input->rangeminabs;
02499   calibrate->rangemaxabs = input->rangemaxabs;
02500   calibrate->precision = input->precision;
02501   calibrate->nsweeps = input->nsweeps;
02502   calibrate->step = input->step;
02503   nbits = input->nbits;
02504   if (input->algorithm == ALGORITHM_SWEEP)
02505     {
02506       calibrate->nsimulations = 1;
02507       for (i = 0; i < input->nvariables; ++i)
02508         {
02509           if (input->algorithm == ALGORITHM_SWEEP)
02510             {
02511               calibrate->nsimulations *= input->nsweeps[i];
02512 #if DEBUG
02513               fprintf (stderr, "calibrate_open: nsweeps=%u nsimulations=%u\n",
02514                        calibrate->nsweeps[i], calibrate->nsimulations);
02515 #endif
02516             }
02517         }
02518     }
02519   if (calibrate->nsteps)
02520     calibrate->gradient
02521       = (double *) alloca (calibrate->nvariables * sizeof (double));
02522
02523   // Allocating values
02524 #if DEBUG
02525   fprintf (stderr, "calibrate_open: allocating variables\n");
02526   fprintf (stderr, "calibrate_open: nvariables=%u\n", calibrate->nvariables);
02527 #endif
02528   calibrate->genetic_variable = NULL;
02529   if (calibrate->algorithm == ALGORITHM_GENETIC)
02530     {
02531       calibrate->genetic_variable = (GeneticVariable *)
02532         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02533       for (i = 0; i < calibrate->nvariables; ++i)
02534         {
02535 #if DEBUG
02536           fprintf (stderr, "calibrate_open: i=%u min=%lg max=%lg nbits=%u\n",
02537                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02538 #endif
02539           calibrate->genetic_variable[i].minimum = calibrate->
    rangemin[i];
02540           calibrate->genetic_variable[i].maximum = calibrate->
    rangemax[i];
02541           calibrate->genetic_variable[i].nbits = nbits[i];
02542         }
02543     }
02544 #if DEBUG
02545   fprintf (stderr, "calibrate_open: nvariables=%u nsimulations=%u\n",
02546            calibrate->nvariables, calibrate->nsimulations);
02547 #endif
02548   calibrate->value = (double *)
02549     g_malloc ((calibrate->nsimulations
02550                + calibrate->nestimates * calibrate->nsteps)
02551               * calibrate->nvariables * sizeof (double));
02552
02553   // Calculating simulations to perform on each task
02554 #if HAVE_MPI
02555 #if DEBUG
02556   fprintf (stderr, "calibrate_open: rank=%u ntasks=%u\n",
02557            calibrate->mpi_rank, ntasks);
02558 #endif
02559   calibrate->nstart = calibrate->mpi_rank * calibrate->
    nsimulations / ntasks;
02560   calibrate->nend
02561     = (1 + calibrate->mpi_rank) * calibrate->nsimulations /
    ntasks;
02562   if (calibrate->nsteps)
02563     {
02564       calibrate->nstart_gradient
02565         = calibrate->mpi_rank * calibrate->nestimates / ntasks;
02566       calibrate->nend_gradient
02567         = (1 + calibrate->mpi_rank) * calibrate->nestimates /
    ntasks;
02568     }
02569 #else
```

```
02570   calibrate->nstart = 0;
02571   calibrate->nend = calibrate->nsimulations;
02572   if (calibrate->nsteps)
02573     {
02574       calibrate->nstart_gradient = 0;
02575       calibrate->nend_gradient = calibrate->nestimates;
02576     }
02577 #endif
02578 #if DEBUG
02579   fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02580            calibrate->nend);
02581 #endif
02582
02583   // Calculating simulations to perform for each thread
02584   calibrate->thread
02585     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02586   for (i = 0; i <= nthreads; ++i)
02587     {
02588       calibrate->thread[i] = calibrate->nstart
02589         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02590 #if DEBUG
02591       fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02592                calibrate->thread[i]);
02593 #endif
02594     }
02595   if (calibrate->nsteps)
02596     calibrate->thread_gradient = (unsigned int *)
02597       alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02598
02599   // Opening result files
02600   calibrate->file_result = g_fopen (calibrate->result, "w");
02601   calibrate->file_variables = g_fopen (calibrate->variables, "w");
02602
02603   // Performing the algorithm
02604   switch (calibrate->algorithm)
02605     {
02606       // Genetic algorithm
02607     case ALGORITHM_GENETIC:
02608       calibrate_genetic ();
02609       break;
02610
02611       // Iterative algorithm
02612     default:
02613       calibrate_iterate ();
02614     }
02615
02616   // Getting calculation time
02617   t = g_date_time_new_now (tz);
02618   calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02619   g_date_time_unref (t);
02620   g_date_time_unref (t0);
02621   g_time_zone_unref (tz);
02622   printf ("%s = %.6lg s\n",
02623           gettext ("Calculation time"), calibrate->calculation_time);
02624   fprintf (calibrate->file_result, "%s = %.6lg s\n",
02625            gettext ("Calculation time"), calibrate->calculation_time);
02626
02627   // Closing result files
02628   fclose (calibrate->file_variables);
02629   fclose (calibrate->file_result);
02630
02631 #if DEBUG
02632   fprintf (stderr, "calibrate_open: end\n");
02633 #endif
02634 }
02635
02636 #if HAVE_GTK
02637
02644 void
02645 input_save_gradient (xmlNode * node)
02646 {
02647 #if DEBUG
02648   fprintf (stderr, "input_save_gradient: start\n");
02649 #endif
02650   if (input->nsteps)
02651     {
02652       xml_node_set_uint (node, XML_NSTEPS, input->
     nsteps);
02653       if (input->relaxation != DEFAULT_RELAXATION)
02654         xml_node_set_float (node, XML_RELAXATION, input->
     relaxation);
02655       switch (input->gradient_method)
02656         {
02657         case GRADIENT_METHOD_COORDINATES:
02658           xmlSetProp (node, XML_GRADIENT_METHOD,
     XML_COORDINATES);
02659           break;
```

```
02660         default:
02661           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02662           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
02663         }
02664     }
02665 #if DEBUG
02666   fprintf (stderr, "input_save_gradient: end\n");
02667 #endif
02668 }
02669
02676 void
02677 input_save (char *filename)
02678 {
02679   unsigned int i, j;
02680   char *buffer;
02681   xmlDoc *doc;
02682   xmlNode *node, *child;
02683   GFile *file, *file2;
02684
02685 #if DEBUG
02686   fprintf (stderr, "input_save: start\n");
02687 #endif
02688
02689   // Getting the input file directory
02690   input->name = g_path_get_basename (filename);
02691   input->directory = g_path_get_dirname (filename);
02692   file = g_file_new_for_path (input->directory);
02693
02694   // Opening the input file
02695   doc = xmlNewDoc ((const xmlChar *) "1.0");
02696
02697   // Setting root XML node
02698   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02699   xmlDocSetRootElement (doc, node);
02700
02701   // Adding properties to the root XML node
02702   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02703     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02704   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02705     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02706   file2 = g_file_new_for_path (input->simulator);
02707   buffer = g_file_get_relative_path (file, file2);
02708   g_object_unref (file2);
02709   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02710   g_free (buffer);
02711   if (input->evaluator)
02712     {
02713       file2 = g_file_new_for_path (input->evaluator);
02714       buffer = g_file_get_relative_path (file, file2);
02715       g_object_unref (file2);
02716       if (xmlStrlen ((xmlChar *) buffer))
02717         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02718       g_free (buffer);
02719     }
02720   if (input->seed != DEFAULT_RANDOM_SEED)
02721     xml_node_set_uint (node, XML_SEED, input->seed);
02722
02723   // Setting the algorithm
02724   buffer = (char *) g_malloc (64);
02725   switch (input->algorithm)
02726     {
02727     case ALGORITHM_MONTE_CARLO:
02728       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02729       snprintf (buffer, 64, "%u", input->nsimulations);
02730       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02731       snprintf (buffer, 64, "%u", input->niterations);
02732       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02733       snprintf (buffer, 64, "%.31g", input->tolerance);
02734       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02735       snprintf (buffer, 64, "%u", input->nbest);
02736       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02737       input_save_gradient (node);
02738       break;
02739     case ALGORITHM_SWEEP:
02740       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02741       snprintf (buffer, 64, "%u", input->niterations);
02742       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02743       snprintf (buffer, 64, "%.31g", input->tolerance);
02744       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02745       snprintf (buffer, 64, "%u", input->nbest);
02746       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02747       input_save_gradient (node);
02748       break;
02749     default:
02750       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02751       snprintf (buffer, 64, "%u", input->nsimulations);
```

```
02752       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02753       snprintf (buffer, 64, "%u", input->niterations);
02754       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02755       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02756       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02757       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02758       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02759       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02760       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02761       break;
02762     }
02763   g_free (buffer);
02764
02765   // Setting the experimental data
02766   for (i = 0; i < input->nexperiments; ++i)
02767     {
02768       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02769       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02770       if (input->weight[i] != 1.)
02771         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02772       for (j = 0; j < input->ninputs; ++j)
02773         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02774     }
02775
02776   // Setting the variables data
02777   for (i = 0; i < input->nvariables; ++i)
02778     {
02779       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02780       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02781       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02782       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02783         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02784       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02785       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02786         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02787       if (input->precision[i] != DEFAULT_PRECISION)
02788         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02789       if (input->algorithm == ALGORITHM_SWEEP)
02790         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02791       else if (input->algorithm == ALGORITHM_GENETIC)
02792         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02793       if (input->nsteps)
02794         xml_node_set_float (child, XML_STEP, input->
      step[i]);
02795     }
02796
02797   // Saving the XML file
02798   xmlSaveFormatFile (filename, doc, 1);
02799
02800   // Freeing memory
02801   xmlFreeDoc (doc);
02802
02803 #if DEBUG
02804   fprintf (stderr, "input_save: end\n");
02805 #endif
02806 }
02807
02812 void
02813 options_new ()
02814 {
02815 #if DEBUG
02816   fprintf (stderr, "options_new: start\n");
02817 #endif
02818   options->label_seed = (GtkLabel *)
02819     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02820   options->spin_seed = (GtkSpinButton *)
02821     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02822   gtk_widget_set_tooltip_text
02823     (GTK_WIDGET (options->spin_seed),
02824      gettext ("Seed to init the pseudo-random numbers generator"));
02825   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02826   options->label_threads = (GtkLabel *)
02827     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02828   options->spin_threads
02829     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02830   gtk_widget_set_tooltip_text
02831     (GTK_WIDGET (options->spin_threads),
02832      gettext ("Number of threads to perform the calibration/optimization for "
02833               "the stochastic algorithm"));
```

```
02834   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
02835   options->label_gradient = (GtkLabel *)
02836     gtk_label_new (gettext ("Threads number for the gradient based method"));
02837   options->spin_gradient
02838     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02839   gtk_widget_set_tooltip_text
02840     (GTK_WIDGET (options->spin_gradient),
02841      gettext ("Number of threads to perform the calibration/optimization for "
02842               "the gradient based method"));
02843   gtk_spin_button_set_value (options->spin_gradient,
02844                              (gdouble) nthreads_gradient);
02845   options->grid = (GtkGrid *) gtk_grid_new ();
02846   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
02847   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
02848   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
02849                    0, 1, 1, 1);
02850   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
02851                    1, 1, 1, 1);
02852   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
02853                    0, 2, 1, 1);
02854   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
02855                    1, 2, 1, 1);
02856   gtk_widget_show_all (GTK_WIDGET (options->grid));
02857   options->dialog = (GtkDialog *)
02858     gtk_dialog_new_with_buttons (gettext ("Options"),
02859                                  window->window,
02860                                  GTK_DIALOG_MODAL,
02861                                  gettext ("_OK"), GTK_RESPONSE_OK,
02862                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02863                                  NULL);
02864   gtk_container_add
02865     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02866      GTK_WIDGET (options->grid));
02867   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02868     {
02869       input->seed
02870         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02871       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
02872       nthreads_gradient
02873         = gtk_spin_button_get_value_as_int (options->spin_gradient);
02874     }
02875   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02876 #if DEBUG
02877   fprintf (stderr, "options_new: end\n");
02878 #endif
02879 }
02880
02885 void
02886 running_new ()
02887 {
02888 #if DEBUG
02889   fprintf (stderr, "running_new: start\n");
02890 #endif
02891   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02892   running->dialog = (GtkDialog *)
02893     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02894                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02895   gtk_container_add
02896     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02897      GTK_WIDGET (running->label));
02898   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02899 #if DEBUG
02900   fprintf (stderr, "running_new: end\n");
02901 #endif
02902 }
02903
02909 int
02910 window_get_algorithm ()
02911 {
02912   unsigned int i;
02913 #if DEBUG
02914   fprintf (stderr, "window_get_algorithm: start\n");
02915 #endif
02916   for (i = 0; i < NALGORITHMS; ++i)
02917     if (gtk_toggle_button_get_active
02918         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02919       break;
02920 #if DEBUG
02921   fprintf (stderr, "window_get_algorithm: %u\n", i);
02922   fprintf (stderr, "window_get_algorithm: end\n");
02923 #endif
02924   return i;
02925 }
02926
02932 int
02933 window_get_gradient ()
```

```
02934 {
02935   unsigned int i;
02936 #if DEBUG
02937   fprintf (stderr, "window_get_gradient: start\n");
02938 #endif
02939   for (i = 0; i < NGRADIENTS; ++i)
02940     if (gtk_toggle_button_get_active
02941         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02942       break;
02943 #if DEBUG
02944   fprintf (stderr, "window_get_gradient: %u\n", i);
02945   fprintf (stderr, "window_get_gradient: end\n");
02946 #endif
02947   return i;
02948 }
02949
02954 void
02955 window_save_gradient ()
02956 {
02957 #if DEBUG
02958   fprintf (stderr, "window_save_gradient: start\n");
02959 #endif
02960   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02961     {
02962       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02963       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
02964       switch (window_get_gradient ())
02965         {
02966         case GRADIENT_METHOD_COORDINATES:
02967           input->gradient_method = GRADIENT_METHOD_COORDINATES;
02968           break;
02969         default:
02970           input->gradient_method = GRADIENT_METHOD_RANDOM;
02971           input->nestimates
02972             = gtk_spin_button_get_value_as_int (window->spin_estimates);
02973         }
02974     }
02975   else
02976     input->nsteps = 0;
02977 #if DEBUG
02978   fprintf (stderr, "window_save_gradient: end\n");
02979 #endif
02980 }
02981
02987 int
02988 window_save ()
02989 {
02990   char *buffer;
02991   GtkFileChooserDialog *dlg;
02992
02993 #if DEBUG
02994   fprintf (stderr, "window_save: start\n");
02995 #endif
02996
02997   // Opening the saving dialog
02998   dlg = (GtkFileChooserDialog *)
02999     gtk_file_chooser_dialog_new (gettext ("Save file"),
03000                                  window->window,
03001                                  GTK_FILE_CHOOSER_ACTION_SAVE,
03002                                  gettext ("_Cancel"),
03003                                  GTK_RESPONSE_CANCEL,
03004                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03005   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
03006   buffer = g_build_filename (input->directory, input->name, NULL);
03007   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
03008   g_free (buffer);
03009
03010   // If OK response then saving
03011   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03012     {
03013
03014       // Adding properties to the root XML node
03015       input->simulator = gtk_file_chooser_get_filename
03016         (GTK_FILE_CHOOSER (window->button_simulator));
03017       if (gtk_toggle_button_get_active
03018           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
03019         input->evaluator = gtk_file_chooser_get_filename
03020           (GTK_FILE_CHOOSER (window->button_evaluator));
03021       else
03022         input->evaluator = NULL;
03023       input->result
03024         = (char *) xmlStrdup ((const xmlChar *)
03025                               gtk_entry_get_text (window->entry_result));
03026       input->variables
03027         = (char *) xmlStrdup ((const xmlChar *)
03028                               gtk_entry_get_text (window->entry_variables));
```

```
03029
03030       // Setting the algorithm
03031       switch (window_get_algorithm ())
03032         {
03033         case ALGORITHM_MONTE_CARLO:
03034           input->algorithm = ALGORITHM_MONTE_CARLO;
03035           input->nsimulations
03036             = gtk_spin_button_get_value_as_int (window->spin_simulations);
03037           input->niterations
03038             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03039           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03040           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03041           window_save_gradient ();
03042           break;
03043         case ALGORITHM_SWEEP:
03044           input->algorithm = ALGORITHM_SWEEP;
03045           input->niterations
03046             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03047           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03048           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03049           window_save_gradient ();
03050           break;
03051         default:
03052           input->algorithm = ALGORITHM_GENETIC;
03053           input->nsimulations
03054             = gtk_spin_button_get_value_as_int (window->spin_population);
03055           input->niterations
03056             = gtk_spin_button_get_value_as_int (window->spin_generations);
03057           input->mutation_ratio
03058             = gtk_spin_button_get_value (window->spin_mutation);
03059           input->reproduction_ratio
03060             = gtk_spin_button_get_value (window->spin_reproduction);
03061           input->adaptation_ratio
03062             = gtk_spin_button_get_value (window->spin_adaptation);
03063           break;
03064         }
03065
03066       // Saving the XML file
03067       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03068       input_save (buffer);
03069
03070       // Closing and freeing memory
03071       g_free (buffer);
03072       gtk_widget_destroy (GTK_WIDGET (dlg));
03073 #if DEBUG
03074       fprintf (stderr, "window_save: end\n");
03075 #endif
03076       return 1;
03077     }
03078
03079   // Closing and freeing memory
03080   gtk_widget_destroy (GTK_WIDGET (dlg));
03081 #if DEBUG
03082   fprintf (stderr, "window_save: end\n");
03083 #endif
03084   return 0;
03085 }
03086
03091 void
03092 window_run ()
03093 {
03094   unsigned int i;
03095   char *msg, *msg2, buffer[64], buffer2[64];
03096 #if DEBUG
03097   fprintf (stderr, "window_run: start\n");
03098 #endif
03099   if (!window_save ())
03100     {
03101 #if DEBUG
03102       fprintf (stderr, "window_run: end\n");
03103 #endif
03104       return;
03105     }
03106   running_new ();
03107   while (gtk_events_pending ())
03108     gtk_main_iteration ();
03109   calibrate_open ();
03110   gtk_widget_destroy (GTK_WIDGET (running->dialog));
03111   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03112   msg2 = g_strdup (buffer);
03113   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03114     {
03115       snprintf (buffer, 64, "%s = %s\n",
```

```
03116                 calibrate->label[i], format[calibrate->precision[i]]);
03117       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03118       msg = g_strconcat (msg2, buffer2, NULL);
03119       g_free (msg2);
03120     }
03121   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03122             calibrate->calculation_time);
03123   msg = g_strconcat (msg2, buffer, NULL);
03124   g_free (msg2);
03125   show_message (gettext ("Best result"), msg, INFO_TYPE);
03126   g_free (msg);
03127   calibrate_free ();
03128 #if DEBUG
03129   fprintf (stderr, "window_run: end\n");
03130 #endif
03131 }
03132
03137 void
03138 window_help ()
03139 {
03140   char *buffer, *buffer2;
03141 #if DEBUG
03142   fprintf (stderr, "window_help: start\n");
03143 #endif
03144   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03145                               gettext ("user-manual.pdf"), NULL);
03146   buffer = g_filename_to_uri (buffer2, NULL, NULL);
03147   g_free (buffer2);
03148   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03149 #if DEBUG
03150   fprintf (stderr, "window_help: uri=%s\n", buffer);
03151 #endif
03152   g_free (buffer);
03153 #if DEBUG
03154   fprintf (stderr, "window_help: end\n");
03155 #endif
03156 }
03157
03162 void
03163 window_about ()
03164 {
03165   static const gchar *authors[] = {
03166     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03167     "Borja Latorre Garcés <borja.latorre@csic.es>",
03168     NULL
03169   };
03170 #if DEBUG
03171   fprintf (stderr, "window_about: start\n");
03172 #endif
03173   gtk_show_about_dialog
03174     (window->window,
03175      "program_name", "MPCOTool",
03176      "comments",
03177      gettext ("A software to perform calibrations/optimizations of empirical "
03178               "parameters"),
03179      "authors", authors,
03180      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03181      "version", "1.2.3",
03182      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03183      "logo", window->logo,
03184      "website", "https://github.com/jburguete/mpcotool",
03185      "license-type", GTK_LICENSE_BSD, NULL);
03186 #if DEBUG
03187   fprintf (stderr, "window_about: end\n");
03188 #endif
03189 }
03190
03196 void
03197 window_update_gradient ()
03198 {
03199 #if DEBUG
03200   fprintf (stderr, "window_update_gradient: start\n");
03201 #endif
03202   gtk_widget_show (GTK_WIDGET (window->check_gradient));
03203   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03204     {
03205       gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03206       gtk_widget_show (GTK_WIDGET (window->label_step));
03207       gtk_widget_show (GTK_WIDGET (window->spin_step));
03208     }
03209   switch (window_get_gradient ())
03210     {
03211     case GRADIENT_METHOD_COORDINATES:
03212       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03213       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03214       break;
03215     default:
```

```
03216       gtk_widget_show (GTK_WIDGET (window->label_estimates));
03217       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03218     }
03219 #if DEBUG
03220   fprintf (stderr, "window_update_gradient: end\n");
03221 #endif
03222 }
03223
03228 void
03229 window_update ()
03230 {
03231   unsigned int i;
03232 #if DEBUG
03233   fprintf (stderr, "window_update: start\n");
03234 #endif
03235   gtk_widget_set_sensitive
03236     (GTK_WIDGET (window->button_evaluator),
03237      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03238                                    (window->check_evaluator)));
03239   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03240   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03241   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03242   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03243   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03244   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03245   gtk_widget_hide (GTK_WIDGET (window->label_bests));
03246   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03247   gtk_widget_hide (GTK_WIDGET (window->label_population));
03248   gtk_widget_hide (GTK_WIDGET (window->spin_population));
03249   gtk_widget_hide (GTK_WIDGET (window->label_generations));
03250   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03251   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03252   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03253   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03254   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03255   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03256   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03257   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03258   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03259   gtk_widget_hide (GTK_WIDGET (window->label_bits));
03260   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03261   gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03262   gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03263   gtk_widget_hide (GTK_WIDGET (window->label_step));
03264   gtk_widget_hide (GTK_WIDGET (window->spin_step));
03265   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03266   switch (window_get_algorithm ())
03267     {
03268     case ALGORITHM_MONTE_CARLO:
03269       gtk_widget_show (GTK_WIDGET (window->label_simulations));
03270       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03271       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03272       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03273       if (i > 1)
03274         {
03275           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03276           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03277           gtk_widget_show (GTK_WIDGET (window->label_bests));
03278           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03279         }
03280       window_update_gradient ();
03281       break;
03282     case ALGORITHM_SWEEP:
03283       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03284       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03285       if (i > 1)
03286         {
03287           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03288           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03289           gtk_widget_show (GTK_WIDGET (window->label_bests));
03290           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03291         }
03292       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03293       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03294       gtk_widget_show (GTK_WIDGET (window->check_gradient));
03295       window_update_gradient ();
03296       break;
03297     default:
03298       gtk_widget_show (GTK_WIDGET (window->label_population));
03299       gtk_widget_show (GTK_WIDGET (window->spin_population));
03300       gtk_widget_show (GTK_WIDGET (window->label_generations));
03301       gtk_widget_show (GTK_WIDGET (window->spin_generations));
03302       gtk_widget_show (GTK_WIDGET (window->label_mutation));
03303       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03304       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03305       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03306       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
```

```
03307       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03308       gtk_widget_show (GTK_WIDGET (window->label_bits));
03309       gtk_widget_show (GTK_WIDGET (window->spin_bits));
03310     }
03311   gtk_widget_set_sensitive
03312     (GTK_WIDGET (window->button_remove_experiment), input->
      nexperiments > 1);
03313   gtk_widget_set_sensitive
03314     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
03315   for (i = 0; i < input->ninputs; ++i)
03316     {
03317       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03318       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03319       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03320       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03321       g_signal_handler_block
03322         (window->check_template[i], window->id_template[i]);
03323       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
03324       gtk_toggle_button_set_active
03325         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03326       g_signal_handler_unblock
03327         (window->button_template[i], window->id_input[i]);
03328       g_signal_handler_unblock
03329         (window->check_template[i], window->id_template[i]);
03330     }
03331   if (i > 0)
03332     {
03333       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03334       gtk_widget_set_sensitive
03335         (GTK_WIDGET (window->button_template[i - 1]),
03336          gtk_toggle_button_get_active
03337          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
03338     }
03339   if (i < MAX_NINPUTS)
03340     {
03341       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03342       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03343       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03344       gtk_widget_set_sensitive
03345         (GTK_WIDGET (window->button_template[i]),
03346          gtk_toggle_button_get_active
03347          GTK_TOGGLE_BUTTON (window->check_template[i]));
03348       g_signal_handler_block
03349         (window->check_template[i], window->id_template[i]);
03350       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
03351       gtk_toggle_button_set_active
03352         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03353       g_signal_handler_unblock
03354         (window->button_template[i], window->id_input[i]);
03355       g_signal_handler_unblock
03356         (window->check_template[i], window->id_template[i]);
03357     }
03358   while (++i < MAX_NINPUTS)
03359     {
03360       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03361       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03362     }
03363   gtk_widget_set_sensitive
03364     (GTK_WIDGET (window->spin_minabs),
03365      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03366   gtk_widget_set_sensitive
03367     (GTK_WIDGET (window->spin_maxabs),
03368      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03369 #if DEBUG
03370   fprintf (stderr, "window_update: end\n");
03371 #endif
03372 }
03373
03378 void
03379 window_set_algorithm ()
03380 {
03381   int i;
03382 #if DEBUG
03383   fprintf (stderr, "window_set_algorithm: start\n");
03384 #endif
03385   i = window_get_algorithm ();
03386   switch (i)
03387     {
03388     case ALGORITHM_SWEEP:
03389       input->nsweeps = (unsigned int *) g_realloc
03390         (input->nsweeps, input->nvariables * sizeof (unsigned int));
03391       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03392       if (i < 0)
03393         i = 0;
```

```
03394       gtk_spin_button_set_value (window->spin_sweeps,
03395                                  (gdouble) input->nsweeps[i]);
03396       break;
03397     case ALGORITHM_GENETIC:
03398       input->nbits = (unsigned int *) g_realloc
03399         (input->nbits, input->nvariables * sizeof (unsigned int));
03400       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03401       if (i < 0)
03402         i = 0;
03403       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03404     }
03405   window_update ();
03406 #if DEBUG
03407   fprintf (stderr, "window_set_algorithm: end\n");
03408 #endif
03409 }
03410
03415 void
03416 window_set_experiment ()
03417 {
03418   unsigned int i, j;
03419   char *buffer1, *buffer2;
03420 #if DEBUG
03421   fprintf (stderr, "window_set_experiment: start\n");
03422 #endif
03423   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03424   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03425   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03426   buffer2 = g_build_filename (input->directory, buffer1, NULL);
03427   g_free (buffer1);
03428   g_signal_handler_block
03429     (window->button_experiment, window->id_experiment_name);
03430   gtk_file_chooser_set_filename
03431     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03432   g_signal_handler_unblock
03433     (window->button_experiment, window->id_experiment_name);
03434   g_free (buffer2);
03435   for (j = 0; j < input->ninputs; ++j)
03436     {
03437       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03438       buffer2
03439         = g_build_filename (input->directory, input->template[j][i], NULL);
03440       gtk_file_chooser_set_filename
03441         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03442       g_free (buffer2);
03443       g_signal_handler_unblock
03444         (window->button_template[j], window->id_input[j]);
03445     }
03446 #if DEBUG
03447   fprintf (stderr, "window_set_experiment: end\n");
03448 #endif
03449 }
03450
03455 void
03456 window_remove_experiment ()
03457 {
03458   unsigned int i, j;
03459 #if DEBUG
03460   fprintf (stderr, "window_remove_experiment: start\n");
03461 #endif
03462   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03463   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03464   gtk_combo_box_text_remove (window->combo_experiment, i);
03465   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03466   xmlFree (input->experiment[i]);
03467   --input->nexperiments;
03468   for (j = i; j < input->nexperiments; ++j)
03469     {
03470       input->experiment[j] = input->experiment[j + 1];
03471       input->weight[j] = input->weight[j + 1];
03472     }
03473   j = input->nexperiments - 1;
03474   if (i > j)
03475     i = j;
03476   for (j = 0; j < input->ninputs; ++j)
03477     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03478   g_signal_handler_block
03479     (window->button_experiment, window->id_experiment_name);
03480   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03481   g_signal_handler_unblock
03482     (window->button_experiment, window->id_experiment_name);
03483   for (j = 0; j < input->ninputs; ++j)
```

```
03484     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03485   window_update ();
03486 #if DEBUG
03487   fprintf (stderr, "window_remove_experiment: end\n");
03488 #endif
03489 }
03490
03495 void
03496 window_add_experiment ()
03497 {
03498   unsigned int i, j;
03499 #if DEBUG
03500   fprintf (stderr, "window_add_experiment: start\n");
03501 #endif
03502   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03503   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03504   gtk_combo_box_text_insert_text
03505     (window->combo_experiment, i, input->experiment[i]);
03506   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03507   input->experiment = (char **) g_realloc
03508     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03509   input->weight = (double *) g_realloc
03510     (input->weight, (input->nexperiments + 1) * sizeof (double));
03511   for (j = input->nexperiments - 1; j > i; --j)
03512     {
03513       input->experiment[j + 1] = input->experiment[j];
03514       input->weight[j + 1] = input->weight[j];
03515     }
03516   input->experiment[j + 1]
03517     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03518   input->weight[j + 1] = input->weight[j];
03519   ++input->nexperiments;
03520   for (j = 0; j < input->ninputs; ++j)
03521     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03522   g_signal_handler_block
03523     (window->button_experiment, window->id_experiment_name);
03524   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03525   g_signal_handler_unblock
03526     (window->button_experiment, window->id_experiment_name);
03527   for (j = 0; j < input->ninputs; ++j)
03528     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03529   window_update ();
03530 #if DEBUG
03531   fprintf (stderr, "window_add_experiment: end\n");
03532 #endif
03533 }
03534
03539 void
03540 window_name_experiment ()
03541 {
03542   unsigned int i;
03543   char *buffer;
03544   GFile *file1, *file2;
03545 #if DEBUG
03546   fprintf (stderr, "window_name_experiment: start\n");
03547 #endif
03548   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03549   file1
03550     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03551   file2 = g_file_new_for_path (input->directory);
03552   buffer = g_file_get_relative_path (file2, file1);
03553   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03554   gtk_combo_box_text_remove (window->combo_experiment, i);
03555   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03556   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03557   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03558   g_free (buffer);
03559   g_object_unref (file2);
03560   g_object_unref (file1);
03561 #if DEBUG
03562   fprintf (stderr, "window_name_experiment: end\n");
03563 #endif
03564 }
03565
03570 void
03571 window_weight_experiment ()
03572 {
03573   unsigned int i;
03574 #if DEBUG
03575   fprintf (stderr, "window_weight_experiment: start\n");
```

```
03576 #endif
03577   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03578   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03579 #if DEBUG
03580   fprintf (stderr, "window_weight_experiment: end\n");
03581 #endif
03582 }
03583
03589 void
03590 window_inputs_experiment ()
03591 {
03592   unsigned int j;
03593 #if DEBUG
03594   fprintf (stderr, "window_inputs_experiment: start\n");
03595 #endif
03596   j = input->ninputs - 1;
03597   if (j
03598       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03599                                         (window->check_template[j])))
03600     --input->ninputs;
03601   if (input->ninputs < MAX_NINPUTS
03602       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03603                                        (window->check_template[j])))
03604     {
03605       ++input->ninputs;
03606       for (j = 0; j < input->ninputs; ++j)
03607         {
03608           input->template[j] = (char **)
03609             g_realloc (input->template[j], input->nvariables * sizeof (char *));
03610         }
03611     }
03612   window_update ();
03613 #if DEBUG
03614   fprintf (stderr, "window_inputs_experiment: end\n");
03615 #endif
03616 }
03617
03625 void
03626 window_template_experiment (void *data)
03627 {
03628   unsigned int i, j;
03629   char *buffer;
03630   GFile *file1, *file2;
03631 #if DEBUG
03632   fprintf (stderr, "window_template_experiment: start\n");
03633 #endif
03634   i = (size_t) data;
03635   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03636   file1
03637     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03638   file2 = g_file_new_for_path (input->directory);
03639   buffer = g_file_get_relative_path (file2, file1);
03640   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03641   g_free (buffer);
03642   g_object_unref (file2);
03643   g_object_unref (file1);
03644 #if DEBUG
03645   fprintf (stderr, "window_template_experiment: end\n");
03646 #endif
03647 }
03648
03653 void
03654 window_set_variable ()
03655 {
03656   unsigned int i;
03657 #if DEBUG
03658   fprintf (stderr, "window_set_variable: start\n");
03659 #endif
03660   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03661   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03662   gtk_entry_set_text (window->entry_variable, input->label[i]);
03663   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03664   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03665   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03666   if (input->rangeminabs[i] != -G_MAXDOUBLE)
03667     {
03668       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
03669       gtk_toggle_button_set_active
03670         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03671     }
03672   else
03673     {
03674       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03675       gtk_toggle_button_set_active
```

```
03676         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03677     }
03678   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03679     {
03680       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
03681       gtk_toggle_button_set_active
03682         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03683     }
03684   else
03685     {
03686       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03687       gtk_toggle_button_set_active
03688         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03689     }
03690   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
03691   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
03692   if (input->nsteps)
03693     gtk_spin_button_set_value (window->spin_step, input->step[i]);
03694 #if DEBUG
03695   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03696            input->precision[i]);
03697 #endif
03698   switch (window_get_algorithm ())
03699     {
03700     case ALGORITHM_SWEEP:
03701       gtk_spin_button_set_value (window->spin_sweeps,
03702                                  (gdouble) input->nsweeps[i]);
03703 #if DEBUG
03704       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03705                input->nsweeps[i]);
03706 #endif
03707       break;
03708     case ALGORITHM_GENETIC:
03709       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03710 #if DEBUG
03711       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03712                input->nbits[i]);
03713 #endif
03714       break;
03715     }
03716   window_update ();
03717 #if DEBUG
03718   fprintf (stderr, "window_set_variable: end\n");
03719 #endif
03720 }
03721
03726 void
03727 window_remove_variable ()
03728 {
03729   unsigned int i, j;
03730 #if DEBUG
03731   fprintf (stderr, "window_remove_variable: start\n");
03732 #endif
03733   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03734   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03735   gtk_combo_box_text_remove (window->combo_variable, i);
03736   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03737   xmlFree (input->label[i]);
03738   --input->nvariables;
03739   for (j = i; j < input->nvariables; ++j)
03740     {
03741       input->label[j] = input->label[j + 1];
03742       input->rangemin[j] = input->rangemin[j + 1];
03743       input->rangemax[j] = input->rangemax[j + 1];
03744       input->rangeminabs[j] = input->rangeminabs[j + 1];
03745       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03746       input->precision[j] = input->precision[j + 1];
03747       input->step[j] = input->step[j + 1];
03748       switch (window_get_algorithm ())
03749         {
03750         case ALGORITHM_SWEEP:
03751           input->nsweeps[j] = input->nsweeps[j + 1];
03752           break;
03753         case ALGORITHM_GENETIC:
03754           input->nbits[j] = input->nbits[j + 1];
03755         }
03756     }
03757   j = input->nvariables - 1;
03758   if (i > j)
03759     i = j;
03760   g_signal_handler_block (window->entry_variable, window->
```

```
      id_variable_label);
03761   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03762   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03763   window_update ();
03764 #if DEBUG
03765   fprintf (stderr, "window_remove_variable: end\n");
03766 #endif
03767 }
03768
03773 void
03774 window_add_variable ()
03775 {
03776   unsigned int i, j;
03777 #if DEBUG
03778   fprintf (stderr, "window_add_variable: start\n");
03779 #endif
03780   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03781   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03782   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03783   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03784   input->label = (char **) g_realloc
03785     (input->label, (input->nvariables + 1) * sizeof (char *));
03786   input->rangemin = (double *) g_realloc
03787     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03788   input->rangemax = (double *) g_realloc
03789     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03790   input->rangeminabs = (double *) g_realloc
03791     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03792   input->rangemaxabs = (double *) g_realloc
03793     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03794   input->precision = (unsigned int *) g_realloc
03795     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03796   input->step = (double *) g_realloc
03797     (input->step, (input->nvariables + 1) * sizeof (double));
03798   for (j = input->nvariables - 1; j > i; --j)
03799     {
03800       input->label[j + 1] = input->label[j];
03801       input->rangemin[j + 1] = input->rangemin[j];
03802       input->rangemax[j + 1] = input->rangemax[j];
03803       input->rangeminabs[j + 1] = input->rangeminabs[j];
03804       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03805       input->precision[j + 1] = input->precision[j];
03806       input->step[j + 1] = input->step[j];
03807     }
03808   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03809   input->rangemin[j + 1] = input->rangemin[j];
03810   input->rangemax[j + 1] = input->rangemax[j];
03811   input->rangeminabs[j + 1] = input->rangeminabs[j];
03812   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03813   input->precision[j + 1] = input->precision[j];
03814   input->step[j + 1] = input->step[j];
03815   switch (window_get_algorithm ())
03816     {
03817     case ALGORITHM_SWEEP:
03818       input->nsweeps = (unsigned int *) g_realloc
03819         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03820       for (j = input->nvariables - 1; j > i; --j)
03821         input->nsweeps[j + 1] = input->nsweeps[j];
03822       input->nsweeps[j + 1] = input->nsweeps[j];
03823       break;
03824     case ALGORITHM_GENETIC:
03825       input->nbits = (unsigned int *) g_realloc
03826         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03827       for (j = input->nvariables - 1; j > i; --j)
03828         input->nbits[j + 1] = input->nbits[j];
03829       input->nbits[j + 1] = input->nbits[j];
03830     }
03831   ++input->nvariables;
03832   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03833   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03834   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03835   window_update ();
03836 #if DEBUG
03837   fprintf (stderr, "window_add_variable: end\n");
03838 #endif
03839 }
03840
03845 void
03846 window_label_variable ()
03847 {
03848   unsigned int i;
```

```
03849   const char *buffer;
03850 #if DEBUG
03851   fprintf (stderr, "window_label_variable: start\n");
03852 #endif
03853   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03854   buffer = gtk_entry_get_text (window->entry_variable);
03855   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03856   gtk_combo_box_text_remove (window->combo_variable, i);
03857   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03858   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03859   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03860 #if DEBUG
03861   fprintf (stderr, "window_label_variable: end\n");
03862 #endif
03863 }
03864
03869 void
03870 window_precision_variable ()
03871 {
03872   unsigned int i;
03873 #if DEBUG
03874   fprintf (stderr, "window_precision_variable: start\n");
03875 #endif
03876   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03877   input->precision[i]
03878     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03879   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03880   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03881   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03882   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03883 #if DEBUG
03884   fprintf (stderr, "window_precision_variable: end\n");
03885 #endif
03886 }
03887
03892 void
03893 window_rangemin_variable ()
03894 {
03895   unsigned int i;
03896 #if DEBUG
03897   fprintf (stderr, "window_rangemin_variable: start\n");
03898 #endif
03899   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03900   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03901 #if DEBUG
03902   fprintf (stderr, "window_rangemin_variable: end\n");
03903 #endif
03904 }
03905
03910 void
03911 window_rangemax_variable ()
03912 {
03913   unsigned int i;
03914 #if DEBUG
03915   fprintf (stderr, "window_rangemax_variable: start\n");
03916 #endif
03917   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03918   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03919 #if DEBUG
03920   fprintf (stderr, "window_rangemax_variable: end\n");
03921 #endif
03922 }
03923
03928 void
03929 window_rangeminabs_variable ()
03930 {
03931   unsigned int i;
03932 #if DEBUG
03933   fprintf (stderr, "window_rangeminabs_variable: start\n");
03934 #endif
03935   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03936   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03937 #if DEBUG
03938   fprintf (stderr, "window_rangeminabs_variable: end\n");
03939 #endif
03940 }
03941
03946 void
03947 window_rangemaxabs_variable ()
03948 {
03949   unsigned int i;
03950 #if DEBUG
03951   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03952 #endif
```

```
03953   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03954   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03955 #if DEBUG
03956   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03957 #endif
03958 }
03959
03964 void
03965 window_step_variable ()
03966 {
03967   unsigned int i;
03968 #if DEBUG
03969   fprintf (stderr, "window_step_variable: start\n");
03970 #endif
03971   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03972   input->step[i] = gtk_spin_button_get_value (window->spin_step);
03973 #if DEBUG
03974   fprintf (stderr, "window_step_variable: end\n");
03975 #endif
03976 }
03977
03982 void
03983 window_update_variable ()
03984 {
03985   int i;
03986 #if DEBUG
03987   fprintf (stderr, "window_update_variable: start\n");
03988 #endif
03989   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03990   if (i < 0)
03991     i = 0;
03992   switch (window_get_algorithm ())
03993     {
03994     case ALGORITHM_SWEEP:
03995       input->nsweeps[i]
03996         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03997 #if DEBUG
03998       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03999                input->nsweeps[i]);
04000 #endif
04001       break;
04002     case ALGORITHM_GENETIC:
04003       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
04004 #if DEBUG
04005       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
04006                input->nbits[i]);
04007 #endif
04008     }
04009 #if DEBUG
04010   fprintf (stderr, "window_update_variable: end\n");
04011 #endif
04012 }
04013
04021 int
04022 window_read (char *filename)
04023 {
04024   unsigned int i;
04025   char *buffer;
04026 #if DEBUG
04027   fprintf (stderr, "window_read: start\n");
04028 #endif
04029
04030   // Reading new input file
04031   input_free ();
04032   if (!input_open (filename))
04033     return 0;
04034
04035   // Setting GTK+ widgets data
04036   gtk_entry_set_text (window->entry_result, input->result);
04037   gtk_entry_set_text (window->entry_variables, input->variables);
04038   buffer = g_build_filename (input->directory, input->simulator, NULL);
04039   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04040                                  (window->button_simulator), buffer);
04041   g_free (buffer);
04042   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
04043                                 (size_t) input->evaluator);
04044   if (input->evaluator)
04045     {
04046       buffer = g_build_filename (input->directory, input->evaluator, NULL);
04047       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
04048                                      (window->button_evaluator), buffer);
04049       g_free (buffer);
04050     }
04051   gtk_toggle_button_set_active
04052     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
```

```
04053   switch (input->algorithm)
04054     {
04055     case ALGORITHM_MONTE_CARLO:
04056       gtk_spin_button_set_value (window->spin_simulations,
04057                                  (gdouble) input->nsimulations);
04058     case ALGORITHM_SWEEP:
04059       gtk_spin_button_set_value (window->spin_iterations,
04060                                  (gdouble) input->niterations);
04061       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
04062       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
04063       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
04064                                     input->nsteps);
04065       if (input->nsteps)
04066         {
04067           gtk_toggle_button_set_active
04068             (GTK_TOGGLE_BUTTON (window->button_gradient
04069                                 [input->gradient_method]), TRUE);
04070           gtk_spin_button_set_value (window->spin_steps,
04071                                      (gdouble) input->nsteps);
04072           gtk_spin_button_set_value (window->spin_relaxation,
04073                                      (gdouble) input->relaxation);
04074           switch (input->gradient_method)
04075             {
04076             case GRADIENT_METHOD_RANDOM:
04077               gtk_spin_button_set_value (window->spin_estimates,
04078                                          (gdouble) input->nestimates);
04079             }
04080         }
04081       break;
04082     default:
04083       gtk_spin_button_set_value (window->spin_population,
04084                                  (gdouble) input->nsimulations);
04085       gtk_spin_button_set_value (window->spin_generations,
04086                                  (gdouble) input->niterations);
04087       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
04088       gtk_spin_button_set_value (window->spin_reproduction,
04089                                  input->reproduction_ratio);
04090       gtk_spin_button_set_value (window->spin_adaptation,
04091                                  input->adaptation_ratio);
04092     }
04093   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
04094   g_signal_handler_block (window->button_experiment,
04095                           window->id_experiment_name);
04096   gtk_combo_box_text_remove_all (window->combo_experiment);
04097   for (i = 0; i < input->nexperiments; ++i)
04098     gtk_combo_box_text_append_text (window->combo_experiment,
04099                                     input->experiment[i]);
04100   g_signal_handler_unblock
04101     (window->button_experiment, window->id_experiment_name);
04102   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
04103   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
04104   g_signal_handler_block (window->combo_variable, window->
      id_variable);
04105   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
04106   gtk_combo_box_text_remove_all (window->combo_variable);
04107   for (i = 0; i < input->nvariables; ++i)
04108     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
04109   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
04110   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
04111   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
04112   window_set_variable ();
04113   window_update ();
04114
04115 #if DEBUG
04116   fprintf (stderr, "window_read: end\n");
04117 #endif
04118   return 1;
04119 }
04120
04125 void
04126 window_open ()
04127 {
04128   char *buffer, *directory, *name;
04129   GtkFileChooserDialog *dlg;
04130
04131 #if DEBUG
04132   fprintf (stderr, "window_open: start\n");
04133 #endif
```

```
04134
04135   // Saving a backup of the current input file
04136   directory = g_strdup (input->directory);
04137   name = g_strdup (input->name);
04138
04139   // Opening dialog
04140   dlg = (GtkFileChooserDialog *)
04141     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04142                                  window->window,
04143                                  GTK_FILE_CHOOSER_ACTION_OPEN,
04144                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04145                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04146   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04147     {
04148
04149       // Traying to open the input file
04150       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04151       if (!window_read (buffer))
04152         {
04153 #if DEBUG
04154           fprintf (stderr, "window_open: error reading input file\n");
04155 #endif
04156           g_free (buffer);
04157
04158           // Reading backup file on error
04159           buffer = g_build_filename (directory, name, NULL);
04160           if (!input_open (buffer))
04161             {
04162
04163               // Closing on backup file reading error
04164 #if DEBUG
04165               fprintf (stderr, "window_read: error reading backup file\n");
04166 #endif
04167               g_free (buffer);
04168               break;
04169             }
04170           g_free (buffer);
04171         }
04172       else
04173         {
04174           g_free (buffer);
04175           break;
04176         }
04177     }
04178
04179   // Freeing and closing
04180   g_free (name);
04181   g_free (directory);
04182   gtk_widget_destroy (GTK_WIDGET (dlg));
04183 #if DEBUG
04184   fprintf (stderr, "window_open: end\n");
04185 #endif
04186 }
04187
04192 void
04193 window_new ()
04194 {
04195   unsigned int i;
04196   char *buffer, *buffer2, buffer3[64];
04197   char *label_algorithm[NALGORITHMS] = {
04198     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04199   };
04200   char *tip_algorithm[NALGORITHMS] = {
04201     gettext ("Monte-Carlo brute force algorithm"),
04202     gettext ("Sweep brute force algorithm"),
04203     gettext ("Genetic algorithm")
04204   };
04205   char *label_gradient[NGRADIENTS] = {
04206     gettext ("_Coordinates descent"), gettext ("_Random")
04207   };
04208   char *tip_gradient[NGRADIENTS] = {
04209     gettext ("Coordinates descent gradient estimate method"),
04210     gettext ("Random gradient estimate method")
04211   };
04212
04213 #if DEBUG
04214   fprintf (stderr, "window_new: start\n");
04215 #endif
04216
04217   // Creating the window
04218   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04219
04220   // Finish when closing the window
04221   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04222
04223   // Setting the window title
04224   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
```

```
04225
04226   // Creating the open button
04227   window->button_open = (GtkToolButton *) gtk_tool_button_new
04228     (gtk_image_new_from_icon_name ("document-open",
04229                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04230      gettext ("Open"));
04231   g_signal_connect (window->button_open, "clicked", window_open, NULL);
04232
04233   // Creating the save button
04234   window->button_save = (GtkToolButton *) gtk_tool_button_new
04235     (gtk_image_new_from_icon_name ("document-save",
04236                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04237      gettext ("Save"));
04238   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
04239                     NULL);
04240
04241   // Creating the run button
04242   window->button_run = (GtkToolButton *) gtk_tool_button_new
04243     (gtk_image_new_from_icon_name ("system-run",
04244                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04245      gettext ("Run"));
04246   g_signal_connect (window->button_run, "clicked", window_run, NULL);
04247
04248   // Creating the options button
04249   window->button_options = (GtkToolButton *) gtk_tool_button_new
04250     (gtk_image_new_from_icon_name ("preferences-system",
04251                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04252      gettext ("Options"));
04253   g_signal_connect (window->button_options, "clicked", options_new, NULL);
04254
04255   // Creating the help button
04256   window->button_help = (GtkToolButton *) gtk_tool_button_new
04257     (gtk_image_new_from_icon_name ("help-browser",
04258                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04259      gettext ("Help"));
04260   g_signal_connect (window->button_help, "clicked", window_help, NULL);
04261
04262   // Creating the about button
04263   window->button_about = (GtkToolButton *) gtk_tool_button_new
04264     (gtk_image_new_from_icon_name ("help-about",
04265                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04266      gettext ("About"));
04267   g_signal_connect (window->button_about, "clicked", window_about, NULL);
04268
04269   // Creating the exit button
04270   window->button_exit = (GtkToolButton *) gtk_tool_button_new
04271     (gtk_image_new_from_icon_name ("application-exit",
04272                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04273      gettext ("Exit"));
04274   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
04275
04276   // Creating the buttons bar
04277   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04278   gtk_toolbar_insert
04279     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
04280   gtk_toolbar_insert
04281     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
04282   gtk_toolbar_insert
04283     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
04284   gtk_toolbar_insert
04285     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
04286   gtk_toolbar_insert
04287     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
04288   gtk_toolbar_insert
04289     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
04290   gtk_toolbar_insert
04291     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
04292   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04293
04294   // Creating the simulator program label and entry
04295   window->label_simulator
04296     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04297   window->button_simulator = (GtkFileChooserButton *)
04298     gtk_file_chooser_button_new (gettext ("Simulator program"),
04299                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04300   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04301                                gettext ("Simulator program executable file"));
04302
04303   // Creating the evaluator program label and entry
04304   window->check_evaluator = (GtkCheckButton *)
04305     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
04306   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
04307   window->button_evaluator = (GtkFileChooserButton *)
04308     gtk_file_chooser_button_new (gettext ("Evaluator program"),
04309                                  GTK_FILE_CHOOSER_ACTION_OPEN);
```

```
04310   gtk_widget_set_tooltip_text
04311     (GTK_WIDGET (window->button_evaluator),
04312      gettext ("Optional evaluator program executable file"));
04313
04314   // Creating the results files labels and entries
04315   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
04316   window->entry_result = (GtkEntry *) gtk_entry_new ();
04317   gtk_widget_set_tooltip_text
04318     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
04319   window->label_variables
04320     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
04321   window->entry_variables = (GtkEntry *) gtk_entry_new ();
04322   gtk_widget_set_tooltip_text
04323     (GTK_WIDGET (window->entry_variables),
04324      gettext ("All simulated results file"));
04325
04326   // Creating the files grid and attaching widgets
04327   window->grid_files = (GtkGrid *) gtk_grid_new ();
04328   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
04329                    0, 0, 1, 1);
04330   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
04331                    1, 0, 1, 1);
04332   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
04333                    2, 0, 1, 1);
04334   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
04335                    3, 0, 1, 1);
04336   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
04337                    0, 1, 1, 1);
04338   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
04339                    1, 1, 1, 1);
04340   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
04341                    2, 1, 1, 1);
04342   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
04343                    3, 1, 1, 1);
04344
04345   // Creating the algorithm properties
04346   window->label_simulations = (GtkLabel *) gtk_label_new
04347     (gettext ("Simulations number"));
04348   window->spin_simulations
04349     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04350   gtk_widget_set_tooltip_text
04351     (GTK_WIDGET (window->spin_simulations),
04352      gettext ("Number of simulations to perform for each iteration"));
04353   window->label_iterations = (GtkLabel *)
04354     gtk_label_new (gettext ("Iterations number"));
04355   window->spin_iterations
04356     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04357   gtk_widget_set_tooltip_text
04358     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04359   g_signal_connect
04360     (window->spin_iterations, "value-changed", window_update, NULL);
04361   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04362   window->spin_tolerance
04363     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04364   gtk_widget_set_tooltip_text
04365     (GTK_WIDGET (window->spin_tolerance),
04366      gettext ("Tolerance to set the variable interval on the next iteration"));
04367   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04368   window->spin_bests
04369     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04370   gtk_widget_set_tooltip_text
04371     (GTK_WIDGET (window->spin_bests),
04372      gettext ("Number of best simulations used to set the variable interval "
04373               "on the next iteration"));
04374   window->label_population
04375     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04376   window->spin_population
04377     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04378   gtk_widget_set_tooltip_text
04379     (GTK_WIDGET (window->spin_population),
04380      gettext ("Number of population for the genetic algorithm"));
04381   window->label_generations
04382     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04383   window->spin_generations
04384     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04385   gtk_widget_set_tooltip_text
04386     (GTK_WIDGET (window->spin_generations),
04387      gettext ("Number of generations for the genetic algorithm"));
04388   window->label_mutation
```

```
04389     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04390   window->spin_mutation
04391     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04392   gtk_widget_set_tooltip_text
04393     (GTK_WIDGET (window->spin_mutation),
04394      gettext ("Ratio of mutation for the genetic algorithm"));
04395   window->label_reproduction
04396     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04397   window->spin_reproduction
04398     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04399   gtk_widget_set_tooltip_text
04400     (GTK_WIDGET (window->spin_reproduction),
04401      gettext ("Ratio of reproduction for the genetic algorithm"));
04402   window->label_adaptation
04403     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04404   window->spin_adaptation
04405     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04406   gtk_widget_set_tooltip_text
04407     (GTK_WIDGET (window->spin_adaptation),
04408      gettext ("Ratio of adaptation for the genetic algorithm"));
04409
04410   // Creating the gradient based method properties
04411   window->check_gradient = (GtkCheckButton *)
04412     gtk_check_button_new_with_mnemonic (gettext ("_Gradient based method"));
04413   g_signal_connect (window->check_gradient, "clicked",
      window_update, NULL);
04414   window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04415   window->button_gradient[0] = (GtkRadioButton *)
04416     gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04417   gtk_grid_attach (window->grid_gradient,
04418                    GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04419   g_signal_connect (window->button_gradient[0], "clicked",
      window_update, NULL);
04420   for (i = 0; ++i < NGRADIENTS;)
04421     {
04422       window->button_gradient[i] = (GtkRadioButton *)
04423         gtk_radio_button_new_with_mnemonic
04424         (gtk_radio_button_get_group (window->button_gradient[0]),
04425          label_gradient[i]);
04426       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04427                                    tip_gradient[i]);
04428       gtk_grid_attach (window->grid_gradient,
04429                        GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04430       g_signal_connect (window->button_gradient[i], "clicked",
04431                         window_update, NULL);
04432     }
04433   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04434   window->spin_steps = (GtkSpinButton *)
04435     gtk_spin_button_new_with_range (1., 1.e12, 1.);
04436   window->label_estimates
04437     = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04438   window->spin_estimates = (GtkSpinButton *)
04439     gtk_spin_button_new_with_range (1., 1.e3, 1.);
04440   window->label_relaxation
04441     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04442   window->spin_relaxation = (GtkSpinButton *)
04443     gtk_spin_button_new_with_range (0., 2., 0.001);
04444   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_steps),
04445                    0, NGRADIENTS, 1, 1);
04446   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_steps),
04447                    1, NGRADIENTS, 1, 1);
04448   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_estimates),
04449                    0, NGRADIENTS + 1, 1, 1);
04450   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_estimates),
04451                    1, NGRADIENTS + 1, 1, 1);
04452   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      label_relaxation),
04453                    0, NGRADIENTS + 2, 1, 1);
04454   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
      spin_relaxation),
04455                    1, NGRADIENTS + 2, 1, 1);
04456
04457   // Creating the array of algorithms
04458   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
04459   window->button_algorithm[0] = (GtkRadioButton *)
04460     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04461   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
04462                                tip_algorithm[0]);
04463   gtk_grid_attach (window->grid_algorithm,
04464                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
04465   g_signal_connect (window->button_algorithm[0], "clicked",
04466                     window_set_algorithm, NULL);
04467   for (i = 0; ++i < NALGORITHMS;)
```

```
04468     {
04469       window->button_algorithm[i] = (GtkRadioButton *)
04470         gtk_radio_button_new_with_mnemonic
04471         (gtk_radio_button_get_group (window->button_algorithm[0]),
04472          label_algorithm[i]);
04473       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
04474                                    tip_algorithm[i]);
04475       gtk_grid_attach (window->grid_algorithm,
04476                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
04477       g_signal_connect (window->button_algorithm[i], "clicked",
04478                         window_set_algorithm, NULL);
04479     }
04480   gtk_grid_attach (window->grid_algorithm,
04481                    GTK_WIDGET (window->label_simulations), 0,
04482                    NALGORITHMS, 1, 1);
04483   gtk_grid_attach (window->grid_algorithm,
04484                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
04485   gtk_grid_attach (window->grid_algorithm,
04486                    GTK_WIDGET (window->label_iterations), 0,
04487                    NALGORITHMS + 1, 1, 1);
04488   gtk_grid_attach (window->grid_algorithm,
04489                    GTK_WIDGET (window->spin_iterations), 1,
04490                    NALGORITHMS + 1, 1, 1);
04491   gtk_grid_attach (window->grid_algorithm,
04492                    GTK_WIDGET (window->label_tolerance), 0,
04493                    NALGORITHMS + 2, 1, 1);
04494   gtk_grid_attach (window->grid_algorithm,
04495                    GTK_WIDGET (window->spin_tolerance), 1,
04496                    NALGORITHMS + 2, 1, 1);
04497   gtk_grid_attach (window->grid_algorithm,
04498                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
04499   gtk_grid_attach (window->grid_algorithm,
04500                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
04501   gtk_grid_attach (window->grid_algorithm,
04502                    GTK_WIDGET (window->label_population), 0,
04503                    NALGORITHMS + 4, 1, 1);
04504   gtk_grid_attach (window->grid_algorithm,
04505                    GTK_WIDGET (window->spin_population), 1,
04506                    NALGORITHMS + 4, 1, 1);
04507   gtk_grid_attach (window->grid_algorithm,
04508                    GTK_WIDGET (window->label_generations), 0,
04509                    NALGORITHMS + 5, 1, 1);
04510   gtk_grid_attach (window->grid_algorithm,
04511                    GTK_WIDGET (window->spin_generations), 1,
04512                    NALGORITHMS + 5, 1, 1);
04513   gtk_grid_attach (window->grid_algorithm,
04514                    GTK_WIDGET (window->label_mutation), 0,
04515                    NALGORITHMS + 6, 1, 1);
04516   gtk_grid_attach (window->grid_algorithm,
04517                    GTK_WIDGET (window->spin_mutation), 1,
04518                    NALGORITHMS + 6, 1, 1);
04519   gtk_grid_attach (window->grid_algorithm,
04520                    GTK_WIDGET (window->label_reproduction), 0,
04521                    NALGORITHMS + 7, 1, 1);
04522   gtk_grid_attach (window->grid_algorithm,
04523                    GTK_WIDGET (window->spin_reproduction), 1,
04524                    NALGORITHMS + 7, 1, 1);
04525   gtk_grid_attach (window->grid_algorithm,
04526                    GTK_WIDGET (window->label_adaptation), 0,
04527                    NALGORITHMS + 8, 1, 1);
04528   gtk_grid_attach (window->grid_algorithm,
04529                    GTK_WIDGET (window->spin_adaptation), 1,
04530                    NALGORITHMS + 8, 1, 1);
04531   gtk_grid_attach (window->grid_algorithm,
04532                    GTK_WIDGET (window->check_gradient), 0,
04533                    NALGORITHMS + 9, 2, 1);
04534   gtk_grid_attach (window->grid_algorithm,
04535                    GTK_WIDGET (window->grid_gradient), 0,
04536                    NALGORITHMS + 10, 2, 1);
04537   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
04538   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04539                      GTK_WIDGET (window->grid_algorithm));
04540
04541   // Creating the variable widgets
04542   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04543   gtk_widget_set_tooltip_text
04544     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04545   window->id_variable = g_signal_connect
04546     (window->combo_variable, "changed", window_set_variable, NULL);
04547   window->button_add_variable
04548     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04549                                                    GTK_ICON_SIZE_BUTTON);
04550   g_signal_connect
04551     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
04552   gtk_widget_set_tooltip_text
04553     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
```

```
04554   window->button_remove_variable
04555     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04556                                                    GTK_ICON_SIZE_BUTTON);
04557   g_signal_connect
04558     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
04559   gtk_widget_set_tooltip_text
04560     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04561   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04562   window->entry_variable = (GtkEntry *) gtk_entry_new ();
04563   gtk_widget_set_tooltip_text
04564     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04565   window->id_variable_label = g_signal_connect
04566     (window->entry_variable, "changed", window_label_variable, NULL);
04567   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04568   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04569     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04570   gtk_widget_set_tooltip_text
04571     (GTK_WIDGET (window->spin_min),
04572      gettext ("Minimum initial value of the variable"));
04573   window->scrolled_min
04574     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04575   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04576                      GTK_WIDGET (window->spin_min));
04577   g_signal_connect (window->spin_min, "value-changed",
04578                     window_rangemin_variable, NULL);
04579   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04580   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04581     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04582   gtk_widget_set_tooltip_text
04583     (GTK_WIDGET (window->spin_max),
04584      gettext ("Maximum initial value of the variable"));
04585   window->scrolled_max
04586     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04587   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04588                      GTK_WIDGET (window->spin_max));
04589   g_signal_connect (window->spin_max, "value-changed",
04590                     window_rangemax_variable, NULL);
04591   window->check_minabs = (GtkCheckButton *)
04592     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
04593   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04594   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04596   gtk_widget_set_tooltip_text
04597     (GTK_WIDGET (window->spin_minabs),
04598      gettext ("Minimum allowed value of the variable"));
04599   window->scrolled_minabs
04600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04601   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04602                      GTK_WIDGET (window->spin_minabs));
04603   g_signal_connect (window->spin_minabs, "value-changed",
04604                     window_rangeminabs_variable, NULL);
04605   window->check_maxabs = (GtkCheckButton *)
04606     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
04607   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04608   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04609     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04610   gtk_widget_set_tooltip_text
04611     (GTK_WIDGET (window->spin_maxabs),
04612      gettext ("Maximum allowed value of the variable"));
04613   window->scrolled_maxabs
04614     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04615   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04616                      GTK_WIDGET (window->spin_maxabs));
04617   g_signal_connect (window->spin_maxabs, "value-changed",
04618                     window_rangemaxabs_variable, NULL);
04619   window->label_precision
04620     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04621   window->spin_precision = (GtkSpinButton *)
04622     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04623   gtk_widget_set_tooltip_text
04624     (GTK_WIDGET (window->spin_precision),
04625      gettext ("Number of precision floating point digits\n"
04626               "0 is for integer numbers"));
04627   g_signal_connect (window->spin_precision, "value-changed",
04628                     window_precision_variable, NULL);
04629   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
04630   window->spin_sweeps
04631     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04632   gtk_widget_set_tooltip_text
04633     (GTK_WIDGET (window->spin_sweeps),
04634      gettext ("Number of steps sweeping the variable"));
04635   g_signal_connect
04636     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
04637   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
04638   window->spin_bits
04639     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
```

```
04640   gtk_widget_set_tooltip_text
04641     (GTK_WIDGET (window->spin_bits),
04642      gettext ("Number of bits to encode the variable"));
04643   g_signal_connect
04644     (window->spin_bits, "value-changed", window_update_variable, NULL);
04645   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
04646   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
04647     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04648   gtk_widget_set_tooltip_text
04649     (GTK_WIDGET (window->spin_step),
04650      gettext ("Initial step size for the gradient based method"));
04651   window->scrolled_step
04652     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04653   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
04654                      GTK_WIDGET (window->spin_step));
04655   g_signal_connect
04656     (window->spin_step, "value-changed", window_step_variable, NULL);
04657   window->grid_variable = (GtkGrid *) gtk_grid_new ();
04658   gtk_grid_attach (window->grid_variable,
04659                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
04660   gtk_grid_attach (window->grid_variable,
04661                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
04662   gtk_grid_attach (window->grid_variable,
04663                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
04664   gtk_grid_attach (window->grid_variable,
04665                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
04666   gtk_grid_attach (window->grid_variable,
04667                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
04668   gtk_grid_attach (window->grid_variable,
04669                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
04670   gtk_grid_attach (window->grid_variable,
04671                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04672   gtk_grid_attach (window->grid_variable,
04673                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04674   gtk_grid_attach (window->grid_variable,
04675                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04676   gtk_grid_attach (window->grid_variable,
04677                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
04678   gtk_grid_attach (window->grid_variable,
04679                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
04680   gtk_grid_attach (window->grid_variable,
04681                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04682   gtk_grid_attach (window->grid_variable,
04683                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
04684   gtk_grid_attach (window->grid_variable,
04685                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
04686   gtk_grid_attach (window->grid_variable,
04687                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
04688   gtk_grid_attach (window->grid_variable,
04689                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04690   gtk_grid_attach (window->grid_variable,
04691                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04692   gtk_grid_attach (window->grid_variable,
04693                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04694   gtk_grid_attach (window->grid_variable,
04695                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04696   gtk_grid_attach (window->grid_variable,
04697                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
04698   gtk_grid_attach (window->grid_variable,
04699                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
04700   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
04701   gtk_container_add (GTK_CONTAINER (window->frame_variable),
04702                      GTK_WIDGET (window->grid_variable));
04703
04704   // Creating the experiment widgets
04705   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
04706   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
04707                                gettext ("Experiment selector"));
04708   window->id_experiment = g_signal_connect
04709     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
04710   window->button_add_experiment
04711     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04712                                                    GTK_ICON_SIZE_BUTTON);
04713   g_signal_connect
04714     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
04715   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04716                                gettext ("Add experiment"));
04717   window->button_remove_experiment
04718     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04719                                                    GTK_ICON_SIZE_BUTTON);
04720   g_signal_connect (window->button_remove_experiment, "clicked",
04721                     window_remove_experiment, NULL);
04722   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04723                                gettext ("Remove experiment"));
04724   window->label_experiment
```

```
04725     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04726   window->button_experiment = (GtkFileChooserButton *)
04727     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04728                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04729   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04730                                gettext ("Experimental data file"));
04731   window->id_experiment_name
04732     = g_signal_connect (window->button_experiment, "selection-changed",
04733                         window_name_experiment, NULL);
04734   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
04735   window->spin_weight
04736     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04737   gtk_widget_set_tooltip_text
04738     (GTK_WIDGET (window->spin_weight),
04739      gettext ("Weight factor to build the objective function"));
04740   g_signal_connect
04741     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
04742   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04743   gtk_grid_attach (window->grid_experiment,
04744                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
04745   gtk_grid_attach (window->grid_experiment,
04746                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
04747   gtk_grid_attach (window->grid_experiment,
04748                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
04749   gtk_grid_attach (window->grid_experiment,
04750                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
04751   gtk_grid_attach (window->grid_experiment,
04752                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
04753   gtk_grid_attach (window->grid_experiment,
04754                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04755   gtk_grid_attach (window->grid_experiment,
04756                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04757   for (i = 0; i < MAX_NINPUTS; ++i)
04758     {
04759       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04760       window->check_template[i] = (GtkCheckButton *)
04761         gtk_check_button_new_with_label (buffer3);
04762       window->id_template[i]
04763         = g_signal_connect (window->check_template[i], "toggled",
04764                             window_inputs_experiment, NULL);
04765       gtk_grid_attach (window->grid_experiment,
04766                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
04767       window->button_template[i] = (GtkFileChooserButton *)
04768         gtk_file_chooser_button_new (gettext ("Input template"),
04769                                      GTK_FILE_CHOOSER_ACTION_OPEN);
04770       gtk_widget_set_tooltip_text
04771         (GTK_WIDGET (window->button_template[i]),
04772          gettext ("Experimental input template file"));
04773       window->id_input[i]
04774         = g_signal_connect_swapped (window->button_template[i],
04775                                     "selection-changed",
04776                                     (void (*)) window_template_experiment,
04777                                     (void *) (size_t) i);
04778       gtk_grid_attach (window->grid_experiment,
04779                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
04780     }
04781   window->frame_experiment
04782     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04783   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04784                      GTK_WIDGET (window->grid_experiment));
04785
04786   // Creating the grid and attaching the widgets to the grid
04787   window->grid = (GtkGrid *) gtk_grid_new ();
04788   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04789   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04790   gtk_grid_attach (window->grid,
04791                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04792   gtk_grid_attach (window->grid,
04793                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04794   gtk_grid_attach (window->grid,
04795                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04796   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
04797
04798   // Setting the window logo
04799   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04800   gtk_window_set_icon (window->window, window->logo);
04801
04802   // Showing the window
04803   gtk_widget_show_all (GTK_WIDGET (window->window));
04804
04805   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
04806 #if GTK_MINOR_VERSION >= 16
04807   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04808   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04809   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
```

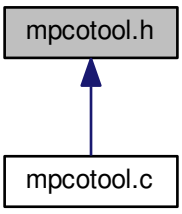
```
04810   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04811   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
04812 #endif
04813
04814   // Reading initial example
04815   input_new ();
04816   buffer2 = g_get_current_dir ();
04817   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04818   g_free (buffer2);
04819   window_read (buffer);
04820   g_free (buffer);
04821
04822 #if DEBUG
04823   fprintf (stderr, "window_new: start\n");
04824 #endif
04825 }
04826
04827 #endif
04828
04834 int
04835 cores_number ()
04836 {
04837 #ifdef G_OS_WIN32
04838   SYSTEM_INFO sysinfo;
04839   GetSystemInfo (&sysinfo);
04840   return sysinfo.dwNumberOfProcessors;
04841 #else
04842   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04843 #endif
04844 }
04845
04855 int
04856 main (int argn, char **argc)
04857 {
04858 #if HAVE_GTK
04859   char *buffer;
04860 #endif
04861
04862   // Starting pseudo-random numbers generator
04863   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04864   calibrate->seed = DEFAULT_RANDOM_SEED;
04865
04866   // Allowing spaces in the XML data file
04867   xmlKeepBlanksDefault (0);
04868
04869   // Starting MPI
04870 #if HAVE_MPI
04871   MPI_Init (&argn, &argc);
04872   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04873   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04874   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04875 #else
04876   ntasks = 1;
04877 #endif
04878
04879 #if HAVE_GTK
04880
04881   // Getting threads number
04882   nthreads_gradient = nthreads = cores_number ();
04883
04884   // Setting local language and international floating point numbers notation
04885   setlocale (LC_ALL, "");
04886   setlocale (LC_NUMERIC, "C");
04887   window->application_directory = g_get_current_dir ();
04888   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04889   bindtextdomain (PROGRAM_INTERFACE, buffer);
04890   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04891   textdomain (PROGRAM_INTERFACE);
04892
04893   // Initing GTK+
04894   gtk_disable_setlocale ();
04895   gtk_init (&argn, &argc);
04896
04897   // Opening the main window
04898   window_new ();
04899   gtk_main ();
04900
04901   // Freeing memory
04902   input_free ();
04903   g_free (buffer);
04904   gtk_widget_destroy (GTK_WIDGET (window->window));
04905   g_free (window->application_directory);
04906
04907 #else
04908
04909   // Checking syntax
```

```
04910   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04911     {
04912       printf ("The syntax is:\nmpcotoolbin [-nthreads x] data_file\n");
04913       return 1;
04914     }
04915
04916   // Getting threads number
04917   if (argn == 2)
04918     nthreads_gradient = nthreads = cores_number ();
04919   else
04920     {
04921       nthreads_gradient = nthreads = atoi (argc[2]);
04922       if (!nthreads)
04923         {
04924           printf ("Bad threads number\n");
04925           return 2;
04926         }
04927     }
04928   printf ("nthreads=%u\n", nthreads);
04929
04930   // Making calibration
04931   if (input_open (argc[argn - 1]))
04932     calibrate_open ();
04933
04934   // Freeing memory
04935   calibrate_free ();
04936
04937 #endif
04938
04939   // Closing MPI
04940 #if HAVE_MPI
04941   MPI_Finalize ();
04942 #endif
04943
04944   // Freeing memory
04945   gsl_rng_free (calibrate->rng);
04946
04947   // Closing
04948   return 0;
04949 }
```

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.
- 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.
- 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 1421 of file mpcotool.c.

```
01422 {
01423   unsigned int i, j;
01424   double e;
01425 #if DEBUG
01426   fprintf (stderr, "calibrate_best: start\n");
01427   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01428            calibrate->nsaveds, calibrate->nbest);
01429 #endif
01430   if (calibrate->nsaveds < calibrate->nbest
01431       || value < calibrate->error_best[calibrate->nsaveds - 1])
01432     {
01433       if (calibrate->nsaveds < calibrate->nbest)
01434         ++calibrate->nsaveds;
01435       calibrate->error_best[calibrate->nsaveds - 1] = value;
01436       calibrate->simulation_best[calibrate->
```

```
      nsaveds - 1] = simulation;
01437       for (i = calibrate->nsaveds; --i;)
01438         {
01439           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01440             {
01441               j = calibrate->simulation_best[i];
01442               e = calibrate->error_best[i];
01443               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01444               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01445               calibrate->simulation_best[i - 1] = j;
01446               calibrate->error_best[i - 1] = e;
01447             }
01448           else
01449             break;
01450         }
01451     }
01452 #if DEBUG
01453   fprintf (stderr, "calibrate_best: end\n");
01454 #endif
01455 }
```

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 1734 of file mpcotool.c.

```
01735 {
01736 #if DEBUG
01737   fprintf (stderr, "calibrate_best_gradient: start\n");
01738   fprintf (stderr,
01739            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01740            simulation, value, calibrate->error_best[0]);
01741 #endif
01742   if (value < calibrate->error_best[0])
01743     {
01744       calibrate->error_best[0] = value;
01745       calibrate->simulation_best[0] = simulation;
01746 #if DEBUG
01747       fprintf (stderr,
01748                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01749                simulation, value);
01750 #endif
01751     }
01752 #if DEBUG
01753   fprintf (stderr, "calibrate_best_gradient: end\n");
01754 #endif
01755 }
```

5.7.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

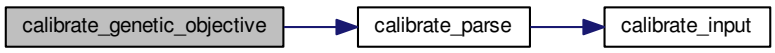
Returns

objective function value.

Definition at line 2037 of file mpcotool.c.

```
02038 {
02039   unsigned int j;
02040   double objective;
02041   char buffer[64];
02042 #if DEBUG
02043   fprintf (stderr, "calibrate_genetic_objective: start\n");
02044 #endif
02045   for (j = 0; j < calibrate->nvariables; ++j)
02046     {
02047       calibrate->value[entity->id * calibrate->nvariables + j]
02048         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02049     }
02050   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02051     objective += calibrate_parse (entity->id, j);
02052   g_mutex_lock (mutex);
02053   for (j = 0; j < calibrate->nvariables; ++j)
02054     {
02055       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02056       fprintf (calibrate->file_variables, buffer,
02057                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
02058     }
02059   fprintf (calibrate->file_variables, "%.14le\n", objective);
02060   g_mutex_unlock (mutex);
02061 #if DEBUG
02062   fprintf (stderr, "calibrate_genetic_objective: end\n");
02063 #endif
02064   return objective;
02065 }
```

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

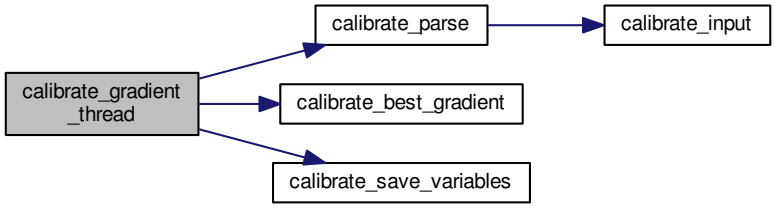
NULL

Definition at line 1799 of file mpcotool.c.

```
01800 {
01801   unsigned int i, j, thread;
01802   double e;
01803 #if DEBUG
01804   fprintf (stderr, "calibrate_gradient_thread: start\n");
01805 #endif
01806   thread = data->thread;
01807 #if DEBUG
01808   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01809            thread,
01810            calibrate->thread_gradient[thread],
01811            calibrate->thread_gradient[thread + 1]);
01812 #endif
01813   for (i = calibrate->thread_gradient[thread];
01814        i < calibrate->thread_gradient[thread + 1]; ++i)
01815     {
01816       e = 0.;
```

```
01817       for (j = 0; j < calibrate->nexperiments; ++j)
01818         e += calibrate_parse (i, j);
01819       g_mutex_lock (mutex);
01820       calibrate_best_gradient (i, e);
01821       calibrate_save_variables (i, e);
01822       g_mutex_unlock (mutex);
01823 #if DEBUG
01824       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01825 #endif
01826     }
01827 #if DEBUG
01828   fprintf (stderr, "calibrate_gradient_thread: end\n");
01829 #endif
01830   g_thread_exit (NULL);
01831   return NULL;
01832 }
```

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 1174 of file mpcotool.c.

```
01175 {
01176   unsigned int i;
01177   char buffer[32], value[32], *buffer2, *buffer3, *content;
01178   FILE *file;
01179   gsize length;
01180   GRegex *regex;
01181
01182 #if DEBUG
01183   fprintf (stderr, "calibrate_input: start\n");
01184 #endif
01185
01186   // Checking the file
01187   if (!template)
01188     goto calibrate_input_end;
01189
01190   // Opening template
01191   content = g_mapped_file_get_contents (template);
01192   length = g_mapped_file_get_length (template);
01193 #if DEBUG
01194   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01195            content);
01196 #endif
01197   file = g_fopen (input, "w");
01198
01199   // Parsing template
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
```

```
01202 #if DEBUG
01203       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01204 #endif
01205       snprintf (buffer, 32, "@variable%u@", i + 1);
01206       regex = g_regex_new (buffer, 0, 0, NULL);
01207       if (i == 0)
01208         {
01209           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01210                                              calibrate->label[i], 0, NULL);
01211 #if DEBUG
01212           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01213 #endif
01214         }
01215       else
01216         {
01217           length = strlen (buffer3);
01218           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01219                                              calibrate->label[i], 0, NULL);
01220           g_free (buffer3);
01221         }
01222       g_regex_unref (regex);
01223       length = strlen (buffer2);
01224       snprintf (buffer, 32, "@value%u@", i + 1);
01225       regex = g_regex_new (buffer, 0, 0, NULL);
01226       snprintf (value, 32, format[calibrate->precision[i]],
01227                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01228
01229 #if DEBUG
01230       fprintf (stderr, "calibrate_input: value=%s\n", value);
01231 #endif
01232       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01233                                          0, NULL);
01234       g_free (buffer2);
01235       g_regex_unref (regex);
01236     }
01237
01238   // Saving input file
01239   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01240   g_free (buffer3);
01241   fclose (file);
01242
01243 calibrate_input_end:
01244 #if DEBUG
01245   fprintf (stderr, "calibrate_input: end\n");
01246 #endif
01247   return;
01248 }
```

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 1539 of file mpcotool.c.

```
01541 {
01542   unsigned int i, j, k, s[calibrate->nbest];
01543   double e[calibrate->nbest];
01544 #if DEBUG
01545   fprintf (stderr, "calibrate_merge: start\n");
01546 #endif
01547   i = j = k = 0;
01548   do
01549     {
01550       if (i == calibrate->nsaveds)
01551         {
01552           s[k] = simulation_best[j];
01553           e[k] = error_best[j];
01554           ++j;
01555           ++k;
01556           if (j == nsaveds)
01557             break;
01558         }
01559       else if (j == nsaveds)
```

```
01560         {
01561           s[k] = calibrate->simulation_best[i];
01562           e[k] = calibrate->error_best[i];
01563           ++i;
01564           ++k;
01565           if (i == calibrate->nsaveds)
01566             break;
01567         }
01568       else if (calibrate->error_best[i] > error_best[j])
01569         {
01570           s[k] = simulation_best[j];
01571           e[k] = error_best[j];
01572           ++j;
01573           ++k;
01574         }
01575       else
01576         {
01577           s[k] = calibrate->simulation_best[i];
01578           e[k] = calibrate->error_best[i];
01579           ++i;
01580           ++k;
01581         }
01582     }
01583   while (k < calibrate->nbest);
01584   calibrate->nsaveds = k;
01585   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01586   memcpy (calibrate->error_best, e, k * sizeof (double));
01587 #if DEBUG
01588   fprintf (stderr, "calibrate_merge: end\n");
01589 #endif
01590 }
```

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 1261 of file mpcotool.c.

```
01262 {
01263   unsigned int i;
01264   double e;
01265   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01266     *buffer3, *buffer4;
01267   FILE *file_result;
01268
01269 #if DEBUG
01270   fprintf (stderr, "calibrate_parse: start\n");
01271   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01272            experiment);
01273 #endif
01274
01275   // Opening input files
01276   for (i = 0; i < calibrate->ninputs; ++i)
01277     {
01278       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01279 #if DEBUG
01280       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01281 #endif
01282       calibrate_input (simulation, &input[i][0],
01283                        calibrate->file[i][experiment]);
01284     }
01285   for (; i < MAX_NINPUTS; ++i)
01286     strcpy (&input[i][0], "");
01287 #if DEBUG
01288   fprintf (stderr, "calibrate_parse: parsing end\n");
01289 #endif
01290
01291   // Performing the simulation
01292   snprintf (output, 32, "output-%u-%u", simulation, experiment);
```

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

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 1393 of file mpcotool.c.

```
01394 {
01395   unsigned int i;
01396   char buffer[64];
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_save_variables: start\n");
01399 #endif
01400   for (i = 0; i < calibrate->nvariables; ++i)
01401     {
01402       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01403       fprintf (calibrate->file_variables, buffer,
01404                calibrate->value[simulation * calibrate->
      nvariables + i]);
01405     }
01406   fprintf (calibrate->file_variables, "%.14le\n", error);
01407 #if DEBUG
01408   fprintf (stderr, "calibrate_save_variables: end\n");
01409 #endif
01410 }
```

5.7.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

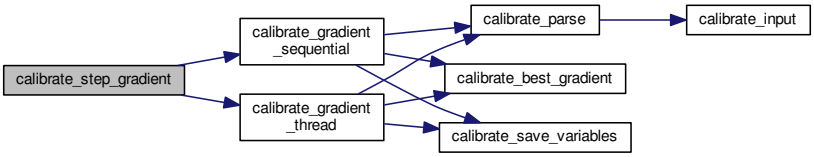
simulation	Simulation number.

Definition at line 1901 of file mpcotool.c.

```
01902 {
01903   GThread *thread[nthreads_gradient];
01904   ParallelData data[nthreads_gradient];
01905   unsigned int i, j, k, b;
01906 #if DEBUG
01907   fprintf (stderr, "calibrate_step_gradient: start\n");
01908 #endif
01909   for (i = 0; i < calibrate->nestimates; ++i)
01910     {
01911       k = (simulation + i) * calibrate->nvariables;
01912       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01913 #if DEBUG
01914       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01915                simulation + i, calibrate->simulation_best[0]);
01916 #endif
01917       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01918         {
01919 #if DEBUG
01920           fprintf (stderr,
01921                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01922                    i, j, calibrate->value[b]);
01923 #endif
01924           calibrate->value[k]
01925             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01926           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01927                                             calibrate->rangeminabs[j]),
01928                                       calibrate->rangemaxabs[j]);
01929 #if DEBUG
01930           fprintf (stderr,
01931                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01932                    i, j, calibrate->value[k]);
01933 #endif
01934         }
01935     }
01936   if (nthreads_gradient == 1)
01937     calibrate_gradient_sequential (simulation);
01938   else
01939     {
01940       for (i = 0; i <= nthreads_gradient; ++i)
01941         {
```

```
01942           calibrate->thread_gradient[i]
01943             = simulation + calibrate->nstart_gradient
01944             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01945               / nthreads_gradient;
01946 #if DEBUG
01947           fprintf (stderr,
01948                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01949                    i, calibrate->thread_gradient[i]);
01950 #endif
01951         }
01952       for (i = 0; i < nthreads_gradient; ++i)
01953         {
01954           data[i].thread = i;
01955           thread[i] = g_thread_new
01956             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01957         }
01958       for (i = 0; i < nthreads_gradient; ++i)
01959         g_thread_join (thread[i]);
01960     }
01961 #if DEBUG
01962   fprintf (stderr, "calibrate_step_gradient: end\n");
01963 #endif
01964 }
```

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

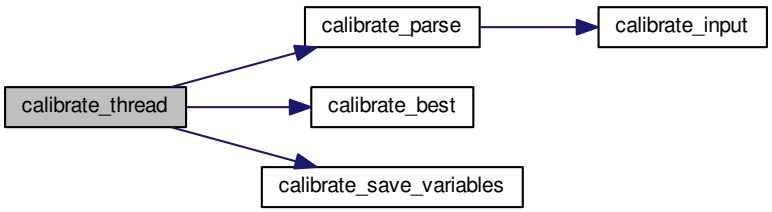
NULL

Definition at line 1495 of file mpcotool.c.

```
01496 {
01497   unsigned int i, j, thread;
01498   double e;
01499 #if DEBUG
01500   fprintf (stderr, "calibrate_thread: start\n");
01501 #endif
01502   thread = data->thread;
01503 #if DEBUG
01504   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01505            calibrate->thread[thread], calibrate->thread[thread + 1]);
01506 #endif
01507   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01508     {
01509       e = 0.;
01510       for (j = 0; j < calibrate->nexperiments; ++j)
01511         e += calibrate_parse (i, j);
01512       g_mutex_lock (mutex);
01513       calibrate_best (i, e);
01514       calibrate_save_variables (i, e);
01515       g_mutex_unlock (mutex);
```

```
01516 #if DEBUG
01517       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01518 #endif
01519     }
01520 #if DEBUG
01521   fprintf (stderr, "calibrate_thread: end\n");
01522 #endif
01523   g_thread_exit (NULL);
01524   return NULL;
01525 }
```

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 489 of file mpcotool.c.

```
00490 {
00491   char buffer2[64];
00492   char *buffert[MAX_NINPUTS] =
00493     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00494   xmlDoc *doc;
00495   xmlNode *node, *child;
00496   xmlChar *buffer;
00497   char *msg;
00498   int error_code;
00499   unsigned int i;
00500
00501 #if DEBUG
00502   fprintf (stderr, "input_open: start\n");
00503 #endif
00504
00505   // Resetting input data
00506   buffer = NULL;
00507   input_new ();
00508
00509   // Parsing the input file
00510 #if DEBUG
00511   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00512 #endif
00513   doc = xmlParseFile (filename);
00514   if (!doc)
00515     {
00516       msg = gettext ("Unable to parse the input file");
00517       goto exit_on_error;
00518     }
```

```
00519
00520   // Getting the root node
00521 #if DEBUG
00522   fprintf (stderr, "input_open: getting the root node\n");
00523 #endif
00524   node = xmlDocGetRootElement (doc);
00525   if (xmlStrcmp (node->name, XML_CALIBRATE))
00526     {
00527       msg = gettext ("Bad root XML node");
00528       goto exit_on_error;
00529     }
00530
00531   // Getting results file names
00532   input->result = (char *) xmlGetProp (node, XML_RESULT);
00533   if (!input->result)
00534     input->result = (char *) xmlStrdup (result_name);
00535   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00536   if (!input->variables)
00537     input->variables = (char *) xmlStrdup (variables_name);
00538
00539   // Opening simulator program name
00540   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00541   if (!input->simulator)
00542     {
00543       msg = gettext ("Bad simulator program");
00544       goto exit_on_error;
00545     }
00546
00547   // Opening evaluator program name
00548   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00549
00550   // Obtaining pseudo-random numbers generator seed
00551   if (!xmlHasProp (node, XML_SEED))
00552     input->seed = DEFAULT_RANDOM_SEED;
00553   else
00554     {
00555       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00556       if (error_code)
00557         {
00558           msg = gettext ("Bad pseudo-random numbers generator seed");
00559           goto exit_on_error;
00560         }
00561     }
00562
00563   // Opening algorithm
00564   buffer = xmlGetProp (node, XML_ALGORITHM);
00565   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00566     {
00567       input->algorithm = ALGORITHM_MONTE_CARLO;
00568
00569       // Obtaining simulations number
00570       input->nsimulations
00571         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00572       if (error_code)
00573         {
00574           msg = gettext ("Bad simulations number");
00575           goto exit_on_error;
00576         }
00577     }
00578   else if (!xmlStrcmp (buffer, XML_SWEEP))
00579     input->algorithm = ALGORITHM_SWEEP;
00580   else if (!xmlStrcmp (buffer, XML_GENETIC))
00581     {
00582       input->algorithm = ALGORITHM_GENETIC;
00583
00584       // Obtaining population
00585       if (xmlHasProp (node, XML_NPOPULATION))
00586         {
00587           input->nsimulations
00588             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00589           if (error_code || input->nsimulations < 3)
00590             {
00591               msg = gettext ("Invalid population number");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No population number");
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining generations
00602       if (xmlHasProp (node, XML_NGENERATIONS))
00603         {
00604           input->niterations
00605             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
```

```
00606           if (error_code || !input->niterations)
00607             {
00608               msg = gettext ("Invalid generations number");
00609               goto exit_on_error;
00610             }
00611         }
00612       else
00613         {
00614           msg = gettext ("No generations number");
00615           goto exit_on_error;
00616         }
00617
00618       // Obtaining mutation probability
00619       if (xmlHasProp (node, XML_MUTATION))
00620         {
00621           input->mutation_ratio
00622             = xml_node_get_float (node, XML_MUTATION, &error_code);
00623           if (error_code || input->mutation_ratio < 0.
00624               || input->mutation_ratio >= 1.)
00625             {
00626               msg = gettext ("Invalid mutation probability");
00627               goto exit_on_error;
00628             }
00629         }
00630       else
00631         {
00632           msg = gettext ("No mutation probability");
00633           goto exit_on_error;
00634         }
00635
00636       // Obtaining reproduction probability
00637       if (xmlHasProp (node, XML_REPRODUCTION))
00638         {
00639           input->reproduction_ratio
00640             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00641           if (error_code || input->reproduction_ratio < 0.
00642               || input->reproduction_ratio >= 1.0)
00643             {
00644               msg = gettext ("Invalid reproduction probability");
00645               goto exit_on_error;
00646             }
00647         }
00648       else
00649         {
00650           msg = gettext ("No reproduction probability");
00651           goto exit_on_error;
00652         }
00653
00654       // Obtaining adaptation probability
00655       if (xmlHasProp (node, XML_ADAPTATION))
00656         {
00657           input->adaptation_ratio
00658             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00659           if (error_code || input->adaptation_ratio < 0.
00660               || input->adaptation_ratio >= 1.)
00661             {
00662               msg = gettext ("Invalid adaptation probability");
00663               goto exit_on_error;
00664             }
00665         }
00666       else
00667         {
00668           msg = gettext ("No adaptation probability");
00669           goto exit_on_error;
00670         }
00671
00672       // Checking survivals
00673       i = input->mutation_ratio * input->nsimulations;
00674       i += input->reproduction_ratio * input->
      nsimulations;
00675       i += input->adaptation_ratio * input->
      nsimulations;
00676       if (i > input->nsimulations - 2)
00677         {
00678           msg = gettext
00679             ("No enough survival entities to reproduce the population");
00680           goto exit_on_error;
00681         }
00682     }
00683   else
00684     {
00685       msg = gettext ("Unknown algorithm");
00686       goto exit_on_error;
00687     }
00688   xmlFree (buffer);
00689   buffer = NULL;
00690
```

```
00691   if (input->algorithm == ALGORITHM_MONTE_CARLO
00692       || input->algorithm == ALGORITHM_SWEEP)
00693     {
00694
00695       // Obtaining iterations number
00696       input->niterations
00697         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00698       if (error_code == 1)
00699         input->niterations = 1;
00700       else if (error_code)
00701         {
00702           msg = gettext ("Bad iterations number");
00703           goto exit_on_error;
00704         }
00705
00706       // Obtaining best number
00707       if (xmlHasProp (node, XML_NBEST))
00708         {
00709           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00710           if (error_code || !input->nbest)
00711             {
00712               msg = gettext ("Invalid best number");
00713               goto exit_on_error;
00714             }
00715         }
00716       else
00717         input->nbest = 1;
00718
00719       // Obtaining tolerance
00720       if (xmlHasProp (node, XML_TOLERANCE))
00721         {
00722           input->tolerance
00723             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00724           if (error_code || input->tolerance < 0.)
00725             {
00726               msg = gettext ("Invalid tolerance");
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         input->tolerance = 0.;
00732
00733       // Getting gradient method parameters
00734       if (xmlHasProp (node, XML_NSTEPS))
00735         {
00736           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00737           if (error_code || !input->nsteps)
00738             {
00739               msg = gettext ("Invalid steps number");
00740               goto exit_on_error;
00741             }
00742           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00743           if (!xmlStrcmp (buffer, XML_COORDINATES))
00744             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00745           else if (!xmlStrcmp (buffer, XML_RANDOM))
00746             {
00747               input->gradient_method =
      GRADIENT_METHOD_RANDOM;
00748               input->nestimates
00749                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00750               if (error_code || !input->nestimates)
00751                 {
00752                   msg = gettext ("Invalid estimates number");
00753                   goto exit_on_error;
00754                 }
00755             }
00756           else
00757             {
00758               msg = gettext ("Unknown method to estimate the gradient");
00759               goto exit_on_error;
00760             }
00761           xmlFree (buffer);
00762           buffer = NULL;
00763           if (xmlHasProp (node, XML_RELAXATION))
00764             {
00765               input->relaxation
00766                 = xml_node_get_float (node, XML_RELAXATION, &error_code);
00767               if (error_code || input->relaxation < 0.
00768                   || input->relaxation > 2.)
00769                 {
00770                   msg = gettext ("Invalid relaxation parameter");
00771                   goto exit_on_error;
00772                 }
00773             }
```

```
00774           else
00775             input->relaxation = DEFAULT_RELAXATION;
00776         }
00777       else
00778         input->nsteps = 0;
00779     }
00780
00781   // Reading the experimental data
00782   for (child = node->children; child; child = child->next)
00783     {
00784       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00785         break;
00786 #if DEBUG
00787       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00788 #endif
00789       if (xmlHasProp (child, XML_NAME))
00790         buffer = xmlGetProp (child, XML_NAME);
00791       else
00792         {
00793           snprintf (buffer2, 64, "%s %u: %s",
00794                     gettext ("Experiment"),
00795                     input->nexperiments + 1, gettext ("no data file name"));
00796           msg = buffer2;
00797           goto exit_on_error;
00798         }
00799 #if DEBUG
00800       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00801 #endif
00802       input->weight = g_realloc (input->weight,
00803                                  (1 + input->nexperiments) * sizeof (double));
00804       if (xmlHasProp (child, XML_WEIGHT))
00805         {
00806           input->weight[input->nexperiments]
00807             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00808           if (error_code)
00809             {
00810               snprintf (buffer2, 64, "%s %s: %s",
00811                         gettext ("Experiment"), buffer, gettext ("bad weight"));
00812               msg = buffer2;
00813               goto exit_on_error;
00814             }
00815         }
00816       else
00817         input->weight[input->nexperiments] = 1.;
00818 #if DEBUG
00819       fprintf (stderr, "input_open: weight=%lg\n",
00820                input->weight[input->nexperiments]);
00821 #endif
00822       if (!input->nexperiments)
00823         input->ninputs = 0;
00824 #if DEBUG
00825       fprintf (stderr, "input_open: template[0]\n");
00826 #endif
00827       if (xmlHasProp (child, XML_TEMPLATE1))
00828         {
00829           input->template[0]
00830             = (char **) g_realloc (input->template[0],
00831                                    (1 + input->nexperiments) * sizeof (char *));
00832           buffert[0] = (char *) xmlGetProp (child, template[0]);
00833 #if DEBUG
00834           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00835                    input->nexperiments, buffert[0]);
00836 #endif
00837           if (!input->nexperiments)
00838             ++input->ninputs;
00839 #if DEBUG
00840           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00841 #endif
00842         }
00843       else
00844         {
00845           snprintf (buffer2, 64, "%s %s: %s",
00846                     gettext ("Experiment"), buffer, gettext ("no template"));
00847           msg = buffer2;
00848           goto exit_on_error;
00849         }
00850       for (i = 1; i < MAX_NINPUTS; ++i)
00851         {
00852 #if DEBUG
00853           fprintf (stderr, "input_open: template%u\n", i + 1);
00854 #endif
00855           if (xmlHasProp (child, template[i]))
00856             {
00857               if (input->nexperiments && input->ninputs <= i)
00858                 {
00859                   snprintf (buffer2, 64, "%s %s: %s",
00860                             gettext ("Experiment"),
```

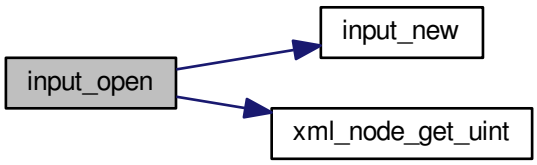
```
00861                           buffer, gettext ("bad templates number"));
00862                       msg = buffer2;
00863                       while (i-- > 0)
00864                         xmlFree (buffert[i]);
00865                       goto exit_on_error;
00866                     }
00867                   input->template[i] = (char **)
00868                     g_realloc (input->template[i],
00869                                (1 + input->nexperiments) * sizeof (char *));
00870                   buffert[i] = (char *) xmlGetProp (child, template[i]);
00871 #if DEBUG
00872                   fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00873                            input->nexperiments, i + 1,
00874                            input->template[i][input->nexperiments]);
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 if (input->nexperiments && input->ninputs >= i)
00883                 {
00884                   snprintf (buffer2, 64, "%s %s: %s%u",
00885                             gettext ("Experiment"),
00886                             buffer, gettext ("no template"), i + 1);
00887                   msg = buffer2;
00888                   while (i-- > 0)
00889                     xmlFree (buffert[i]);
00890                   goto exit_on_error;
00891                 }
00892               else
00893                 break;
00894             }
00895           input->experiment
00896             = g_realloc (input->experiment,
00897                          (1 + input->nexperiments) * sizeof (char *));
00898           input->experiment[input->nexperiments] = (char *) buffer;
00899           for (i = 0; i < input->ninputs; ++i)
00900             input->template[i][input->nexperiments] = buffert[i];
00901           ++input->nexperiments;
00902 #if DEBUG
00903           fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00904 #endif
00905         }
00906       if (!input->nexperiments)
00907         {
00908           msg = gettext ("No calibration experiments");
00909           goto exit_on_error;
00910         }
00911       buffer = NULL;
00912
00913       // Reading the variables data
00914       for (; child; child = child->next)
00915         {
00916           if (xmlStrcmp (child->name, XML_VARIABLE))
00917             {
00918               snprintf (buffer2, 64, "%s %u: %s",
00919                         gettext ("Variable"),
00920                         input->nvariables + 1, gettext ("bad XML node"));
00921               msg = buffer2;
00922               goto exit_on_error;
00923             }
00924           if (xmlHasProp (child, XML_NAME))
00925             buffer = xmlGetProp (child, XML_NAME);
00926           else
00927             {
00928               snprintf (buffer2, 64, "%s %u: %s",
00929                         gettext ("Variable"),
00930                         input->nvariables + 1, gettext ("no name"));
00931               msg = buffer2;
00932               goto exit_on_error;
00933             }
00934           if (xmlHasProp (child, XML_MINIMUM))
00935             {
00936               input->rangemin = g_realloc
00937                 (input->rangemin, (1 + input->nvariables) * sizeof (double));
00938               input->rangeminabs = g_realloc
00939                 (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00940               input->rangemin[input->nvariables]
00941                 = xml_node_get_float (child, XML_MINIMUM, &error_code);
00942               if (error_code)
00943                 {
00944                   snprintf (buffer2, 64, "%s %s: %s",
00945                             gettext ("Variable"), buffer, gettext ("bad minimum"));
00946                   msg = buffer2;
00947                   goto exit_on_error;
```

```
00948             }
00949           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00950             {
00951               input->rangeminabs[input->nvariables]
00952                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00953               if (error_code)
00954                 {
00955                   snprintf (buffer2, 64, "%s %s: %s",
00956                             gettext ("Variable"),
00957                             buffer, gettext ("bad absolute minimum"));
00958                   msg = buffer2;
00959                   goto exit_on_error;
00960                 }
00961             }
00962           else
00963             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00964           if (input->rangemin[input->nvariables]
00965               < input->rangeminabs[input->nvariables])
00966             {
00967               snprintf (buffer2, 64, "%s %s: %s",
00968                         gettext ("Variable"),
00969                         buffer, gettext ("minimum range not allowed"));
00970               msg = buffer2;
00971               goto exit_on_error;
00972             }
00973         }
00974       else
00975         {
00976           snprintf (buffer2, 64, "%s %s: %s",
00977                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00978           msg = buffer2;
00979           goto exit_on_error;
00980         }
00981       if (xmlHasProp (child, XML_MAXIMUM))
00982         {
00983           input->rangemax = g_realloc
00984             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00985           input->rangemaxabs = g_realloc
00986             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00987           input->rangemax[input->nvariables]
00988             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00989           if (error_code)
00990             {
00991               snprintf (buffer2, 64, "%s %s: %s",
00992                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00993               msg = buffer2;
00994               goto exit_on_error;
00995             }
00996           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00997             {
00998               input->rangemaxabs[input->nvariables]
00999                 = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
01000               if (error_code)
01001                 {
01002                   snprintf (buffer2, 64, "%s %s: %s",
01003                             gettext ("Variable"),
01004                             buffer, gettext ("bad absolute maximum"));
01005                   msg = buffer2;
01006                   goto exit_on_error;
01007                 }
01008             }
01009           else
01010             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01011           if (input->rangemax[input->nvariables]
01012               > input->rangemaxabs[input->nvariables])
01013             {
01014               snprintf (buffer2, 64, "%s %s: %s",
01015                         gettext ("Variable"),
01016                         buffer, gettext ("maximum range not allowed"));
01017               msg = buffer2;
01018               goto exit_on_error;
01019             }
01020         }
01021       else
01022         {
01023           snprintf (buffer2, 64, "%s %s: %s",
01024                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01025           msg = buffer2;
01026           goto exit_on_error;
01027         }
01028       if (input->rangemax[input->nvariables]
01029           < input->rangemin[input->nvariables])
01030         {
01031           snprintf (buffer2, 64, "%s %s: %s",
01032                     gettext ("Variable"), buffer, gettext ("bad range"));
```

```
01033           msg = buffer2;
01034           goto exit_on_error;
01035         }
01036       input->precision = g_realloc
01037         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01038       if (xmlHasProp (child, XML_PRECISION))
01039         {
01040           input->precision[input->nvariables]
01041             = xml_node_get_uint (child, XML_PRECISION, &error_code);
01042           if (error_code || input->precision[input->
      nvariables] >= NPRECISIONS)
01043             {
01044               snprintf (buffer2, 64, "%s %s: %s",
01045                         gettext ("Variable"),
01046                         buffer, gettext ("bad precision"));
01047               msg = buffer2;
01048               goto exit_on_error;
01049             }
01050         }
01051       else
01052         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
01053       if (input->algorithm == ALGORITHM_SWEEP)
01054         {
01055           if (xmlHasProp (child, XML_NSWEEPS))
01056             {
01057               input->nsweeps = (unsigned int *)
01058                 g_realloc (input->nsweeps,
01059                            (1 + input->nvariables) * sizeof (unsigned int));
01060               input->nsweeps[input->nvariables]
01061                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01062               if (error_code || !input->nsweeps[input->
      nvariables])
01063                 {
01064                   snprintf (buffer2, 64, "%s %s: %s",
01065                             gettext ("Variable"),
01066                             buffer, gettext ("bad sweeps"));
01067                   msg = buffer2;
01068                   goto exit_on_error;
01069                 }
01070             }
01071           else
01072             {
01073               snprintf (buffer2, 64, "%s %s: %s",
01074                         gettext ("Variable"),
01075                         buffer, gettext ("no sweeps number"));
01076               msg = buffer2;
01077               goto exit_on_error;
01078             }
01079 #if DEBUG
01080           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01081                    input->nsweeps[input->nvariables],
      input->nsimulations);
01082 #endif
01083         }
01084       if (input->algorithm == ALGORITHM_GENETIC)
01085         {
01086           // Obtaining bits representing each variable
01087           if (xmlHasProp (child, XML_NBITS))
01088             {
01089               input->nbits = (unsigned int *)
01090                 g_realloc (input->nbits,
01091                            (1 + input->nvariables) * sizeof (unsigned int));
01092               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01093               if (error_code || !i)
01094                 {
01095                   snprintf (buffer2, 64, "%s %s: %s",
01096                             gettext ("Variable"),
01097                             buffer, gettext ("invalid bits number"));
01098                   msg = buffer2;
01099                   goto exit_on_error;
01100                 }
01101               input->nbits[input->nvariables] = i;
01102             }
01103           else
01104             {
01105               snprintf (buffer2, 64, "%s %s: %s",
01106                         gettext ("Variable"),
01107                         buffer, gettext ("no bits number"));
01108               msg = buffer2;
01109               goto exit_on_error;
01110             }
01111         }
01112       else if (input->nsteps)
01113         {
01114           input->step = (double *)
01115             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
```

```
01116           input->step[input->nvariables]
01117             = xml_node_get_float (child, XML_STEP, &error_code);
01118           if (error_code || input->step[input->nvariables] < 0.)
01119             {
01120               snprintf (buffer2, 64, "%s %s: %s",
01121                         gettext ("Variable"),
01122                         buffer, gettext ("bad step size"));
01123               msg = buffer2;
01124               goto exit_on_error;
01125             }
01126         }
01127       input->label = g_realloc
01128         (input->label, (1 + input->nvariables) * sizeof (char *));
01129       input->label[input->nvariables] = (char *) buffer;
01130       ++input->nvariables;
01131     }
01132   if (!input->nvariables)
01133     {
01134       msg = gettext ("No calibration variables");
01135       goto exit_on_error;
01136     }
01137   buffer = NULL;
01138
01139   // Getting the working directory
01140   input->directory = g_path_get_dirname (filename);
01141   input->name = g_path_get_basename (filename);
01142
01143   // Closing the XML document
01144   xmlFreeDoc (doc);
01145
01146 #if DEBUG
01147   fprintf (stderr, "input_open: end\n");
01148 #endif
01149   return 1;
01150
01151 exit_on_error:
01152   xmlFree (buffer);
01153   xmlFreeDoc (doc);
01154   show_error (msg);
01155   input_free ();
01156 #if DEBUG
01157   fprintf (stderr, "input_open: end\n");
01158 #endif
01159   return 0;
01160 }
```

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.

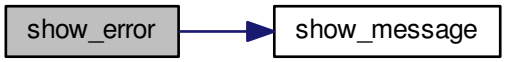
Parameters

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 337 of file mpcotool.c.

```
00338 {
00339   double x = 0.;
00340   xmlChar *buffer;
00341   buffer = xmlGetProp (node, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350       xmlFree (buffer);
00351     }
00352   return x;
00353 }
```

5.7.3.15 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 }
```

5.7.3.16 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 }
```

5.7.3.17 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 404 of file mpcotool.c.

```
00405 {
00406   xmlChar buffer[64];
00407   snprintf ((char *) buffer, 64, "%.14lg", value);
00408   xmlSetProp (node, prop, buffer);
00409 }
```

5.7.3.18 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 366 of file mpcotool.c.

```
00367 {
00368   xmlChar buffer[64];
00369   snprintf ((char *) buffer, 64, "%d", value);
00370   xmlSetProp (node, prop, buffer);
00371 }
```

5.7.3.19 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 385 of file mpcotool.c.

```
00386 {
00387   xmlChar buffer[64];
00388   snprintf ((char *) buffer, 64, "%u", value);
00389   xmlSetProp (node, prop, buffer);
00390 }
```

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

Index