Calibrator

1.3.6

Generated by Doxygen 1.8.9.1

Wed Nov 25 2015 12:29:31

Contents

Chapter 1

CALIBRATOR

A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 1.0.6: Stable and recommended version.
- 1.3.6: 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:

- 1.0.6/configure.ac: configure generator.
- 1.0.6/Makefile.in: Makefile generator.
- 1.0.6/config.h.in: config header generator.
- 1.0.6/calibrator.c: main source code.
- 1.0.6/calibrator.h: main header code.
- 1.0.6/interface.h: interface header code.
- 1.0.6/build: script to build all.
- 1.0.6/logo.png: logo figure.
- 1.0.6/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/calibrator.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/calibrator.git`

3. Link the latest genetic version to genetic:

   $ cd calibrator/1.0.6
   $ 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 calibrator/1.0.6):

   $ 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 ../../1.0.6
   $ make tests

USER INSTRUCTIONS

- Command line in sequential mode:

  $ ./calibratorbin [-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] ./calibratorbin [-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:

  $ ./calibrator

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" 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"> ... <variable name="variable↩
_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="sweeps_number"
nbits="bits_number"> </calibrate> “‘

with:

- ∗"simulator"∗ simulator executable file name.
- ∗"evaluator"∗: Optional. When needed is the evaluator executable file name.
- ∗"result"∗: Optional. Is the name of the optime result file (default is "result").
- ∗"variables"∗: Optional. Is the name of all simulated variables file (default is "variables").
- ∗"precision"∗ defined for each variable. Number of precision digits to evaluate the variable. 0 apply for integer numbers.
- ∗"weight"∗ defined for each experiment. Multiplies the objective value obtained for each experiment in the final objective function value.
- ∗"seed"∗: Seed of the pseudo-random numbers generator.

Implemented algorithms are:

- ∗"sweep"∗: Sweep brute force algorithm. 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. 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).

- ∗"genetic"∗: Genetic algorithm. 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 <calibrator.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 calibrator.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 calibrator.h.

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

- calibrator.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 <calibrator.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 calibrator.h.

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

- calibrator.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 74 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 <calibrator.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 calibrator.h.

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

- calibrator.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 92 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.
- 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:

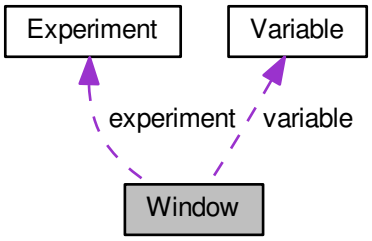
- interface.h

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.
- 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 102 of file interface.h.

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

- interface.h

Chapter 5

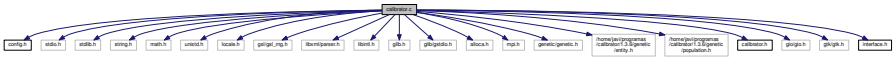
File Documentation

5.1 calibrator.c File Reference

Source file of the calibrator.

```
#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 "calibrator.h"
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "interface.h"
```

Include dependency graph for calibrator.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_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 calibrator.
- 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.

5.1.1 Detailed Description

Source file of the calibrator.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file calibrator.c.

5.1.2 Function Documentation

5.1.2.1 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 1423 of file calibrator.c.

```
01424 {
01425   unsigned int i, j;
01426   double e;
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_best: start\n");
01429   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01430            calibrate->nsaveds, calibrate->nbest);
01431 #endif
01432   if (calibrate->nsaveds < calibrate->nbest
01433       || value < calibrate->error_best[calibrate->nsaveds - 1])
01434     {
01435       if (calibrate->nsaveds < calibrate->nbest)
01436         ++calibrate->nsaveds;
01437       calibrate->error_best[calibrate->nsaveds - 1] = value;
01438       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01439       for (i = calibrate->nsaveds; --i;)
01440         {
01441           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01442             {
01443               j = calibrate->simulation_best[i];
01444               e = calibrate->error_best[i];
01445               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01446               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01447               calibrate->simulation_best[i - 1] = j;
01448               calibrate->error_best[i - 1] = e;
01449             }
01450           else
01451             break;
01452         }
01453     }
01454 #if DEBUG
01455   fprintf (stderr, "calibrate_best: end\n");
01456 #endif
01457 }
```

5.1.2.2 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 1736 of file calibrator.c.

```
01737 {
01738 #if DEBUG
01739   fprintf (stderr, "calibrate_best_gradient: start\n");
01740   fprintf (stderr,
01741            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01742            simulation, value, calibrate->error_best[0]);
01743 #endif
01744   if (value < calibrate->error_best[0])
01745     {
01746       calibrate->error_best[0] = value;
01747       calibrate->simulation_best[0] = simulation;
01748 #if DEBUG
01749       fprintf (stderr,
01750                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01751                simulation, value);
01752 #endif
01753     }
01754 #if DEBUG
01755   fprintf (stderr, "calibrate_best_gradient: end\n");
01756 #endif
01757 }
```

5.1.2.3 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 1873 of file calibrator.c.

```
01875 {
01876   double x;
01877 #if DEBUG
01878   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01879 #endif
01880   x = calibrate->gradient[variable];
01881   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01882     {
01883       if (estimate & 1)
01884         x += calibrate->step[variable];
01885       else
01886         x -= calibrate->step[variable];
01887     }
01888 #if DEBUG
01889   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01890            variable, x);
01891   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01892 #endif
01893   return x;
01894 }
```

5.1.2.4 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 1846 of file calibrator.c.

```
01848 {
01849   double x;
01850 #if DEBUG
01851   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01852 #endif
01853   x = calibrate->gradient[variable]
01854     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->
      step[variable];
01855 #if DEBUG
01856   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01857            variable, x);
01858   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01859 #endif
01860   return x;
01861 }
```

5.1.2.5 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

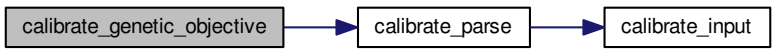
Returns

objective function value.

Definition at line 2028 of file calibrator.c.

```
02029 {
02030   unsigned int j;
02031   double objective;
02032   char buffer[64];
02033 #if DEBUG
02034   fprintf (stderr, "calibrate_genetic_objective: start\n");
02035 #endif
02036   for (j = 0; j < calibrate->nvariables; ++j)
02037     {
02038       calibrate->value[entity->id * calibrate->nvariables + j]
02039         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02040     }
02041   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02042     objective += calibrate_parse (entity->id, j);
02043   g_mutex_lock (mutex);
02044   for (j = 0; j < calibrate->nvariables; ++j)
02045     {
02046       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02047       fprintf (calibrate->file_variables, buffer,
02048                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
02049     }
02050   fprintf (calibrate->file_variables, "%.14le\n", objective);
02051   g_mutex_unlock (mutex);
02052 #if DEBUG
02053   fprintf (stderr, "calibrate_genetic_objective: end\n");
02054 #endif
02055   return objective;
02056 }
```

Here is the call graph for this function:



5.1.2.6 void calibrate_gradient_sequential (unsigned int *simulation*)

Function to estimate the gradient sequentially.

Parameters

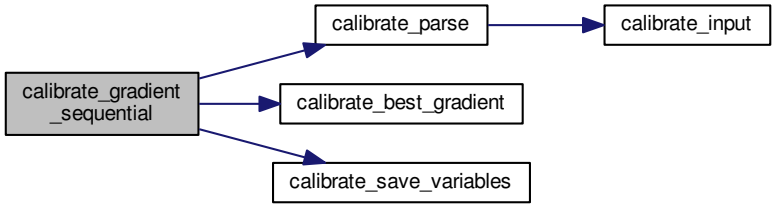
simulation	Simulation number.

Definition at line 1766 of file calibrator.c.

```
01767 {
01768   unsigned int i, j, k;
01769   double e;
01770 #if DEBUG
01771   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01772   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01773            "nend_gradient=%u\n",
01774            calibrate->nstart_gradient, calibrate->
      nend_gradient);
01775 #endif
01776   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01777     {
```

```
01778       k = simulation + i;
01779       e = 0.;
01780       for (j = 0; j < calibrate->nexperiments; ++j)
01781         e += calibrate_parse (k, j);
01782       calibrate_best_gradient (k, e);
01783       calibrate_save_variables (k, e);
01784 #if DEBUG
01785       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01786 #endif
01787     }
01788 #if DEBUG
01789   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01790 #endif
01791 }
```

Here is the call graph for this function:



5.1.2.7 void * calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

Returns

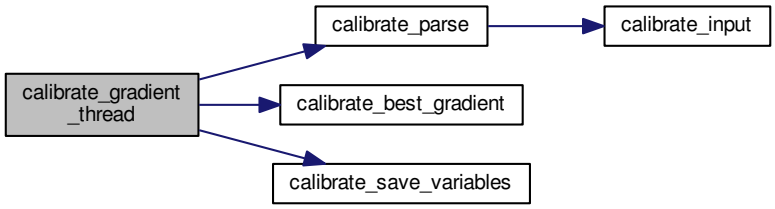
NULL

Definition at line 1801 of file calibrator.c.

```
01802 {
01803   unsigned int i, j, thread;
01804   double e;
01805 #if DEBUG
01806   fprintf (stderr, "calibrate_gradient_thread: start\n");
01807 #endif
01808   thread = data->thread;
01809 #if DEBUG
01810   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01811            thread,
01812            calibrate->thread_gradient[thread],
01813            calibrate->thread_gradient[thread + 1]);
01814 #endif
01815   for (i = calibrate->thread_gradient[thread];
01816        i < calibrate->thread_gradient[thread + 1]; ++i)
01817     {
01818       e = 0.;
01819       for (j = 0; j < calibrate->nexperiments; ++j)
01820         e += calibrate_parse (i, j);
01821       g_mutex_lock (mutex);
01822       calibrate_best_gradient (i, e);
01823       calibrate_save_variables (i, e);
01824       g_mutex_unlock (mutex);
01825 #if DEBUG
01826       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
```

```
01827 #endif
01828     }
01829 #if DEBUG
01830   fprintf (stderr, "calibrate_gradient_thread: end\n");
01831 #endif
01832   g_thread_exit (NULL);
01833   return NULL;
01834 }
```

Here is the call graph for this function:



5.1.2.8 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 1176 of file calibrator.c.

```
01177 {
01178   unsigned int i;
01179   char buffer[32], value[32], *buffer2, *buffer3, *content;
01180   FILE *file;
01181   gsize length;
01182   GRegex *regex;
01183
01184 #if DEBUG
01185   fprintf (stderr, "calibrate_input: start\n");
01186 #endif
01187
01188   // Checking the file
01189   if (!template)
01190     goto calibrate_input_end;
01191
01192   // Opening template
01193   content = g_mapped_file_get_contents (template);
01194   length = g_mapped_file_get_length (template);
01195 #if DEBUG
01196   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01197            content);
01198 #endif
01199   file = g_fopen (input, "w");
01200
01201   // Parsing template
01202   for (i = 0; i < calibrate->nvariables; ++i)
01203     {
01204 #if DEBUG
01205       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01206 #endif
01207       snprintf (buffer, 32, "@variable%u@", i + 1);
01208       regex = g_regex_new (buffer, 0, 0, NULL);
01209       if (i == 0)
01210         {
01211           buffer2 = g_regex_replace_literal (regex, content, length, 0,
```

```
01212                                           calibrate->label[i], 0, NULL);
01213 #if DEBUG
01214           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01215 #endif
01216         }
01217       else
01218         {
01219           length = strlen (buffer3);
01220           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01221                                              calibrate->label[i], 0, NULL);
01222           g_free (buffer3);
01223         }
01224       g_regex_unref (regex);
01225       length = strlen (buffer2);
01226       snprintf (buffer, 32, "@value%u@", i + 1);
01227       regex = g_regex_new (buffer, 0, 0, NULL);
01228       snprintf (value, 32, format[calibrate->precision[i]],
01229                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01230
01231 #if DEBUG
01232       fprintf (stderr, "calibrate_input: value=%s\n", value);
01233 #endif
01234       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01235                                          0, NULL);
01236       g_free (buffer2);
01237       g_regex_unref (regex);
01238     }
01239
01240   // Saving input file
01241   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01242   g_free (buffer3);
01243   fclose (file);
01244
01245 calibrate_input_end:
01246 #if DEBUG
01247   fprintf (stderr, "calibrate_input: end\n");
01248 #endif
01249   return;
01250 }
```

5.1.2.9 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 1541 of file calibrator.c.

```
01543 {
01544   unsigned int i, j, k, s[calibrate->nbest];
01545   double e[calibrate->nbest];
01546 #if DEBUG
01547   fprintf (stderr, "calibrate_merge: start\n");
01548 #endif
01549   i = j = k = 0;
01550   do
01551     {
01552       if (i == calibrate->nsaveds)
01553         {
01554           s[k] = simulation_best[j];
01555           e[k] = error_best[j];
01556           ++j;
01557           ++k;
01558           if (j == nsaveds)
01559             break;
01560         }
01561       else if (j == nsaveds)
01562         {
01563           s[k] = calibrate->simulation_best[i];
01564           e[k] = calibrate->error_best[i];
01565           ++i;
01566           ++k;
01567           if (i == calibrate->nsaveds)
01568             break;
01569         }
```

```
01570       else if (calibrate->error_best[i] > error_best[j])
01571         {
01572           s[k] = simulation_best[j];
01573           e[k] = error_best[j];
01574           ++j;
01575           ++k;
01576         }
01577       else
01578         {
01579           s[k] = calibrate->simulation_best[i];
01580           e[k] = calibrate->error_best[i];
01581           ++i;
01582           ++k;
01583         }
01584     }
01585   while (k < calibrate->nbest);
01586   calibrate->nsaveds = k;
01587   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01588   memcpy (calibrate->error_best, e, k * sizeof (double));
01589 #if DEBUG
01590   fprintf (stderr, "calibrate_merge: end\n");
01591 #endif
01592 }
```

5.1.2.10 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 1263 of file calibrator.c.

```
01264 {
01265   unsigned int i;
01266   double e;
01267   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01268     *buffer3, *buffer4;
01269   FILE *file_result;
01270
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_parse: start\n");
01273   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01274            experiment);
01275 #endif
01276
01277   // Opening input files
01278   for (i = 0; i < calibrate->ninputs; ++i)
01279     {
01280       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01281 #if DEBUG
01282       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01283 #endif
01284       calibrate_input (simulation, &input[i][0],
01285                        calibrate->file[i][experiment]);
01286     }
01287   for (; i < MAX_NINPUTS; ++i)
01288     strcpy (&input[i][0], "");
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_parse: parsing end\n");
01291 #endif
01292
01293   // Performing the simulation
01294   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01295   buffer2 = g_path_get_dirname (calibrate->simulator);
01296   buffer3 = g_path_get_basename (calibrate->simulator);
01297   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01298   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01299             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01300             input[6], input[7], output);
01301   g_free (buffer4);
01302   g_free (buffer3);
```

```
01303   g_free (buffer2);
01304 #if DEBUG
01305   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01306 #endif
01307   system (buffer);
01308
01309   // Checking the objective value function
01310   if (calibrate->evaluator)
01311     {
01312       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01313       buffer2 = g_path_get_dirname (calibrate->evaluator);
01314       buffer3 = g_path_get_basename (calibrate->evaluator);
01315       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01316       snprintf (buffer, 512, "\"%s\" %s %s %s",
01317                 buffer4, output, calibrate->experiment[experiment], result);
01318       g_free (buffer4);
01319       g_free (buffer3);
01320       g_free (buffer2);
01321 #if DEBUG
01322       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01323 #endif
01324       system (buffer);
01325       file_result = g_fopen (result, "r");
01326       e = atof (fgets (buffer, 512, file_result));
01327       fclose (file_result);
01328     }
01329   else
01330     {
01331       strcpy (result, "");
01332       file_result = g_fopen (output, "r");
01333       e = atof (fgets (buffer, 512, file_result));
01334       fclose (file_result);
01335     }
01336
01337   // Removing files
01338 #if !DEBUG
01339   for (i = 0; i < calibrate->ninputs; ++i)
01340     {
01341       if (calibrate->file[i][0])
01342         {
01343           snprintf (buffer, 512, RM " %s", &input[i][0]);
01344           system (buffer);
01345         }
01346     }
01347   snprintf (buffer, 512, RM " %s %s", output, result);
01348   system (buffer);
01349 #endif
01350
01351 #if DEBUG
01352   fprintf (stderr, "calibrate_parse: end\n");
01353 #endif
01354
01355   // Returning the objective function
01356   return e * calibrate->weight[experiment];
01357 }
```

Here is the call graph for this function:

5.1.2.11 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 1395 of file calibrator.c.

```
01396 {
01397   unsigned int i;
01398   char buffer[64];
01399 #if DEBUG
01400   fprintf (stderr, "calibrate_save_variables: start\n");
01401 #endif
01402   for (i = 0; i < calibrate->nvariables; ++i)
01403     {
01404       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01405       fprintf (calibrate->file_variables, buffer,
01406                calibrate->value[simulation * calibrate->
      nvariables + i]);
01407     }
01408   fprintf (calibrate->file_variables, "%.14le\n", error);
01409 #if DEBUG
01410   fprintf (stderr, "calibrate_save_variables: end\n");
01411 #endif
01412 }
```

5.1.2.12 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

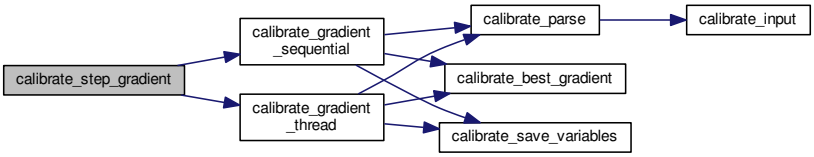
simulation	Simulation number.

Definition at line 1903 of file calibrator.c.

```
01904 {
01905   GThread *thread[nthreads_gradient];
01906   ParallelData data[nthreads_gradient];
01907   unsigned int i, j, k, b;
01908 #if DEBUG
01909   fprintf (stderr, "calibrate_step_gradient: start\n");
01910 #endif
01911   for (i = 0; i < calibrate->nestimates; ++i)
01912     {
01913       k = (simulation + i) * calibrate->nvariables;
01914       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01915 #if DEBUG
01916       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01917                simulation + i, calibrate->simulation_best[0]);
01918 #endif
01919       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01920         {
01921 #if DEBUG
01922           fprintf (stderr,
01923                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01924                    i, j, calibrate->value[b]);
01925 #endif
01926           calibrate->value[k]
01927             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01928           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01929                                             calibrate->rangeminabs[j]),
01930                                       calibrate->rangemaxabs[j]);
01931 #if DEBUG
01932           fprintf (stderr,
01933                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01934                    i, j, calibrate->value[k]);
01935 #endif
01936         }
01937     }
01938   if (nthreads_gradient == 1)
01939     calibrate_gradient_sequential (simulation);
01940   else
01941     {
01942       for (i = 0; i <= nthreads_gradient; ++i)
01943         {
```

```
01944           calibrate->thread_gradient[i]
01945             = simulation + calibrate->nstart_gradient
01946             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01947               / nthreads_gradient;
01948 #if DEBUG
01949           fprintf (stderr,
01950                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01951                    i, calibrate->thread_gradient[i]);
01952 #endif
01953         }
01954       for (i = 0; i < nthreads_gradient; ++i)
01955         {
01956           data[i].thread = i;
01957           thread[i] = g_thread_new
01958             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01959         }
01960       for (i = 0; i < nthreads_gradient; ++i)
01961         g_thread_join (thread[i]);
01962     }
01963 #if DEBUG
01964   fprintf (stderr, "calibrate_step_gradient: end\n");
01965 #endif
01966 }
```

Here is the call graph for this function:



5.1.2.13 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

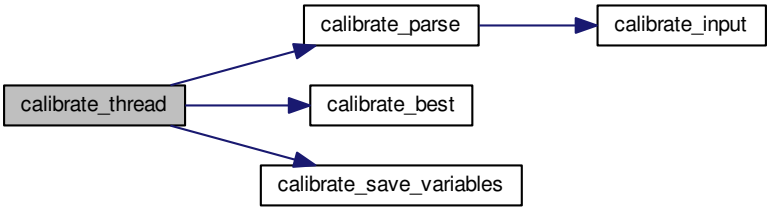
NULL

Definition at line 1497 of file calibrator.c.

```
01498 {
01499   unsigned int i, j, thread;
01500   double e;
01501 #if DEBUG
01502   fprintf (stderr, "calibrate_thread: start\n");
01503 #endif
01504   thread = data->thread;
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01507            calibrate->thread[thread], calibrate->thread[thread + 1]);
01508 #endif
01509   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01510     {
01511       e = 0.;
01512       for (j = 0; j < calibrate->nexperiments; ++j)
01513         e += calibrate_parse (i, j);
01514       g_mutex_lock (mutex);
01515       calibrate_best (i, e);
01516       calibrate_save_variables (i, e);
01517       g_mutex_unlock (mutex);
```

```
01518 #if DEBUG
01519       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01520 #endif
01521     }
01522 #if DEBUG
01523   fprintf (stderr, "calibrate_thread: end\n");
01524 #endif
01525   g_thread_exit (NULL);
01526   return NULL;
01527 }
```

Here is the call graph for this function:



5.1.2.14 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4699 of file calibrator.c.

```
04700 {
04701 #ifdef G_OS_WIN32
04702   SYSTEM_INFO sysinfo;
04703   GetSystemInfo (&sysinfo);
04704   return sysinfo.dwNumberOfProcessors;
04705 #else
04706   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04707 #endif
04708 }
```

5.1.2.15 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 488 of file calibrator.c.

```
00489 {
00490   char buffer2[64];
00491   char *buffert[MAX_NINPUTS] =
00492     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493   xmlDoc *doc;
00494   xmlNode *node, *child;
00495   xmlChar *buffer;
00496   char *msg;
00497   int error_code;
00498   unsigned int i;
00499
00500 #if DEBUG
00501   fprintf (stderr, "input_open: start\n");
00502 #endif
00503
00504   // Resetting input data
00505   buffer = NULL;
00506   input_new ();
00507
00508   // Parsing the input file
00509 #if DEBUG
00510   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00511 #endif
00512   doc = xmlParseFile (filename);
00513   if (!doc)
00514     {
00515       msg = gettext ("Unable to parse the input file");
00516       goto exit_on_error;
00517     }
00518
00519   // Getting the root node
00520 #if DEBUG
00521   fprintf (stderr, "input_open: getting the root node\n");
00522 #endif
00523   node = xmlDocGetRootElement (doc);
00524   if (xmlStrcmp (node->name, XML_CALIBRATE))
00525     {
00526       msg = gettext ("Bad root XML node");
00527       goto exit_on_error;
00528     }
00529
00530   // Getting results file names
00531   input->result = (char *) xmlGetProp (node, XML_RESULT);
00532   if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535   if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538   // Opening simulator program name
00539   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540   if (!input->simulator)
00541     {
00542       msg = gettext ("Bad simulator program");
00543       goto exit_on_error;
00544     }
00545
00546   // Opening evaluator program name
00547   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549   // Obtaining pseudo-random numbers generator seed
00550   if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552   else
00553     {
00554       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555       if (error_code)
00556         {
00557           msg = gettext ("Bad pseudo-random numbers generator seed");
00558           goto exit_on_error;
00559         }
00560     }
00561
00562   // Opening algorithm
00563   buffer = xmlGetProp (node, XML_ALGORITHM);
00564   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565     {
00566       input->algorithm = ALGORITHM_MONTE_CARLO;
00567
00568       // Obtaining simulations number
00569       input->nsimulations
00570         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571       if (error_code)
00572         {
00573           msg = gettext ("Bad simulations number");
00574           goto exit_on_error;
00575         }
```

```
00576     }
00577   else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     {
00579       input->algorithm = ALGORITHM_SWEEP;
00580     }
00581   else if (!xmlStrcmp (buffer, XML_GENETIC))
00582     {
00583       input->algorithm = ALGORITHM_GENETIC;
00584 
00585       // Obtaining population
00586       if (xmlHasProp (node, XML_NPOPULATION))
00587         {
00588           input->nsimulations
00589             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00590           if (error_code || input->nsimulations < 3)
00591             {
00592               msg = gettext ("Invalid population number");
00593               goto exit_on_error;
00594             }
00595         }
00596       else
00597         {
00598           msg = gettext ("No population number");
00599           goto exit_on_error;
00600         }
00601 
00602       // Obtaining generations
00603       if (xmlHasProp (node, XML_NGENERATIONS))
00604         {
00605           input->niterations
00606             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00607           if (error_code || !input->niterations)
00608             {
00609               msg = gettext ("Invalid generations number");
00610               goto exit_on_error;
00611             }
00612         }
00613       else
00614         {
00615           msg = gettext ("No generations number");
00616           goto exit_on_error;
00617         }
00618 
00619       // Obtaining mutation probability
00620       if (xmlHasProp (node, XML_MUTATION))
00621         {
00622           input->mutation_ratio
00623             = xml_node_get_float (node, XML_MUTATION, &error_code);
00624           if (error_code || input->mutation_ratio < 0.
00625               || input->mutation_ratio >= 1.)
00626             {
00627               msg = gettext ("Invalid mutation probability");
00628               goto exit_on_error;
00629             }
00630         }
00631       else
00632         {
00633           msg = gettext ("No mutation probability");
00634           goto exit_on_error;
00635         }
00636 
00637       // Obtaining reproduction probability
00638       if (xmlHasProp (node, XML_REPRODUCTION))
00639         {
00640           input->reproduction_ratio
00641             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00642           if (error_code || input->reproduction_ratio < 0.
00643               || input->reproduction_ratio >= 1.0)
00644             {
00645               msg = gettext ("Invalid reproduction probability");
00646               goto exit_on_error;
00647             }
00648         }
00649       else
00650         {
00651           msg = gettext ("No reproduction probability");
00652           goto exit_on_error;
00653         }
00654 
00655       // Obtaining adaptation probability
00656       if (xmlHasProp (node, XML_ADAPTATION))
00657         {
00658           input->adaptation_ratio
00659             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00660           if (error_code || input->adaptation_ratio < 0.
00661               || input->adaptation_ratio >= 1.)
00662             {
```

```
00663               msg = gettext ("Invalid adaptation probability");
00664               goto exit_on_error;
00665             }
00666         }
00667       else
00668         {
00669           msg = gettext ("No adaptation probability");
00670           goto exit_on_error;
00671         }
00672
00673       // Checking survivals
00674       i = input->mutation_ratio * input->nsimulations;
00675       i += input->reproduction_ratio * input->
      nsimulations;
00676       i += input->adaptation_ratio * input->
      nsimulations;
00677       if (i > input->nsimulations - 2)
00678         {
00679           msg = gettext
00680             ("No enough survival entities to reproduce the population");
00681           goto exit_on_error;
00682         }
00683     }
00684   else
00685     {
00686       msg = gettext ("Unknown algorithm");
00687       goto exit_on_error;
00688     }
00689   xmlFree (buffer);
00690   buffer = NULL;
00691
00692   if (input->algorithm == ALGORITHM_MONTE_CARLO
00693       || input->algorithm == ALGORITHM_SWEEP)
00694     {
00695
00696       // Obtaining iterations number
00697       input->niterations
00698         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00699       if (error_code == 1)
00700         input->niterations = 1;
00701       else if (error_code)
00702         {
00703           msg = gettext ("Bad iterations number");
00704           goto exit_on_error;
00705         }
00706
00707       // Obtaining best number
00708       if (xmlHasProp (node, XML_NBEST))
00709         {
00710           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00711           if (error_code || !input->nbest)
00712             {
00713               msg = gettext ("Invalid best number");
00714               goto exit_on_error;
00715             }
00716         }
00717       else
00718         input->nbest = 1;
00719
00720       // Obtaining tolerance
00721       if (xmlHasProp (node, XML_TOLERANCE))
00722         {
00723           input->tolerance
00724             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00725           if (error_code || input->tolerance < 0.)
00726             {
00727               msg = gettext ("Invalid tolerance");
00728               goto exit_on_error;
00729             }
00730         }
00731       else
00732         input->tolerance = 0.;
00733
00734       // Getting gradient method parameters
00735       if (xmlHasProp (node, XML_NSTEPS))
00736         {
00737           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
00738           if (error_code || !input->nsteps)
00739             {
00740               msg = gettext ("Invalid steps number");
00741               goto exit_on_error;
00742             }
00743           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00744           if (!xmlStrcmp (buffer, XML_COORDINATES))
00745             input->gradient_method =
```

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

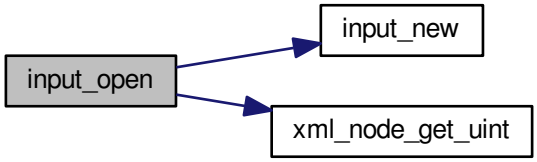
```
00831             = (char **) g_realloc (input->template[0],
00832                                    (1 + input->nexperiments) * sizeof (char *));
00833           buffert[0] = (char *) xmlGetProp (child, template[0]);
00834 #if DEBUG
00835           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00836                    input->nexperiments,
00837                    buffert[0]);
00838 #endif
00839           if (!input->nexperiments)
00840             ++input->ninputs;
00841 #if DEBUG
00842           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00843 #endif
00844         }
00845       else
00846         {
00847           snprintf (buffer2, 64, "%s %s: %s",
00848                     gettext ("Experiment"), buffer, gettext ("no template"));
00849           msg = buffer2;
00850           goto exit_on_error;
00851         }
00852       for (i = 1; i < MAX_NINPUTS; ++i)
00853         {
00854 #if DEBUG
00855           fprintf (stderr, "input_open: template%u\n", i + 1);
00856 #endif
00857           if (xmlHasProp (child, template[i]))
00858             {
00859               if (input->nexperiments && input->ninputs <= i)
00860                 {
00861                   snprintf (buffer2, 64, "%s %s: %s",
00862                             gettext ("Experiment"),
00863                             buffer, gettext ("bad templates number"));
00864                   msg = buffer2;
00865                   while (i-- > 0)
00866                     xmlFree (buffert[i]);
00867                   goto exit_on_error;
00868                 }
00869               input->template[i] = (char **)
00870                 g_realloc (input->template[i],
00871                            (1 + input->nexperiments) * sizeof (char *));
00872               buffert[i] = (char *) xmlGetProp (child, template[i]);
00873 #if DEBUG
00874               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00875                        input->nexperiments, i + 1,
00876                        input->template[i][input->nexperiments]);
00877 #endif
00878               if (!input->nexperiments)
00879                 ++input->ninputs;
00880 #if DEBUG
00881               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00882 #endif
00883             }
00884           else if (input->nexperiments && input->ninputs >= i)
00885             {
00886               snprintf (buffer2, 64, "%s %s: %s%u",
00887                         gettext ("Experiment"),
00888                         buffer, gettext ("no template"), i + 1);
00889               msg = buffer2;
00890               while (i-- > 0)
00891                 xmlFree (buffert[i]);
00892               goto exit_on_error;
00893             }
00894           else
00895             break;
00896         }
00897       input->experiment
00898         = g_realloc (input->experiment,
00899                      (1 + input->nexperiments) * sizeof (char *));
00900       input->experiment[input->nexperiments] = (char *) buffer;
00901       for (i = 0; i < input->ninputs; ++i)
00902         input->template[i][input->nexperiments] = buffert[i];
00903       ++input->nexperiments;
00904 #if DEBUG
00905       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00906 #endif
00907     }
00908   if (!input->nexperiments)
00909     {
00910       msg = gettext ("No calibration experiments");
00911       goto exit_on_error;
00912     }
00913   buffer = NULL;
00914
00915   // Reading the variables data
00916   for (; child; child = child->next)
00917     {
```

```
00918       if (xmlStrcmp (child->name, XML_VARIABLE))
00919         {
00920           snprintf (buffer2, 64, "%s %u: %s",
00921                     gettext ("Variable"),
00922                     input->nvariables + 1, gettext ("bad XML node"));
00923           msg = buffer2;
00924           goto exit_on_error;
00925         }
00926       if (xmlHasProp (child, XML_NAME))
00927         buffer = xmlGetProp (child, XML_NAME);
00928       else
00929         {
00930           snprintf (buffer2, 64, "%s %u: %s",
00931                     gettext ("Variable"),
00932                     input->nvariables + 1, gettext ("no name"));
00933           msg = buffer2;
00934           goto exit_on_error;
00935         }
00936       if (xmlHasProp (child, XML_MINIMUM))
00937         {
00938           input->rangemin = g_realloc
00939             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00940           input->rangeminabs = g_realloc
00941             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00942           input->rangemin[input->nvariables]
00943             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00944           if (error_code)
00945             {
00946               snprintf (buffer2, 64, "%s %s: %s",
00947                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00948               msg = buffer2;
00949               goto exit_on_error;
00950             }
00951           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00952             {
00953               input->rangeminabs[input->nvariables]
00954                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00955               if (error_code)
00956                 {
00957                   snprintf (buffer2, 64, "%s %s: %s",
00958                             gettext ("Variable"),
00959                             buffer, gettext ("bad absolute minimum"));
00960                   msg = buffer2;
00961                   goto exit_on_error;
00962                 }
00963             }
00964           else
00965             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00966           if (input->rangemin[input->nvariables]
00967               < input->rangeminabs[input->nvariables])
00968             {
00969               snprintf (buffer2, 64, "%s %s: %s",
00970                         gettext ("Variable"),
00971                         buffer, gettext ("minimum range not allowed"));
00972               msg = buffer2;
00973               goto exit_on_error;
00974             }
00975         }
00976       else
00977         {
00978           snprintf (buffer2, 64, "%s %s: %s",
00979                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00980           msg = buffer2;
00981           goto exit_on_error;
00982         }
00983       if (xmlHasProp (child, XML_MAXIMUM))
00984         {
00985           input->rangemax = g_realloc
00986             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00987           input->rangemaxabs = g_realloc
00988             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00989           input->rangemax[input->nvariables]
00990             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00991           if (error_code)
00992             {
00993               snprintf (buffer2, 64, "%s %s: %s",
00994                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00995               msg = buffer2;
00996               goto exit_on_error;
00997             }
00998           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00999             {
01000               input->rangemaxabs[input->nvariables]
01001                 = xml_node_get_float (child,
    XML_ABSOLUTE_MAXIMUM, &error_code);
01002               if (error_code)
```

```
01003                 {
01004                   snprintf (buffer2, 64, "%s %s: %s",
01005                             gettext ("Variable"),
01006                             buffer, gettext ("bad absolute maximum"));
01007                   msg = buffer2;
01008                   goto exit_on_error;
01009                 }
01010             }
01011           else
01012             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01013           if (input->rangemax[input->nvariables]
01014               > input->rangemaxabs[input->nvariables])
01015             {
01016               snprintf (buffer2, 64, "%s %s: %s",
01017                         gettext ("Variable"),
01018                         buffer, gettext ("maximum range not allowed"));
01019               msg = buffer2;
01020               goto exit_on_error;
01021             }
01022         }
01023       else
01024         {
01025           snprintf (buffer2, 64, "%s %s: %s",
01026                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01027           msg = buffer2;
01028           goto exit_on_error;
01029         }
01030       if (input->rangemax[input->nvariables]
01031           < input->rangemin[input->nvariables])
01032         {
01033           snprintf (buffer2, 64, "%s %s: %s",
01034                     gettext ("Variable"), buffer, gettext ("bad range"));
01035           msg = buffer2;
01036           goto exit_on_error;
01037         }
01038       input->precision = g_realloc
01039         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01040       if (xmlHasProp (child, XML_PRECISION))
01041         {
01042           input->precision[input->nvariables]
01043             = xml_node_get_uint (child, XML_PRECISION, &error_code);
01044           if (error_code || input->precision[input->
      nvariables] >= NPRECISIONS)
01045             {
01046               snprintf (buffer2, 64, "%s %s: %s",
01047                         gettext ("Variable"),
01048                         buffer, gettext ("bad precision"));
01049               msg = buffer2;
01050               goto exit_on_error;
01051             }
01052         }
01053       else
01054         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
01055       if (input->algorithm == ALGORITHM_SWEEP)
01056         {
01057           if (xmlHasProp (child, XML_NSWEEPS))
01058             {
01059               input->nsweeps = (unsigned int *)
01060                 g_realloc (input->nsweeps,
01061                            (1 + input->nvariables) * sizeof (unsigned int));
01062               input->nsweeps[input->nvariables]
01063                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01064               if (error_code || !input->nsweeps[input->
      nvariables])
01065                 {
01066                   snprintf (buffer2, 64, "%s %s: %s",
01067                             gettext ("Variable"),
01068                             buffer, gettext ("bad sweeps"));
01069                   msg = buffer2;
01070                   goto exit_on_error;
01071                 }
01072             }
01073           else
01074             {
01075               snprintf (buffer2, 64, "%s %s: %s",
01076                         gettext ("Variable"),
01077                         buffer, gettext ("no sweeps number"));
01078               msg = buffer2;
01079               goto exit_on_error;
01080             }
01081 #if DEBUG
01082           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01083                    input->nsweeps[input->nvariables],
      input->nsimulations);
01084 #endif
01085         }
```

```
01086       if (input->algorithm == ALGORITHM_GENETIC)
01087         {
01088           // Obtaining bits representing each variable
01089           if (xmlHasProp (child, XML_NBITS))
01090             {
01091               input->nbits = (unsigned int *)
01092                 g_realloc (input->nbits,
01093                            (1 + input->nvariables) * sizeof (unsigned int));
01094               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01095               if (error_code || !i)
01096                 {
01097                   snprintf (buffer2, 64, "%s %s: %s",
01098                             gettext ("Variable"),
01099                             buffer, gettext ("invalid bits number"));
01100                   msg = buffer2;
01101                   goto exit_on_error;
01102                 }
01103               input->nbits[input->nvariables] = i;
01104             }
01105           else
01106             {
01107               snprintf (buffer2, 64, "%s %s: %s",
01108                         gettext ("Variable"),
01109                         buffer, gettext ("no bits number"));
01110               msg = buffer2;
01111               goto exit_on_error;
01112             }
01113         }
01114       else if (input->nsteps)
01115         {
01116           input->step = (double *)
01117             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01118           input->step[input->nvariables]
01119             = xml_node_get_float (child, XML_STEP, &error_code);
01120           if (error_code || input->step[input->nvariables] < 0.)
01121             {
01122               snprintf (buffer2, 64, "%s %s: %s",
01123                         gettext ("Variable"),
01124                         buffer, gettext ("bad step size"));
01125               msg = buffer2;
01126               goto exit_on_error;
01127             }
01128         }
01129       input->label = g_realloc
01130         (input->label, (1 + input->nvariables) * sizeof (char *));
01131       input->label[input->nvariables] = (char *) buffer;
01132       ++input->nvariables;
01133     }
01134   if (!input->nvariables)
01135     {
01136       msg = gettext ("No calibration variables");
01137       goto exit_on_error;
01138     }
01139   buffer = NULL;
01140
01141   // Getting the working directory
01142   input->directory = g_path_get_dirname (filename);
01143   input->name = g_path_get_basename (filename);
01144
01145   // Closing the XML document
01146   xmlFreeDoc (doc);
01147
01148 #if DEBUG
01149   fprintf (stderr, "input_open: end\n");
01150 #endif
01151   return 1;
01152
01153 exit_on_error:
01154   xmlFree (buffer);
01155   xmlFreeDoc (doc);
01156   show_error (msg);
01157   input_free ();
01158 #if DEBUG
01159   fprintf (stderr, "input_open: end\n");
01160 #endif
01161   return 0;
01162 }
```

Here is the call graph for this function:



5.1.2.16 void input_save (char * *filename*)

Function to save the input file.

Parameters

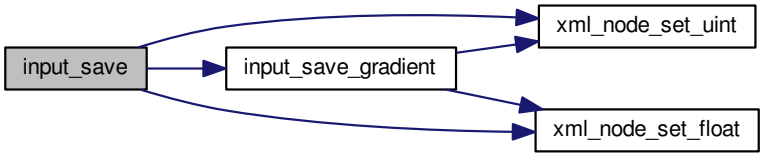
filename	Input file name.

Definition at line 2662 of file calibrator.c.

```
02663 {
02664   unsigned int i, j;
02665   char *buffer;
02666   xmlDoc *doc;
02667   xmlNode *node, *child;
02668   GFile *file, *file2;
02669
02670   // Getting the input file directory
02671   input->name = g_path_get_basename (filename);
02672   input->directory = g_path_get_dirname (filename);
02673   file = g_file_new_for_path (input->directory);
02674
02675   // Opening the input file
02676   doc = xmlNewDoc ((const xmlChar *) "1.0");
02677
02678   // Setting root XML node
02679   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02680   xmlDocSetRootElement (doc, node);
02681
02682   // Adding properties to the root XML node
02683   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02684     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02685   if (xmlStrcmp ((const xmlChar *) input->variables,
     variables_name))
02686     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
     variables);
02687   file2 = g_file_new_for_path (input->simulator);
02688   buffer = g_file_get_relative_path (file, file2);
02689   g_object_unref (file2);
02690   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02691   g_free (buffer);
02692   if (input->evaluator)
02693     {
02694       file2 = g_file_new_for_path (input->evaluator);
02695       buffer = g_file_get_relative_path (file, file2);
02696       g_object_unref (file2);
02697       if (xmlStrlen ((xmlChar *) buffer))
02698         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02699       g_free (buffer);
02700     }
02701   if (input->seed != DEFAULT_RANDOM_SEED)
02702     xml_node_set_uint (node, XML_SEED, input->seed);
02703
02704   // Setting the algorithm
02705   buffer = (char *) g_malloc (64);
02706   switch (input->algorithm)
02707     {
02708     case ALGORITHM_MONTE_CARLO:
```

```
02709       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02710       snprintf (buffer, 64, "%u", input->nsimulations);
02711       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02712       snprintf (buffer, 64, "%u", input->niterations);
02713       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02714       snprintf (buffer, 64, "%.3lg", input->tolerance);
02715       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02716       snprintf (buffer, 64, "%u", input->nbest);
02717       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02718       input_save_gradient (node);
02719       break;
02720     case ALGORITHM_SWEEP:
02721       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02722       snprintf (buffer, 64, "%u", input->niterations);
02723       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02724       snprintf (buffer, 64, "%.3lg", input->tolerance);
02725       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02726       snprintf (buffer, 64, "%u", input->nbest);
02727       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02728       input_save_gradient (node);
02729       break;
02730     default:
02731       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02732       snprintf (buffer, 64, "%u", input->nsimulations);
02733       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02734       snprintf (buffer, 64, "%u", input->niterations);
02735       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02736       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02737       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02738       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02739       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02740       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02741       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02742       break;
02743     }
02744   g_free (buffer);
02745
02746   // Setting the experimental data
02747   for (i = 0; i < input->nexperiments; ++i)
02748     {
02749       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02750       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02751       if (input->weight[i] != 1.)
02752         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02753       for (j = 0; j < input->ninputs; ++j)
02754         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02755     }
02756
02757   // Setting the variables data
02758   for (i = 0; i < input->nvariables; ++i)
02759     {
02760       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02761       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02762       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02763       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02764         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02765       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02766       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02767         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02768       if (input->precision[i] != DEFAULT_PRECISION)
02769         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02770       if (input->algorithm == ALGORITHM_SWEEP)
02771         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02772       else if (input->algorithm == ALGORITHM_GENETIC)
02773         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02774     }
02775
02776   // Saving the XML file
02777   xmlSaveFormatFile (filename, doc, 1);
02778
02779   // Freeing memory
02780   xmlFreeDoc (doc);
02781 }
```

Here is the call graph for this function:



5.1.2.17 void input_save_gradient (xmlNode * *node*)

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

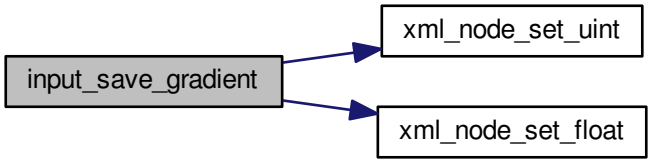
Parameters

node	XML node.

Definition at line 2636 of file calibrator.c.

```
02637 {
02638   if (input->nsteps)
02639     {
02640       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
02641       if (input->relaxation != DEFAULT_RELAXATION)
02642         xml_node_set_float (node, XML_RELAXATION,
      input->relaxation);
02643       switch (input->gradient_method)
02644         {
02645         case GRADIENT_METHOD_COORDINATES:
02646           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02647           break;
02648         default:
02649           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02650           xml_node_set_uint (node, XML_NESTIMATES,
      input->nestimates);
02651         }
02652     }
02653 }
```

Here is the call graph for this function:



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

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

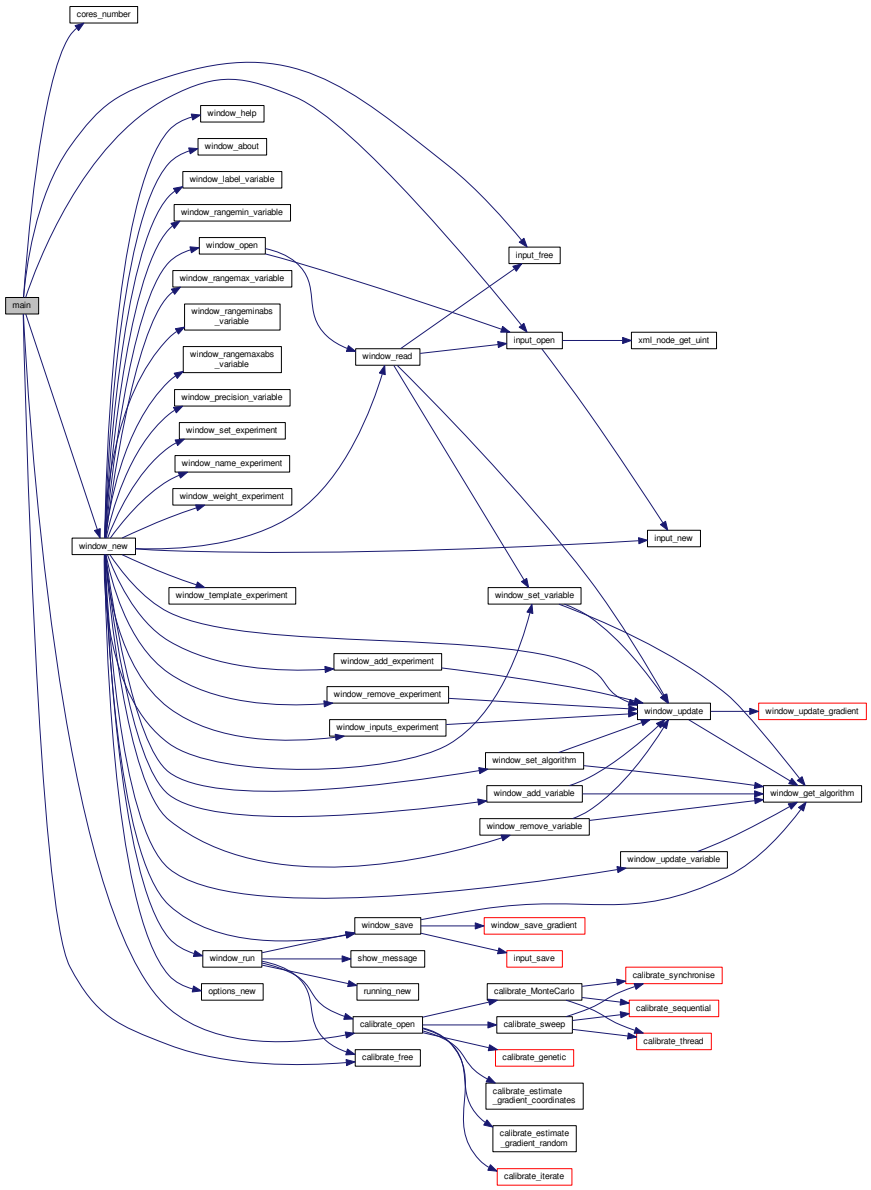
0 on success, >0 on error.

Definition at line 4720 of file calibrator.c.

```
04721 {
04722 #if HAVE_GTK
04723   char *buffer;
04724 #endif
04725
04726   // Starting pseudo-random numbers generator
04727   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04728   calibrate->seed = DEFAULT_RANDOM_SEED;
04729
04730   // Allowing spaces in the XML data file
04731   xmlKeepBlanksDefault (0);
04732
04733   // Starting MPI
04734 #if HAVE_MPI
04735   MPI_Init (&argn, &argc);
04736   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04737   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04738   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04739 #else
04740   ntasks = 1;
04741 #endif
04742
04743 #if HAVE_GTK
04744
04745   // Getting threads number
04746   nthreads_gradient = nthreads = cores_number ();
04747
04748   // Setting local language and international floating point numbers notation
04749   setlocale (LC_ALL, "");
04750   setlocale (LC_NUMERIC, "C");
04751   window->application_directory = g_get_current_dir ();
04752   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04753   bindtextdomain (PROGRAM_INTERFACE, buffer);
04754   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04755   textdomain (PROGRAM_INTERFACE);
04756
04757   // Initing GTK+
04758   gtk_disable_setlocale ();
04759   gtk_init (&argn, &argc);
04760
04761   // Opening the main window
04762   window_new ();
04763   gtk_main ();
04764
04765   // Freeing memory
04766   input_free ();
04767   g_free (buffer);
04768   gtk_widget_destroy (GTK_WIDGET (window->window));
04769   g_free (window->application_directory);
04770
04771 #else
04772
04773   // Checking syntax
04774   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04775     {
04776       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
04777       return 1;
04778     }
04779
04780   // Getting threads number
04781   if (argn == 2)
04782     nthreads_gradient = nthreads = cores_number ();
04783   else
04784     {
04785       nthreads_gradient = nthreads = atoi (argc[2]);
04786       if (!nthreads)
04787         {
04788           printf ("Bad threads number\n");
04789           return 2;
04790         }
```

```
04791     }
04792   printf ("nthreads=%u\n", nthreads);
04793
04794   // Making calibration
04795   if (input_open (argc[argn - 1]))
04796     calibrate_open ();
04797
04798   // Freeing memory
04799   calibrate_free ();
04800
04801 #endif
04802
04803   // Closing MPI
04804 #if HAVE_MPI
04805   MPI_Finalize ();
04806 #endif
04807
04808   // Freeing memory
04809   gsl_rng_free (calibrate->rng);
04810
04811   // Closing
04812   return 0;
04813 }
```

Here is the call graph for this function:



5.1.2.19 void show_error (char * *msg*)

Function to show a dialog with an error message.

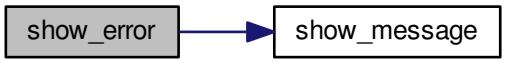
Parameters

msg	Error message.

Definition at line 256 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file calibrator.c.

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

5.1.2.21 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2879 of file calibrator.c.

```
02880 {
02881   unsigned int i;
02882   for (i = 0; i < NALGORITHMS; ++i)
02883     if (gtk_toggle_button_get_active
02884         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02885       break;
02886   return i;
02887 }
```

5.1.2.22 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2895 of file calibrator.c.

```
02896 {
02897   unsigned int i;
02898   for (i = 0; i < NGRADIENTS; ++i)
02899     if (gtk_toggle_button_get_active
02900         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02901       break;
02902   return i;
02903 }
```

5.1.2.23 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.

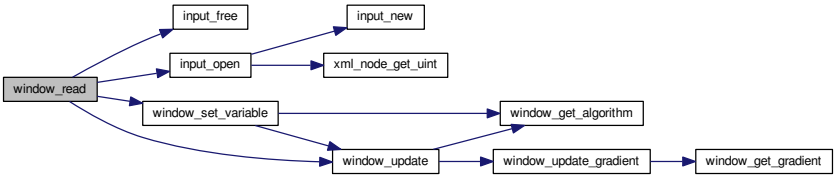
Definition at line 3900 of file calibrator.c.

```
03901 {
03902   unsigned int i;
03903   char *buffer;
03904 #if DEBUG
03905   fprintf (stderr, "window_read: start\n");
03906 #endif
03907
03908   // Reading new input file
03909   input_free ();
03910   if (!input_open (filename))
03911     return 0;
03912
03913   // Setting GTK+ widgets data
03914   gtk_entry_set_text (window->entry_result, input->result);
03915   gtk_entry_set_text (window->entry_variables, input->
      variables);
03916   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03917   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03918                                  (window->button_simulator), buffer);
```

```
03919   g_free (buffer);
03920   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03921                                 (size_t) input->evaluator);
03922   if (input->evaluator)
03923     {
03924       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03925       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03926                                      (window->button_evaluator), buffer);
03927       g_free (buffer);
03928     }
03929   gtk_toggle_button_set_active
03930     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03931   switch (input->algorithm)
03932     {
03933     case ALGORITHM_MONTE_CARLO:
03934       gtk_spin_button_set_value (window->spin_simulations,
03935                                  (gdouble) input->nsimulations);
03936     case ALGORITHM_SWEEP:
03937       gtk_spin_button_set_value (window->spin_iterations,
03938                                  (gdouble) input->niterations);
03939       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03940       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03941       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_gradient),
03942                                     input->nsteps);
03943       if (input->nsteps)
03944         {
03945           gtk_toggle_button_set_active
03946             (GTK_TOGGLE_BUTTON (window->button_gradient
03947                                 [input->gradient_method]), TRUE);
03948           gtk_spin_button_set_value (window->spin_steps,
03949                                      (gdouble) input->nsteps);
03950           gtk_spin_button_set_value (window->spin_relaxation,
03951                                      (gdouble) input->relaxation);
03952           switch (input->gradient_method)
03953             {
03954             case GRADIENT_METHOD_RANDOM:
03955               gtk_spin_button_set_value (window->spin_estimates,
03956                                          (gdouble) input->nestimates);
03957             }
03958         }
03959       break;
03960     default:
03961       gtk_spin_button_set_value (window->spin_population,
03962                                  (gdouble) input->nsimulations);
03963       gtk_spin_button_set_value (window->spin_generations,
03964                                  (gdouble) input->niterations);
03965       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03966       gtk_spin_button_set_value (window->spin_reproduction,
03967                                  input->reproduction_ratio);
03968       gtk_spin_button_set_value (window->spin_adaptation,
03969                                  input->adaptation_ratio);
03970     }
03971   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03972   g_signal_handler_block (window->button_experiment,
03973                           window->id_experiment_name);
03974   gtk_combo_box_text_remove_all (window->combo_experiment);
03975   for (i = 0; i < input->nexperiments; ++i)
03976     gtk_combo_box_text_append_text (window->combo_experiment,
03977                                     input->experiment[i]);
03978   g_signal_handler_unblock
03979     (window->button_experiment, window->
      id_experiment_name);
03980   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03981   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03982   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03983   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03984   gtk_combo_box_text_remove_all (window->combo_variable);
03985   for (i = 0; i < input->nvariables; ++i)
03986     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03987   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03988   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03989   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03990   window_set_variable ();
03991   window_update ();
```

```
03992
03993 #if DEBUG
03994   fprintf (stderr, "window_read: end\n");
03995 #endif
03996   return 1;
03997 }
```

Here is the call graph for this function:



5.1.2.24 int window_save ( )

Function to save the input file.

Returns

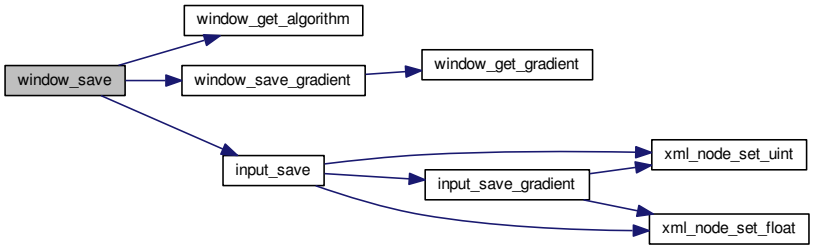
1 on OK, 0 on Cancel.

Definition at line 2943 of file calibrator.c.

```
02944 {
02945   char *buffer;
02946   GtkFileChooserDialog *dlg;
02947
02948 #if DEBUG
02949   fprintf (stderr, "window_save: start\n");
02950 #endif
02951
02952   // Opening the saving dialog
02953   dlg = (GtkFileChooserDialog *)
02954     gtk_file_chooser_dialog_new (gettext ("Save file"),
02955                                  window->window,
02956                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02957                                  gettext ("_Cancel"),
02958                                  GTK_RESPONSE_CANCEL,
02959                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02960   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02961   buffer = g_build_filename (input->directory, input->name, NULL);
02962   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02963   g_free (buffer);
02964
02965   // If OK response then saving
02966   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02967     {
02968
02969       // Adding properties to the root XML node
02970       input->simulator = gtk_file_chooser_get_filename
02971         (GTK_FILE_CHOOSER (window->button_simulator));
02972       if (gtk_toggle_button_get_active
02973           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02974         input->evaluator = gtk_file_chooser_get_filename
02975           (GTK_FILE_CHOOSER (window->button_evaluator));
02976       else
02977         input->evaluator = NULL;
02978       input->result
02979         = (char *) xmlStrdup ((const xmlChar *)
02980                               gtk_entry_get_text (window->entry_result));
02981       input->variables
02982         = (char *) xmlStrdup ((const xmlChar *)
02983                               gtk_entry_get_text (window->entry_variables));
02984
02985       // Setting the algorithm
02986       switch (window_get_algorithm ())
```

```
02987         {
02988         case ALGORITHM_MONTE_CARLO:
02989           input->algorithm = ALGORITHM_MONTE_CARLO;
02990           input->nsimulations
02991             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02992           input->niterations
02993             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02994           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02995           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02996           window_save_gradient ();
02997           break;
02998         case ALGORITHM_SWEEP:
02999           input->algorithm = ALGORITHM_SWEEP;
03000           input->niterations
03001             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03003           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03004           window_save_gradient ();
03005           break;
03006         default:
03007           input->algorithm = ALGORITHM_GENETIC;
03008           input->nsimulations
03009             = gtk_spin_button_get_value_as_int (window->spin_population);
03010           input->niterations
03011             = gtk_spin_button_get_value_as_int (window->spin_generations);
03012           input->mutation_ratio
03013             = gtk_spin_button_get_value (window->spin_mutation);
03014           input->reproduction_ratio
03015             = gtk_spin_button_get_value (window->spin_reproduction);
03016           input->adaptation_ratio
03017             = gtk_spin_button_get_value (window->spin_adaptation);
03018           break;
03019         }
03020
03021       // Saving the XML file
03022       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03023       input_save (buffer);
03024
03025       // Closing and freeing memory
03026       g_free (buffer);
03027       gtk_widget_destroy (GTK_WIDGET (dlg));
03028 #if DEBUG
03029       fprintf (stderr, "window_save: end\n");
03030 #endif
03031       return 1;
03032     }
03033
03034   // Closing and freeing memory
03035   gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037   fprintf (stderr, "window_save: end\n");
03038 #endif
03039   return 0;
03040 }
```

Here is the call graph for this function:



5.1.2.25 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 3536 of file calibrator.c.

```
03537 {
03538   unsigned int i, j;
03539   char *buffer;
03540   GFile *file1, *file2;
03541 #if DEBUG
03542   fprintf (stderr, "window_template_experiment: start\n");
03543 #endif
03544   i = (size_t) data;
03545   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03546   file1
03547     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03548   file2 = g_file_new_for_path (input->directory);
03549   buffer = g_file_get_relative_path (file2, file1);
03550   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03551   g_free (buffer);
03552   g_object_unref (file2);
03553   g_object_unref (file1);
03554 #if DEBUG
03555   fprintf (stderr, "window_template_experiment: end\n");
03556 #endif
03557 }
```

5.1.2.26 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 336 of file calibrator.c.

```
00337 {
00338   double x = 0.;
00339   xmlChar *buffer;
00340   buffer = xmlGetProp (node, prop);
00341   if (!buffer)
00342     *error_code = 1;
00343   else
00344     {
00345       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346         *error_code = 2;
00347       else
00348         *error_code = 0;
00349       xmlFree (buffer);
00350     }
00351   return x;
00352 }
```

5.1.2.27 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 274 of file calibrator.c.

```
00275 {
00276   int i = 0;
00277   xmlChar *buffer;
00278   buffer = xmlGetProp (node, prop);
00279   if (!buffer)
00280     *error_code = 1;
00281   else
00282     {
00283       if (sscanf ((char *) buffer, "%d", &i) != 1)
00284         *error_code = 2;
00285       else
00286         *error_code = 0;
00287       xmlFree (buffer);
00288     }
00289   return i;
00290 }
```

5.1.2.28 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 305 of file calibrator.c.

```
00306 {
00307   unsigned int i = 0;
00308   xmlChar *buffer;
00309   buffer = xmlGetProp (node, prop);
00310   if (!buffer)
00311     *error_code = 1;
00312   else
00313     {
00314       if (sscanf ((char *) buffer, "%u", &i) != 1)
00315         *error_code = 2;
00316       else
00317         *error_code = 0;
00318       xmlFree (buffer);
00319     }
00320   return i;
00321 }
```

5.1.2.29 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 403 of file calibrator.c.

```
00404 {
00405   xmlChar buffer[64];
00406   snprintf ((char *) buffer, 64, "%.14lg", value);
00407   xmlSetProp (node, prop, buffer);
00408 }
```

5.1.2.30 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 365 of file calibrator.c.

```
00366 {
00367   xmlChar buffer[64];
00368   snprintf ((char *) buffer, 64, "%d", value);
00369   xmlSetProp (node, prop, buffer);
00370 }
```

5.1.2.31 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 384 of file calibrator.c.

```
00385 {
00386   xmlChar buffer[64];
00387   snprintf ((char *) buffer, 64, "%u", value);
00388   xmlSetProp (node, prop, buffer);
00389 }
```

5.1.3 Variable Documentation

5.1.3.1 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 117 of file calibrator.c.

5.1.3.2 const double precision[NPRECISIONS]

Initial value:

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

Array of variable precisions.

Definition at line 122 of file calibrator.c.

5.1.3.3 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 110 of file calibrator.c.

5.2 calibrator.c

```
00001 /*
00002 Calibrator: 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 "calibrator.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
00076 #if HAVE_GTK
00077 #define ERROR_TYPE GTK_MESSAGE_ERROR
00078 #define INFO_TYPE GTK_MESSAGE_INFO
00079 #else
00080 #define ERROR_TYPE 0
00081 #define INFO_TYPE 0
00082 #endif
00083 #ifdef G_OS_WIN32
00084 #define INPUT_FILE "test-ga-win.xml"
00085 #define RM "del"
00086 #else
00087 #define INPUT_FILE "test-ga.xml"
00088 #define RM "rm"
00089 #endif
00090
00091 int ntasks;
00092 unsigned int nthreads;
00093 unsigned int nthreads_gradient;
00095 GMutex mutex[1];
00096 void (*calibrate_algorithm) ();
00098 double (*calibrate_estimate_gradient) (unsigned int variable,
00099                                        unsigned int estimate);
00101 Input input[1];
00103 Calibrate calibrate[1];
00104
00105 const xmlChar *result_name = (xmlChar *) "result";
00107 const xmlChar *variables_name = (xmlChar *) "variables";
00109
00110 const xmlChar *template[MAX_NINPUTS] = {
00111   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00112   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00113 };
00114
00116
00117 const char *format[NPRECISIONS] = {
00118   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00119   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00120 };
00121
00122 const double precision[NPRECISIONS] = {
00123   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00124   1e-13, 1e-14
00125 };
00126
00127 const char *logo[] = {
00128   "32 32 3 1",
00129   "     c None",
00130   ".    c #0000FF",
00131   "+    c #FF0000",
00132   "                                ",
00133   "                                ",
00134   "                                ",
00135   "     .      .      .      .     ",
00136   "     .      .      .      .     ",
00137   "     .      .      .      .     ",
00138   "     .      .      .      .     ",
00139   "     .      .     +++     .     ",
00140   "     .      .    +++++    .     ",
00141   "     .      .    +++++    .     ",
00142   "     .      .    +++++    .     ",
00143   "    +++     .     +++    +++    ",
00144   "   +++++    .      .    +++++   ",
00145   "   +++++    .      .    +++++   ",
00146   "   +++++    .      .    +++++   ",
00147   "    +++     .      .     +++    ",
00148   "     .      .      .      .     ",
00149   "     .     +++     .      .     ",
00150   "     .    +++++    .      .     ",
00151   "     .    +++++    .      .     ",
```

```
00152   "      .    +++++    .       .      ",
00153   "      .     +++     .       .      ",
00154   "      .       .      .       .      ",
00155   "      .       .      .       .      ",
00156   "      .       .      .       .      ",
00157   "      .       .      .       .      ",
00158   "      .       .      .       .      ",
00159   "      .       .      .       .      ",
00160   "      .       .      .       .      ",
00161   "                                     ",
00162   "                                     ",
00163   "                                     "
00164 };
00165
00166 /*
00167 const char * logo[] = {
00168 "32 32 3 1",
00169 "     c #FFFFFFFFFFFF",
00170 ".    c #00000000FFFF",
00171 "X    c #FFFF00000000",
00172 "                                     ",
00173 "                                     ",
00174 "                                     ",
00175 "      .      .      .      .        ",
00176 "      .      .      .      .        ",
00177 "      .      .      .      .        ",
00178 "      .      .      .      .        ",
00179 "      .      .     XXX     .        ",
00180 "      .      .    XXXXX    .        ",
00181 "      .      .    XXXXX    .        ",
00182 "      .      .    XXXXX    .        ",
00183 "    XXX      .     XXX    XXX       ",
00184 "   XXXXX     .      .    XXXXX      ",
00185 "   XXXXX     .      .    XXXXX      ",
00186 "   XXXXX     .      .    XXXXX      ",
00187 "    XXX      .      .     XXX       ",
00188 "      .      .      .      .        ",
00189 "      .     XXX     .      .        ",
00190 "      .    XXXXX    .      .        ",
00191 "      .    XXXXX    .      .        ",
00192 "      .    XXXXX    .      .        ",
00193 "      .     XXX     .      .        ",
00194 "      .      .      .      .        ",
00195 "      .      .      .      .        ",
00196 "      .      .      .      .        ",
00197 "      .      .      .      .        ",
00198 "      .      .      .      .        ",
00199 "      .      .      .      .        ",
00200 "      .      .      .      .        ",
00201 "                                     ",
00202 "                                     ",
00203 "                                     "};
00204 */
00205
00206 #if HAVE_GTK
00207 Options options[1];
00209 Running running[1];
00211 Window window[1];
00213 #endif
00214
00225 void
00226 show_message (char *title, char *msg, int type)
00227 {
00228 #if HAVE_GTK
00229   GtkMessageDialog *dlg;
00230
00231   // Creating the dialog
00232   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00233     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00234
00235   // Setting the dialog title
00236   gtk_window_set_title (GTK_WINDOW (dlg), title);
00237
00238   // Showing the dialog and waiting response
00239   gtk_dialog_run (GTK_DIALOG (dlg));
00240
00241   // Closing and freeing memory
00242   gtk_widget_destroy (GTK_WIDGET (dlg));
00243
00244 #else
00245   printf ("%s: %s\n", title, msg);
00246 #endif
00247 }
00248
00255 void
00256 show_error (char *msg)
00257 {
```

```
00258   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00259 }
00260
00273 int
00274 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00275 {
00276   int i = 0;
00277   xmlChar *buffer;
00278   buffer = xmlGetProp (node, prop);
00279   if (!buffer)
00280     *error_code = 1;
00281   else
00282     {
00283       if (sscanf ((char *) buffer, "%d", &i) != 1)
00284         *error_code = 2;
00285       else
00286         *error_code = 0;
00287       xmlFree (buffer);
00288     }
00289   return i;
00290 }
00291
00304 unsigned int
00305 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00306 {
00307   unsigned int i = 0;
00308   xmlChar *buffer;
00309   buffer = xmlGetProp (node, prop);
00310   if (!buffer)
00311     *error_code = 1;
00312   else
00313     {
00314       if (sscanf ((char *) buffer, "%u", &i) != 1)
00315         *error_code = 2;
00316       else
00317         *error_code = 0;
00318       xmlFree (buffer);
00319     }
00320   return i;
00321 }
00322
00335 double
00336 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00337 {
00338   double x = 0.;
00339   xmlChar *buffer;
00340   buffer = xmlGetProp (node, prop);
00341   if (!buffer)
00342     *error_code = 1;
00343   else
00344     {
00345       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346         *error_code = 2;
00347       else
00348         *error_code = 0;
00349       xmlFree (buffer);
00350     }
00351   return x;
00352 }
00353
00364 void
00365 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00366 {
00367   xmlChar buffer[64];
00368   snprintf ((char *) buffer, 64, "%d", value);
00369   xmlSetProp (node, prop, buffer);
00370 }
00371
00383 void
00384 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00385 {
00386   xmlChar buffer[64];
00387   snprintf ((char *) buffer, 64, "%u", value);
00388   xmlSetProp (node, prop, buffer);
00389 }
00390
00402 void
00403 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00404 {
00405   xmlChar buffer[64];
00406   snprintf ((char *) buffer, 64, "%.14lg", value);
00407   xmlSetProp (node, prop, buffer);
00408 }
00409
00414 void
00415 input_new ()
00416 {
```

```
00417   unsigned int i;
00418 #if DEBUG
00419   fprintf (stderr, "input_new: start\n");
00420 #endif
00421   input->nvariables = input->nexperiments = input->ninputs = input->
      nsteps = 0;
00422   input->simulator = input->evaluator = input->directory = input->
      name
00423     = input->result = input->variables = NULL;
00424   input->experiment = input->label = NULL;
00425   input->precision = input->nsweeps = input->nbits = NULL;
00426   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00427     = input->weight = input->step = NULL;
00428   for (i = 0; i < MAX_NINPUTS; ++i)
00429     input->template[i] = NULL;
00430 #if DEBUG
00431   fprintf (stderr, "input_new: end\n");
00432 #endif
00433 }
00434
00439 void
00440 input_free ()
00441 {
00442   unsigned int i, j;
00443 #if DEBUG
00444   fprintf (stderr, "input_free: start\n");
00445 #endif
00446   g_free (input->name);
00447   g_free (input->directory);
00448   for (i = 0; i < input->nexperiments; ++i)
00449     {
00450       xmlFree (input->experiment[i]);
00451       for (j = 0; j < input->ninputs; ++j)
00452         xmlFree (input->template[j][i]);
00453       g_free (input->template[j]);
00454     }
00455   g_free (input->experiment);
00456   for (i = 0; i < input->ninputs; ++i)
00457     g_free (input->template[i]);
00458   for (i = 0; i < input->nvariables; ++i)
00459     xmlFree (input->label[i]);
00460   g_free (input->label);
00461   g_free (input->precision);
00462   g_free (input->rangemin);
00463   g_free (input->rangemax);
00464   g_free (input->rangeminabs);
00465   g_free (input->rangemaxabs);
00466   g_free (input->weight);
00467   g_free (input->step);
00468   g_free (input->nsweeps);
00469   g_free (input->nbits);
00470   xmlFree (input->evaluator);
00471   xmlFree (input->simulator);
00472   xmlFree (input->result);
00473   xmlFree (input->variables);
00474   input->nexperiments = input->ninputs = input->nvariables = input->
      nsteps = 0;
00475 #if DEBUG
00476   fprintf (stderr, "input_free: end\n");
00477 #endif
00478 }
00479
00487 int
00488 input_open (char *filename)
00489 {
00490   char buffer2[64];
00491   char *buffert[MAX_NINPUTS] =
00492     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493   xmlDoc *doc;
00494   xmlNode *node, *child;
00495   xmlChar *buffer;
00496   char *msg;
00497   int error_code;
00498   unsigned int i;
00499
00500 #if DEBUG
00501   fprintf (stderr, "input_open: start\n");
00502 #endif
00503
00504   // Resetting input data
00505   buffer = NULL;
00506   input_new ();
00507
00508   // Parsing the input file
00509 #if DEBUG
00510   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
```

```
00511 #endif
00512   doc = xmlParseFile (filename);
00513   if (!doc)
00514     {
00515       msg = gettext ("Unable to parse the input file");
00516       goto exit_on_error;
00517     }
00518
00519   // Getting the root node
00520 #if DEBUG
00521   fprintf (stderr, "input_open: getting the root node\n");
00522 #endif
00523   node = xmlDocGetRootElement (doc);
00524   if (xmlStrcmp (node->name, XML_CALIBRATE))
00525     {
00526       msg = gettext ("Bad root XML node");
00527       goto exit_on_error;
00528     }
00529
00530   // Getting results file names
00531   input->result = (char *) xmlGetProp (node, XML_RESULT);
00532   if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535   if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538   // Opening simulator program name
00539   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540   if (!input->simulator)
00541     {
00542       msg = gettext ("Bad simulator program");
00543       goto exit_on_error;
00544     }
00545
00546   // Opening evaluator program name
00547   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549   // Obtaining pseudo-random numbers generator seed
00550   if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552   else
00553     {
00554       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555       if (error_code)
00556         {
00557           msg = gettext ("Bad pseudo-random numbers generator seed");
00558           goto exit_on_error;
00559         }
00560     }
00561
00562   // Opening algorithm
00563   buffer = xmlGetProp (node, XML_ALGORITHM);
00564   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565     {
00566       input->algorithm = ALGORITHM_MONTE_CARLO;
00567
00568       // Obtaining simulations number
00569       input->nsimulations
00570         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571       if (error_code)
00572         {
00573           msg = gettext ("Bad simulations number");
00574           goto exit_on_error;
00575         }
00576     }
00577   else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     {
00579       input->algorithm = ALGORITHM_SWEEP;
00580     }
00581   else if (!xmlStrcmp (buffer, XML_GENETIC))
00582     {
00583       input->algorithm = ALGORITHM_GENETIC;
00584
00585       // Obtaining population
00586       if (xmlHasProp (node, XML_NPOPULATION))
00587         {
00588           input->nsimulations
00589             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00590           if (error_code || input->nsimulations < 3)
00591             {
00592               msg = gettext ("Invalid population number");
00593               goto exit_on_error;
00594             }
00595         }
00596       else
00597         {
```

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

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

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

```
00857           if (xmlHasProp (child, template[i]))
00858             {
00859               if (input->nexperiments && input->ninputs <= i)
00860                 {
00861                   snprintf (buffer2, 64, "%s %s: %s",
00862                             gettext ("Experiment"),
00863                             buffer, gettext ("bad templates number"));
00864                   msg = buffer2;
00865                   while (i-- > 0)
00866                     xmlFree (buffert[i]);
00867                   goto exit_on_error;
00868                 }
00869               input->template[i] = (char **)
00870                 g_realloc (input->template[i],
00871                            (1 + input->nexperiments) * sizeof (char *));
00872               buffert[i] = (char *) xmlGetProp (child, template[i]);
00873 #if DEBUG
00874               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00875                        input->nexperiments, i + 1,
00876                        input->template[i][input->nexperiments]);
00877 #endif
00878               if (!input->nexperiments)
00879                 ++input->ninputs;
00880 #if DEBUG
00881               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00882 #endif
00883             }
00884           else if (input->nexperiments && input->ninputs >= i)
00885             {
00886               snprintf (buffer2, 64, "%s %s: %s%u",
00887                         gettext ("Experiment"),
00888                         buffer, gettext ("no template"), i + 1);
00889               msg = buffer2;
00890               while (i-- > 0)
00891                 xmlFree (buffert[i]);
00892               goto exit_on_error;
00893             }
00894           else
00895             break;
00896         }
00897       input->experiment
00898         = g_realloc (input->experiment,
00899                      (1 + input->nexperiments) * sizeof (char *));
00900       input->experiment[input->nexperiments] = (char *) buffer;
00901       for (i = 0; i < input->ninputs; ++i)
00902         input->template[i][input->nexperiments] = buffert[i];
00903       ++input->nexperiments;
00904 #if DEBUG
00905       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00906 #endif
00907     }
00908   if (!input->nexperiments)
00909     {
00910       msg = gettext ("No calibration experiments");
00911       goto exit_on_error;
00912     }
00913   buffer = NULL;
00914
00915   // Reading the variables data
00916   for (; child; child = child->next)
00917     {
00918       if (xmlStrcmp (child->name, XML_VARIABLE))
00919         {
00920           snprintf (buffer2, 64, "%s %u: %s",
00921                     gettext ("Variable"),
00922                     input->nvariables + 1, gettext ("bad XML node"));
00923           msg = buffer2;
00924           goto exit_on_error;
00925         }
00926       if (xmlHasProp (child, XML_NAME))
00927         buffer = xmlGetProp (child, XML_NAME);
00928       else
00929         {
00930           snprintf (buffer2, 64, "%s %u: %s",
00931                     gettext ("Variable"),
00932                     input->nvariables + 1, gettext ("no name"));
00933           msg = buffer2;
00934           goto exit_on_error;
00935         }
00936       if (xmlHasProp (child, XML_MINIMUM))
00937         {
00938           input->rangemin = g_realloc
00939             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00940           input->rangeminabs = g_realloc
00941             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00942           input->rangemin[input->nvariables]
00943             = xml_node_get_float (child, XML_MINIMUM, &error_code);
```

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

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

```
01113         }
01114       else if (input->nsteps)
01115         {
01116           input->step = (double *)
01117             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01118           input->step[input->nvariables]
01119             = xml_node_get_float (child, XML_STEP, &error_code);
01120           if (error_code || input->step[input->nvariables] < 0.)
01121             {
01122               snprintf (buffer2, 64, "%s %s: %s",
01123                         gettext ("Variable"),
01124                         buffer, gettext ("bad step size"));
01125               msg = buffer2;
01126               goto exit_on_error;
01127             }
01128         }
01129       input->label = g_realloc
01130         (input->label, (1 + input->nvariables) * sizeof (char *));
01131       input->label[input->nvariables] = (char *) buffer;
01132       ++input->nvariables;
01133     }
01134   if (!input->nvariables)
01135     {
01136       msg = gettext ("No calibration variables");
01137       goto exit_on_error;
01138     }
01139   buffer = NULL;
01140
01141   // Getting the working directory
01142   input->directory = g_path_get_dirname (filename);
01143   input->name = g_path_get_basename (filename);
01144
01145   // Closing the XML document
01146   xmlFreeDoc (doc);
01147
01148 #if DEBUG
01149   fprintf (stderr, "input_open: end\n");
01150 #endif
01151   return 1;
01152
01153 exit_on_error:
01154   xmlFree (buffer);
01155   xmlFreeDoc (doc);
01156   show_error (msg);
01157   input_free ();
01158 #if DEBUG
01159   fprintf (stderr, "input_open: end\n");
01160 #endif
01161   return 0;
01162 }
01163
01175 void
01176 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01177 {
01178   unsigned int i;
01179   char buffer[32], value[32], *buffer2, *buffer3, *content;
01180   FILE *file;
01181   gsize length;
01182   GRegex *regex;
01183
01184 #if DEBUG
01185   fprintf (stderr, "calibrate_input: start\n");
01186 #endif
01187
01188   // Checking the file
01189   if (!template)
01190     goto calibrate_input_end;
01191
01192   // Opening template
01193   content = g_mapped_file_get_contents (template);
01194   length = g_mapped_file_get_length (template);
01195 #if DEBUG
01196   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01197            content);
01198 #endif
01199   file = g_fopen (input, "w");
01200
01201   // Parsing template
01202   for (i = 0; i < calibrate->nvariables; ++i)
01203     {
01204 #if DEBUG
01205       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01206 #endif
01207       snprintf (buffer, 32, "@variable%u@", i + 1);
01208       regex = g_regex_new (buffer, 0, 0, NULL);
01209       if (i == 0)
01210         {
```

```
01211           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01212                                              calibrate->label[i], 0, NULL);
01213 #if DEBUG
01214           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01215 #endif
01216         }
01217       else
01218         {
01219           length = strlen (buffer3);
01220           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01221                                              calibrate->label[i], 0, NULL);
01222           g_free (buffer3);
01223         }
01224       g_regex_unref (regex);
01225       length = strlen (buffer2);
01226       snprintf (buffer, 32, "@value%u@", i + 1);
01227       regex = g_regex_new (buffer, 0, 0, NULL);
01228       snprintf (value, 32, format[calibrate->precision[i]],
01229                 calibrate->value[simulation * calibrate->nvariables + i]);
01230
01231 #if DEBUG
01232       fprintf (stderr, "calibrate_input: value=%s\n", value);
01233 #endif
01234       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01235                                          0, NULL);
01236       g_free (buffer2);
01237       g_regex_unref (regex);
01238     }
01239
01240   // Saving input file
01241   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01242   g_free (buffer3);
01243   fclose (file);
01244
01245 calibrate_input_end:
01246 #if DEBUG
01247   fprintf (stderr, "calibrate_input: end\n");
01248 #endif
01249   return;
01250 }
01251
01262 double
01263 calibrate_parse (unsigned int simulation, unsigned int experiment)
01264 {
01265   unsigned int i;
01266   double e;
01267   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01268     *buffer3, *buffer4;
01269   FILE *file_result;
01270
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_parse: start\n");
01273   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01274            experiment);
01275 #endif
01276
01277   // Opening input files
01278   for (i = 0; i < calibrate->ninputs; ++i)
01279     {
01280       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01281 #if DEBUG
01282       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01283 #endif
01284       calibrate_input (simulation, &input[i][0],
01285                        calibrate->file[i][experiment]);
01286     }
01287   for (; i < MAX_NINPUTS; ++i)
01288     strcpy (&input[i][0], "");
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_parse: parsing end\n");
01291 #endif
01292
01293   // Performing the simulation
01294   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01295   buffer2 = g_path_get_dirname (calibrate->simulator);
01296   buffer3 = g_path_get_basename (calibrate->simulator);
01297   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01298   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01299             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01300             input[6], input[7], output);
01301   g_free (buffer4);
01302   g_free (buffer3);
01303   g_free (buffer2);
01304 #if DEBUG
01305   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01306 #endif
01307   system (buffer);
```

```
01308
01309   // Checking the objective value function
01310   if (calibrate->evaluator)
01311     {
01312       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01313       buffer2 = g_path_get_dirname (calibrate->evaluator);
01314       buffer3 = g_path_get_basename (calibrate->evaluator);
01315       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01316       snprintf (buffer, 512, "\"%s\" %s %s %s",
01317                 buffer4, output, calibrate->experiment[experiment], result);
01318       g_free (buffer4);
01319       g_free (buffer3);
01320       g_free (buffer2);
01321 #if DEBUG
01322       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01323 #endif
01324       system (buffer);
01325       file_result = g_fopen (result, "r");
01326       e = atof (fgets (buffer, 512, file_result));
01327       fclose (file_result);
01328     }
01329   else
01330     {
01331       strcpy (result, "");
01332       file_result = g_fopen (output, "r");
01333       e = atof (fgets (buffer, 512, file_result));
01334       fclose (file_result);
01335     }
01336
01337   // Removing files
01338 #if !DEBUG
01339   for (i = 0; i < calibrate->ninputs; ++i)
01340     {
01341       if (calibrate->file[i][0])
01342         {
01343           snprintf (buffer, 512, RM " %s", &input[i][0]);
01344           system (buffer);
01345         }
01346     }
01347   snprintf (buffer, 512, RM " %s %s", output, result);
01348   system (buffer);
01349 #endif
01350
01351 #if DEBUG
01352   fprintf (stderr, "calibrate_parse: end\n");
01353 #endif
01354
01355   // Returning the objective function
01356   return e * calibrate->weight[experiment];
01357 }
01358
01363 void
01364 calibrate_print ()
01365 {
01366   unsigned int i;
01367   char buffer[512];
01368 #if HAVE_MPI
01369   if (calibrate->mpi_rank)
01370     return;
01371 #endif
01372   printf ("%s\n", gettext ("Best result"));
01373   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01374   printf ("error = %.15le\n", calibrate->error_old[0]);
01375   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
      error_old[0]);
01376   for (i = 0; i < calibrate->nvariables; ++i)
01377     {
01378       snprintf (buffer, 512, "%s = %s\n",
01379                 calibrate->label[i], format[calibrate->precision[i]]);
01380       printf (buffer, calibrate->value_old[i]);
01381       fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01382     }
01383   fflush (calibrate->file_result);
01384 }
01385
01394 void
01395 calibrate_save_variables (unsigned int simulation, double error)
01396 {
01397   unsigned int i;
01398   char buffer[64];
01399 #if DEBUG
01400   fprintf (stderr, "calibrate_save_variables: start\n");
01401 #endif
01402   for (i = 0; i < calibrate->nvariables; ++i)
01403     {
01404       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01405       fprintf (calibrate->file_variables, buffer,
```

```
01406                 calibrate->value[simulation * calibrate->nvariables + i]);
01407     }
01408   fprintf (calibrate->file_variables, "%.14le\n", error);
01409 #if DEBUG
01410   fprintf (stderr, "calibrate_save_variables: end\n");
01411 #endif
01412 }
01413
01422 void
01423 calibrate_best (unsigned int simulation, double value)
01424 {
01425   unsigned int i, j;
01426   double e;
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_best: start\n");
01429   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01430            calibrate->nsaveds, calibrate->nbest);
01431 #endif
01432   if (calibrate->nsaveds < calibrate->nbest
01433       || value < calibrate->error_best[calibrate->nsaveds - 1])
01434     {
01435       if (calibrate->nsaveds < calibrate->nbest)
01436         ++calibrate->nsaveds;
01437       calibrate->error_best[calibrate->nsaveds - 1] = value;
01438       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01439       for (i = calibrate->nsaveds; --i;)
01440         {
01441           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01442             {
01443               j = calibrate->simulation_best[i];
01444               e = calibrate->error_best[i];
01445               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01446               calibrate->error_best[i] = calibrate->error_best[i - 1];
01447               calibrate->simulation_best[i - 1] = j;
01448               calibrate->error_best[i - 1] = e;
01449             }
01450           else
01451             break;
01452         }
01453     }
01454 #if DEBUG
01455   fprintf (stderr, "calibrate_best: end\n");
01456 #endif
01457 }
01458
01463 void
01464 calibrate_sequential ()
01465 {
01466   unsigned int i, j;
01467   double e;
01468 #if DEBUG
01469   fprintf (stderr, "calibrate_sequential: start\n");
01470   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01471            calibrate->nstart, calibrate->nend);
01472 #endif
01473   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01474     {
01475       e = 0.;
01476       for (j = 0; j < calibrate->nexperiments; ++j)
01477         e += calibrate_parse (i, j);
01478       calibrate_best (i, e);
01479       calibrate_save_variables (i, e);
01480 #if DEBUG
01481       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01482 #endif
01483     }
01484 #if DEBUG
01485   fprintf (stderr, "calibrate_sequential: end\n");
01486 #endif
01487 }
01488
01496 void *
01497 calibrate_thread (ParallelData * data)
01498 {
01499   unsigned int i, j, thread;
01500   double e;
01501 #if DEBUG
01502   fprintf (stderr, "calibrate_thread: start\n");
01503 #endif
01504   thread = data->thread;
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01507            calibrate->thread[thread], calibrate->thread[thread + 1]);
01508 #endif
01509   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01510     {
```

```
01511       e = 0.;
01512       for (j = 0; j < calibrate->nexperiments; ++j)
01513         e += calibrate_parse (i, j);
01514       g_mutex_lock (mutex);
01515       calibrate_best (i, e);
01516       calibrate_save_variables (i, e);
01517       g_mutex_unlock (mutex);
01518 #if DEBUG
01519       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01520 #endif
01521     }
01522 #if DEBUG
01523   fprintf (stderr, "calibrate_thread: end\n");
01524 #endif
01525   g_thread_exit (NULL);
01526   return NULL;
01527 }
01528
01540 void
01541 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01542                  double *error_best)
01543 {
01544   unsigned int i, j, k, s[calibrate->nbest];
01545   double e[calibrate->nbest];
01546 #if DEBUG
01547   fprintf (stderr, "calibrate_merge: start\n");
01548 #endif
01549   i = j = k = 0;
01550   do
01551     {
01552       if (i == calibrate->nsaveds)
01553         {
01554           s[k] = simulation_best[j];
01555           e[k] = error_best[j];
01556           ++j;
01557           ++k;
01558           if (j == nsaveds)
01559             break;
01560         }
01561       else if (j == nsaveds)
01562         {
01563           s[k] = calibrate->simulation_best[i];
01564           e[k] = calibrate->error_best[i];
01565           ++i;
01566           ++k;
01567           if (i == calibrate->nsaveds)
01568             break;
01569         }
01570       else if (calibrate->error_best[i] > error_best[j])
01571         {
01572           s[k] = simulation_best[j];
01573           e[k] = error_best[j];
01574           ++j;
01575           ++k;
01576         }
01577       else
01578         {
01579           s[k] = calibrate->simulation_best[i];
01580           e[k] = calibrate->error_best[i];
01581           ++i;
01582           ++k;
01583         }
01584     }
01585   while (k < calibrate->nbest);
01586   calibrate->nsaveds = k;
01587   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01588   memcpy (calibrate->error_best, e, k * sizeof (double));
01589 #if DEBUG
01590   fprintf (stderr, "calibrate_merge: end\n");
01591 #endif
01592 }
01593
01598 #if HAVE_MPI
01599 void
01600 calibrate_synchronise ()
01601 {
01602   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01603   double error_best[calibrate->nbest];
01604   MPI_Status mpi_stat;
01605 #if DEBUG
01606   fprintf (stderr, "calibrate_synchronise: start\n");
01607 #endif
01608   if (calibrate->mpi_rank == 0)
01609     {
01610       for (i = 1; i < ntasks; ++i)
01611         {
01612           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
```

```
01613           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01614                     MPI_COMM_WORLD, &mpi_stat);
01615           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01616                     MPI_COMM_WORLD, &mpi_stat);
01617           calibrate_merge (nsaveds, simulation_best, error_best);
01618         }
01619     }
01620   else
01621     {
01622       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01623       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01624                 MPI_COMM_WORLD);
01625       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01626                 MPI_COMM_WORLD);
01627     }
01628 #if DEBUG
01629   fprintf (stderr, "calibrate_synchronise: end\n");
01630 #endif
01631 }
01632 #endif
01633
01638 void
01639 calibrate_sweep ()
01640 {
01641   unsigned int i, j, k, l;
01642   double e;
01643   GThread *thread[nthreads];
01644   ParallelData data[nthreads];
01645 #if DEBUG
01646   fprintf (stderr, "calibrate_sweep: start\n");
01647 #endif
01648   for (i = 0; i < calibrate->nsimulations; ++i)
01649     {
01650       k = i;
01651       for (j = 0; j < calibrate->nvariables; ++j)
01652         {
01653           l = k % calibrate->nsweeps[j];
01654           k /= calibrate->nsweeps[j];
01655           e = calibrate->rangemin[j];
01656           if (calibrate->nsweeps[j] > 1)
01657             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01658               / (calibrate->nsweeps[j] - 1);
01659           calibrate->value[i * calibrate->nvariables + j] = e;
01660         }
01661     }
01662   calibrate->nsaveds = 0;
01663   if (nthreads <= 1)
01664     calibrate_sequential ();
01665   else
01666     {
01667       for (i = 0; i < nthreads; ++i)
01668         {
01669           data[i].thread = i;
01670           thread[i]
01671             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01672         }
01673       for (i = 0; i < nthreads; ++i)
01674         g_thread_join (thread[i]);
01675     }
01676 #if HAVE_MPI
01677   // Communicating tasks results
01678   calibrate_synchronise ();
01679 #endif
01680 #if DEBUG
01681   fprintf (stderr, "calibrate_sweep: end\n");
01682 #endif
01683 }
01684
01689 void
01690 calibrate_MonteCarlo ()
01691 {
01692   unsigned int i, j;
01693   GThread *thread[nthreads];
01694   ParallelData data[nthreads];
01695 #if DEBUG
01696   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01697 #endif
01698   for (i = 0; i < calibrate->nsimulations; ++i)
01699     for (j = 0; j < calibrate->nvariables; ++j)
01700       calibrate->value[i * calibrate->nvariables + j]
01701         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01702         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01703   calibrate->nsaveds = 0;
01704   if (nthreads <= 1)
01705     calibrate_sequential ();
01706   else
01707     {
```

```
01708       for (i = 0; i < nthreads; ++i)
01709         {
01710           data[i].thread = i;
01711           thread[i]
01712             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01713         }
01714       for (i = 0; i < nthreads; ++i)
01715         g_thread_join (thread[i]);
01716     }
01717 #if HAVE_MPI
01718   // Communicating tasks results
01719   calibrate_synchronise ();
01720 #endif
01721 #if DEBUG
01722   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01723 #endif
01724 }
01725
01735 void
01736 calibrate_best_gradient (unsigned int simulation, double value)
01737 {
01738 #if DEBUG
01739   fprintf (stderr, "calibrate_best_gradient: start\n");
01740   fprintf (stderr,
01741            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01742            simulation, value, calibrate->error_best[0]);
01743 #endif
01744   if (value < calibrate->error_best[0])
01745     {
01746       calibrate->error_best[0] = value;
01747       calibrate->simulation_best[0] = simulation;
01748 #if DEBUG
01749       fprintf (stderr,
01750                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01751                simulation, value);
01752 #endif
01753     }
01754 #if DEBUG
01755   fprintf (stderr, "calibrate_best_gradient: end\n");
01756 #endif
01757 }
01758
01765 void
01766 calibrate_gradient_sequential (unsigned int simulation)
01767 {
01768   unsigned int i, j, k;
01769   double e;
01770 #if DEBUG
01771   fprintf (stderr, "calibrate_gradient_sequential: start\n");
01772   fprintf (stderr, "calibrate_gradient_sequential: nstart_gradient=%u "
01773            "nend_gradient=%u\n",
01774            calibrate->nstart_gradient, calibrate->nend_gradient);
01775 #endif
01776   for (i = calibrate->nstart_gradient; i < calibrate->nend_gradient; ++i)
01777     {
01778       k = simulation + i;
01779       e = 0.;
01780       for (j = 0; j < calibrate->nexperiments; ++j)
01781         e += calibrate_parse (k, j);
01782       calibrate_best_gradient (k, e);
01783       calibrate_save_variables (k, e);
01784 #if DEBUG
01785       fprintf (stderr, "calibrate_gradient_sequential: i=%u e=%lg\n", i, e);
01786 #endif
01787     }
01788 #if DEBUG
01789   fprintf (stderr, "calibrate_gradient_sequential: end\n");
01790 #endif
01791 }
01792
01800 void *
01801 calibrate_gradient_thread (ParallelData * data)
01802 {
01803   unsigned int i, j, thread;
01804   double e;
01805 #if DEBUG
01806   fprintf (stderr, "calibrate_gradient_thread: start\n");
01807 #endif
01808   thread = data->thread;
01809 #if DEBUG
01810   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01811            thread,
01812            calibrate->thread_gradient[thread],
01813            calibrate->thread_gradient[thread + 1]);
01814 #endif
01815   for (i = calibrate->thread_gradient[thread];
01816        i < calibrate->thread_gradient[thread + 1]; ++i)
```

```
01817     {
01818       e = 0.;
01819       for (j = 0; j < calibrate->nexperiments; ++j)
01820         e += calibrate_parse (i, j);
01821       g_mutex_lock (mutex);
01822       calibrate_best_gradient (i, e);
01823       calibrate_save_variables (i, e);
01824       g_mutex_unlock (mutex);
01825 #if DEBUG
01826       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01827 #endif
01828     }
01829 #if DEBUG
01830   fprintf (stderr, "calibrate_gradient_thread: end\n");
01831 #endif
01832   g_thread_exit (NULL);
01833   return NULL;
01834 }
01835
01845 double
01846 calibrate_estimate_gradient_random (unsigned int variable,
01847                                     unsigned int estimate)
01848 {
01849   double x;
01850 #if DEBUG
01851   fprintf (stderr, "calibrate_estimate_gradient_random: start\n");
01852 #endif
01853   x = calibrate->gradient[variable]
01854     + (1. - 2. * gsl_rng_uniform (calibrate->rng)) * calibrate->step[variable];
01855 #if DEBUG
01856   fprintf (stderr, "calibrate_estimate_gradient_random: gradient%u=%lg\n",
01857            variable, x);
01858   fprintf (stderr, "calibrate_estimate_gradient_random: end\n");
01859 #endif
01860   return x;
01861 }
01862
01872 double
01873 calibrate_estimate_gradient_coordinates (unsigned int variable,
01874                                          unsigned int estimate)
01875 {
01876   double x;
01877 #if DEBUG
01878   fprintf (stderr, "calibrate_estimate_gradient_coordinates: start\n");
01879 #endif
01880   x = calibrate->gradient[variable];
01881   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
01882     {
01883       if (estimate & 1)
01884         x += calibrate->step[variable];
01885       else
01886         x -= calibrate->step[variable];
01887     }
01888 #if DEBUG
01889   fprintf (stderr, "calibrate_estimate_gradient_coordinates: gradient%u=%lg\n",
01890            variable, x);
01891   fprintf (stderr, "calibrate_estimate_gradient_coordinates: end\n");
01892 #endif
01893   return x;
01894 }
01895
01902 void
01903 calibrate_step_gradient (unsigned int simulation)
01904 {
01905   GThread *thread[nthreads_gradient];
01906   ParallelData data[nthreads_gradient];
01907   unsigned int i, j, k, b;
01908 #if DEBUG
01909   fprintf (stderr, "calibrate_step_gradient: start\n");
01910 #endif
01911   for (i = 0; i < calibrate->nestimates; ++i)
01912     {
01913       k = (simulation + i) * calibrate->nvariables;
01914       b = calibrate->simulation_best[0] * calibrate->nvariables;
01915 #if DEBUG
01916       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01917                simulation + i, calibrate->simulation_best[0]);
01918 #endif
01919       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01920         {
01921 #if DEBUG
01922           fprintf (stderr,
01923                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01924                    i, j, calibrate->value[b]);
01925 #endif
01926           calibrate->value[k]
01927             = calibrate->value[b] + calibrate_estimate_gradient (j, i);
```

```
01928           calibrate->value[k] = fmin (fmax (calibrate->value[k],
01929                                             calibrate->rangeminabs[j]),
01930                                       calibrate->rangemaxabs[j]);
01931 #if DEBUG
01932           fprintf (stderr,
01933                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01934                    i, j, calibrate->value[k]);
01935 #endif
01936         }
01937     }
01938   if (nthreads_gradient == 1)
01939     calibrate_gradient_sequential (simulation);
01940   else
01941     {
01942       for (i = 0; i <= nthreads_gradient; ++i)
01943         {
01944           calibrate->thread_gradient[i]
01945             = simulation + calibrate->nstart_gradient
01946             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01947             / nthreads_gradient;
01948 #if DEBUG
01949           fprintf (stderr,
01950                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01951                    i, calibrate->thread_gradient[i]);
01952 #endif
01953         }
01954       for (i = 0; i < nthreads_gradient; ++i)
01955         {
01956           data[i].thread = i;
01957           thread[i] = g_thread_new
01958             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01959         }
01960       for (i = 0; i < nthreads_gradient; ++i)
01961         g_thread_join (thread[i]);
01962     }
01963 #if DEBUG
01964   fprintf (stderr, "calibrate_step_gradient: end\n");
01965 #endif
01966 }
01967
01972 void
01973 calibrate_gradient ()
01974 {
01975   unsigned int i, j, k, b, s;
01976 #if DEBUG
01977   fprintf (stderr, "calibrate_gradient: start\n");
01978 #endif
01979   for (i = 0; i < calibrate->nvariables; ++i)
01980     calibrate->gradient[i] = 0.;
01981   b = calibrate->simulation_best[0] * calibrate->nvariables;
01982   s = calibrate->nsimulations;
01983   for (i = 0; i < calibrate->nsteps; ++i, s += calibrate->nestimates, b = k)
01984     {
01985 #if DEBUG
01986       fprintf (stderr, "calibrate_gradient: step=%u old_best=%u\n",
01987                i, calibrate->simulation_best[0]);
01988 #endif
01989       calibrate_step_gradient (s);
01990       k = calibrate->simulation_best[0] * calibrate->nvariables;
01991 #if DEBUG
01992       fprintf (stderr, "calibrate_gradient: step=%u best=%u\n",
01993                i, calibrate->simulation_best[0]);
01994 #endif
01995       if (k == b)
01996         for (j = 0; j < calibrate->nvariables; ++j)
01997           calibrate->gradient[j] = 0.;
01998       else
01999         for (j = 0; j < calibrate->nvariables; ++j)
02000           {
02001 #if DEBUG
02002             fprintf (stderr, "calibrate_gradient: best%u=%.14le old%u=%.14le\n",
02003                      j, calibrate->value[k + j], j, calibrate->value[b + j]);
02004 #endif
02005             calibrate->gradient[j]
02006               = (1. - calibrate->relaxation) * calibrate->gradient[j]
02007               + calibrate->relaxation
02008               * (calibrate->value[k + j] - calibrate->value[b + j]);
02009 #if DEBUG
02010             fprintf (stderr, "calibrate_gradient: gradient%u=%.14le\n",
02011                      j, calibrate->gradient[j]);
02012 #endif
02013           }
02014     }
02015 #if DEBUG
02016   fprintf (stderr, "calibrate_gradient: end\n");
02017 #endif
```

```
02018 }
02019
02027 double
02028 calibrate_genetic_objective (Entity * entity)
02029 {
02030   unsigned int j;
02031   double objective;
02032   char buffer[64];
02033 #if DEBUG
02034   fprintf (stderr, "calibrate_genetic_objective: start\n");
02035 #endif
02036   for (j = 0; j < calibrate->nvariables; ++j)
02037     {
02038       calibrate->value[entity->id * calibrate->nvariables + j]
02039         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02040     }
02041   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02042     objective += calibrate_parse (entity->id, j);
02043   g_mutex_lock (mutex);
02044   for (j = 0; j < calibrate->nvariables; ++j)
02045     {
02046       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02047       fprintf (calibrate->file_variables, buffer,
02048                genetic_get_variable (entity, calibrate->genetic_variable + j));
02049     }
02050   fprintf (calibrate->file_variables, "%.14le\n", objective);
02051   g_mutex_unlock (mutex);
02052 #if DEBUG
02053   fprintf (stderr, "calibrate_genetic_objective: end\n");
02054 #endif
02055   return objective;
02056 }
02057
02062 void
02063 calibrate_genetic ()
02064 {
02065   char *best_genome;
02066   double best_objective, *best_variable;
02067 #if DEBUG
02068   fprintf (stderr, "calibrate_genetic: start\n");
02069   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
02070            nthreads);
02071   fprintf (stderr,
02072            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
02073            calibrate->nvariables, calibrate->nsimulations,
02074            calibrate->niterations);
02075   fprintf (stderr,
02076            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
02077            calibrate->mutation_ratio, calibrate->
      reproduction_ratio,
02078            calibrate->adaptation_ratio);
02079 #endif
02080   genetic_algorithm_default (calibrate->nvariables,
02081                              calibrate->genetic_variable,
02082                              calibrate->nsimulations,
02083                              calibrate->niterations,
02084                              calibrate->mutation_ratio,
02085                              calibrate->reproduction_ratio,
02086                              calibrate->adaptation_ratio,
02087                              &calibrate_genetic_objective,
02088                              &best_genome, &best_variable, &best_objective);
02089 #if DEBUG
02090   fprintf (stderr, "calibrate_genetic: the best\n");
02091 #endif
02092   calibrate->error_old = (double *) g_malloc (sizeof (double));
02093   calibrate->value_old
02094     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
02095   calibrate->error_old[0] = best_objective;
02096   memcpy (calibrate->value_old, best_variable,
02097           calibrate->nvariables * sizeof (double));
02098   g_free (best_genome);
02099   g_free (best_variable);
02100   calibrate_print ();
02101 #if DEBUG
02102   fprintf (stderr, "calibrate_genetic: end\n");
02103 #endif
02104 }
02105
02110 void
02111 calibrate_save_old ()
02112 {
02113   unsigned int i, j;
02114 #if DEBUG
02115   fprintf (stderr, "calibrate_save_old: start\n");
02116   fprintf (stderr, "calibrate_save_old: nsaveds=%u\n", calibrate->nsaveds);
02117 #endif
02118   memcpy (calibrate->error_old, calibrate->error_best,
```

```
02119           calibrate->nbest * sizeof (double));
02120   for (i = 0; i < calibrate->nbest; ++i)
02121     {
02122       j = calibrate->simulation_best[i];
02123 #if DEBUG
02124       fprintf (stderr, "calibrate_save_old: i=%u j=%u\n", i, j);
02125 #endif
02126       memcpy (calibrate->value_old + i * calibrate->nvariables,
02127               calibrate->value + j * calibrate->nvariables,
02128               calibrate->nvariables * sizeof (double));
02129     }
02130 #if DEBUG
02131   for (i = 0; i < calibrate->nvariables; ++i)
02132     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
02133              i, calibrate->value_old[i]);
02134   fprintf (stderr, "calibrate_save_old: end\n");
02135 #endif
02136 }
02137
02143 void
02144 calibrate_merge_old ()
02145 {
02146   unsigned int i, j, k;
02147   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
02148     *enew, *eold;
02149 #if DEBUG
02150   fprintf (stderr, "calibrate_merge_old: start\n");
02151 #endif
02152   enew = calibrate->error_best;
02153   eold = calibrate->error_old;
02154   i = j = k = 0;
02155   do
02156     {
02157       if (*enew < *eold)
02158         {
02159           memcpy (v + k * calibrate->nvariables,
02160                   calibrate->value
02161                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
02162                   calibrate->nvariables * sizeof (double));
02163           e[k] = *enew;
02164           ++k;
02165           ++enew;
02166           ++i;
02167         }
02168       else
02169         {
02170           memcpy (v + k * calibrate->nvariables,
02171                   calibrate->value_old + j * calibrate->nvariables,
02172                   calibrate->nvariables * sizeof (double));
02173           e[k] = *eold;
02174           ++k;
02175           ++eold;
02176           ++j;
02177         }
02178     }
02179   while (k < calibrate->nbest);
02180   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
02181   memcpy (calibrate->error_old, e, k * sizeof (double));
02182 #if DEBUG
02183   fprintf (stderr, "calibrate_merge_old: end\n");
02184 #endif
02185 }
02186
02192 void
02193 calibrate_refine ()
02194 {
02195   unsigned int i, j;
02196   double d;
02197 #if HAVE_MPI
02198   MPI_Status mpi_stat;
02199 #endif
02200 #if DEBUG
02201   fprintf (stderr, "calibrate_refine: start\n");
02202 #endif
02203 #if HAVE_MPI
02204   if (!calibrate->mpi_rank)
02205     {
02206 #endif
02207       for (j = 0; j < calibrate->nvariables; ++j)
02208         {
02209           calibrate->rangemin[j] = calibrate->rangemax[j]
02210             = calibrate->value_old[j];
02211         }
02212       for (i = 0; ++i < calibrate->nbest;)
02213         {
```

```
02214           for (j = 0; j < calibrate->nvariables; ++j)
02215             {
02216               calibrate->rangemin[j]
02217                 = fmin (calibrate->rangemin[j],
02218                         calibrate->value_old[i * calibrate->nvariables + j]);
02219               calibrate->rangemax[j]
02220                 = fmax (calibrate->rangemax[j],
02221                         calibrate->value_old[i * calibrate->nvariables + j]);
02222             }
02223         }
02224       for (j = 0; j < calibrate->nvariables; ++j)
02225         {
02226           d = calibrate->tolerance
02227             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
02228           switch (calibrate->algorithm)
02229             {
02230             case ALGORITHM_MONTE_CARLO:
02231               d *= 0.5;
02232               break;
02233             default:
02234               if (calibrate->nsweeps[j] > 1)
02235                 d /= calibrate->nsweeps[j] - 1;
02236               else
02237                 d = 0.;
02238             }
02239           calibrate->rangemin[j] -= d;
02240           calibrate->rangemin[j]
02241             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
02242           calibrate->rangemax[j] += d;
02243           calibrate->rangemax[j]
02244             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
02245           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
02246                   calibrate->rangemin[j], calibrate->rangemax[j]);
02247           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
02248                    calibrate->label[j], calibrate->rangemin[j],
02249                    calibrate->rangemax[j]);
02250         }
02251 #if HAVE_MPI
02252       for (i = 1; i < ntasks; ++i)
02253         {
02254           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
02255                     1, MPI_COMM_WORLD);
02256           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
02257                     1, MPI_COMM_WORLD);
02258         }
02259     }
02260   else
02261     {
02262       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02263                 MPI_COMM_WORLD, &mpi_stat);
02264       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
02265                 MPI_COMM_WORLD, &mpi_stat);
02266     }
02267 #endif
02268 #if DEBUG
02269   fprintf (stderr, "calibrate_refine: end\n");
02270 #endif
02271 }
02272
02277 void
02278 calibrate_step ()
02279 {
02280 #if DEBUG
02281   fprintf (stderr, "calibrate_algorithm: start\n");
02282 #endif
02283   calibrate_algorithm ();
02284   if (calibrate->nsteps)
02285     calibrate_gradient ();
02286 #if DEBUG
02287   fprintf (stderr, "calibrate_gradient: end\n");
02288 #endif
02289 }
02290
02295 void
02296 calibrate_iterate ()
02297 {
02298   unsigned int i;
02299 #if DEBUG
02300   fprintf (stderr, "calibrate_iterate: start\n");
02301 #endif
02302   calibrate->error_old
02303     = (double *) g_malloc (calibrate->nbest * sizeof (double));
02304   calibrate->value_old = (double *)
02305     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
02306   calibrate_step ();
02307   calibrate_save_old ();
02308   calibrate_refine ();
```

```
02309   calibrate_print ();
02310   for (i = 1; i < calibrate->niterations; ++i)
02311     {
02312       calibrate_step ();
02313       calibrate_merge_old ();
02314       calibrate_refine ();
02315       calibrate_print ();
02316     }
02317 #if DEBUG
02318   fprintf (stderr, "calibrate_iterate: end\n");
02319 #endif
02320 }
02321
02326 void
02327 calibrate_free ()
02328 {
02329   unsigned int i, j;
02330 #if DEBUG
02331   fprintf (stderr, "calibrate_free: start\n");
02332 #endif
02333   for (j = 0; j < calibrate->ninputs; ++j)
02334     {
02335       for (i = 0; i < calibrate->nexperiments; ++i)
02336         g_mapped_file_unref (calibrate->file[j][i]);
02337       g_free (calibrate->file[j]);
02338     }
02339   g_free (calibrate->error_old);
02340   g_free (calibrate->value_old);
02341   g_free (calibrate->value);
02342   g_free (calibrate->genetic_variable);
02343   g_free (calibrate->rangemax);
02344   g_free (calibrate->rangemin);
02345 #if DEBUG
02346   fprintf (stderr, "calibrate_free: end\n");
02347 #endif
02348 }
02349
02354 void
02355 calibrate_open ()
02356 {
02357   GTimeZone *tz;
02358   GDateTime *t0, *t;
02359   unsigned int i, j, *nbits;
02360
02361 #if DEBUG
02362   char *buffer;
02363   fprintf (stderr, "calibrate_open: start\n");
02364 #endif
02365
02366   // Getting initial time
02367 #if DEBUG
02368   fprintf (stderr, "calibrate_open: getting initial time\n");
02369 #endif
02370   tz = g_time_zone_new_utc ();
02371   t0 = g_date_time_new_now (tz);
02372
02373   // Obtaining and initing the pseudo-random numbers generator seed
02374 #if DEBUG
02375   fprintf (stderr, "calibrate_open: getting initial seed\n");
02376 #endif
02377   calibrate->seed = input->seed;
02378   gsl_rng_set (calibrate->rng, calibrate->seed);
02379
02380   // Replacing the working directory
02381 #if DEBUG
02382   fprintf (stderr, "calibrate_open: replacing the working directory\n");
02383 #endif
02384   g_chdir (input->directory);
02385
02386   // Getting results file names
02387   calibrate->result = input->result;
02388   calibrate->variables = input->variables;
02389
02390   // Obtaining the simulator file
02391   calibrate->simulator = input->simulator;
02392
02393   // Obtaining the evaluator file
02394   calibrate->evaluator = input->evaluator;
02395
02396   // Reading the algorithm
02397   calibrate->algorithm = input->algorithm;
02398   switch (calibrate->algorithm)
02399     {
02400     case ALGORITHM_MONTE_CARLO:
02401       calibrate_algorithm = calibrate_MonteCarlo;
02402       break;
02403     case ALGORITHM_SWEEP:
```

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

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

```
02570   fprintf (stderr, "calibrate_open: nstart=%u nend=%u\n", calibrate->nstart,
02571            calibrate->nend);
02572 #endif
02573
02574   // Calculating simulations to perform for each thread
02575   calibrate->thread
02576     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02577   for (i = 0; i <= nthreads; ++i)
02578     {
02579       calibrate->thread[i] = calibrate->nstart
02580         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02581 #if DEBUG
02582       fprintf (stderr, "calibrate_open: i=%u thread=%u\n", i,
02583                calibrate->thread[i]);
02584 #endif
02585     }
02586   if (calibrate->nsteps)
02587       calibrate->thread_gradient = (unsigned int *)
02588         alloca ((1 + nthreads_gradient) * sizeof (unsigned int));
02589
02590   // Opening result files
02591   calibrate->file_result = g_fopen (calibrate->result, "w");
02592   calibrate->file_variables = g_fopen (calibrate->variables, "w");
02593
02594   // Performing the algorithm
02595   switch (calibrate->algorithm)
02596     {
02597       // Genetic algorithm
02598     case ALGORITHM_GENETIC:
02599       calibrate_genetic ();
02600       break;
02601
02602       // Iterative algorithm
02603     default:
02604       calibrate_iterate ();
02605     }
02606
02607   // Getting calculation time
02608   t = g_date_time_new_now (tz);
02609   calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02610   g_date_time_unref (t);
02611   g_date_time_unref (t0);
02612   g_time_zone_unref (tz);
02613   printf ("%s = %.6lg s\n",
02614           gettext ("Calculation time"), calibrate->calculation_time);
02615   fprintf (calibrate->file_result, "%s = %.6lg s\n",
02616            gettext ("Calculation time"), calibrate->calculation_time);
02617
02618   // Closing result files
02619   fclose (calibrate->file_variables);
02620   fclose (calibrate->file_result);
02621
02622 #if DEBUG
02623   fprintf (stderr, "calibrate_open: end\n");
02624 #endif
02625 }
02626
02627 #if HAVE_GTK
02628
02635 void
02636 input_save_gradient (xmlNode * node)
02637 {
02638   if (input->nsteps)
02639     {
02640       xml_node_set_uint (node, XML_NSTEPS, input->
      nsteps);
02641       if (input->relaxation != DEFAULT_RELAXATION)
02642         xml_node_set_float (node, XML_RELAXATION, input->
      relaxation);
02643       switch (input->gradient_method)
02644         {
02645         case GRADIENT_METHOD_COORDINATES:
02646           xmlSetProp (node, XML_GRADIENT_METHOD,
      XML_COORDINATES);
02647           break;
02648         default:
02649           xmlSetProp (node, XML_GRADIENT_METHOD, XML_RANDOM);
02650           xml_node_set_uint (node, XML_NESTIMATES, input->
      nestimates);
02651         }
02652     }
02653 }
02654
02661 void
02662 input_save (char *filename)
02663 {
02664   unsigned int i, j;
```

```
02665   char *buffer;
02666   xmlDoc *doc;
02667   xmlNode *node, *child;
02668   GFile *file, *file2;
02669
02670   // Getting the input file directory
02671   input->name = g_path_get_basename (filename);
02672   input->directory = g_path_get_dirname (filename);
02673   file = g_file_new_for_path (input->directory);
02674
02675   // Opening the input file
02676   doc = xmlNewDoc ((const xmlChar *) "1.0");
02677
02678   // Setting root XML node
02679   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02680   xmlDocSetRootElement (doc, node);
02681
02682   // Adding properties to the root XML node
02683   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02684     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02685   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02686     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02687   file2 = g_file_new_for_path (input->simulator);
02688   buffer = g_file_get_relative_path (file, file2);
02689   g_object_unref (file2);
02690   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02691   g_free (buffer);
02692   if (input->evaluator)
02693     {
02694       file2 = g_file_new_for_path (input->evaluator);
02695       buffer = g_file_get_relative_path (file, file2);
02696       g_object_unref (file2);
02697       if (xmlStrlen ((xmlChar *) buffer))
02698         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02699       g_free (buffer);
02700     }
02701   if (input->seed != DEFAULT_RANDOM_SEED)
02702     xml_node_set_uint (node, XML_SEED, input->seed);
02703
02704   // Setting the algorithm
02705   buffer = (char *) g_malloc (64);
02706   switch (input->algorithm)
02707     {
02708     case ALGORITHM_MONTE_CARLO:
02709       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02710       snprintf (buffer, 64, "%u", input->nsimulations);
02711       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02712       snprintf (buffer, 64, "%u", input->niterations);
02713       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02714       snprintf (buffer, 64, "%.3lg", input->tolerance);
02715       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02716       snprintf (buffer, 64, "%u", input->nbest);
02717       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02718       input_save_gradient (node);
02719       break;
02720     case ALGORITHM_SWEEP:
02721       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02722       snprintf (buffer, 64, "%u", input->niterations);
02723       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02724       snprintf (buffer, 64, "%.3lg", input->tolerance);
02725       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02726       snprintf (buffer, 64, "%u", input->nbest);
02727       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02728       input_save_gradient (node);
02729       break;
02730     default:
02731       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02732       snprintf (buffer, 64, "%u", input->nsimulations);
02733       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02734       snprintf (buffer, 64, "%u", input->niterations);
02735       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02736       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02737       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02738       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02739       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02740       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02741       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02742       break;
02743     }
02744   g_free (buffer);
02745
02746   // Setting the experimental data
02747   for (i = 0; i < input->nexperiments; ++i)
02748     {
02749       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02750       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02751       if (input->weight[i] != 1.)
```

```
02752         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02753       for (j = 0; j < input->ninputs; ++j)
02754         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02755     }
02756
02757   // Setting the variables data
02758   for (i = 0; i < input->nvariables; ++i)
02759     {
02760       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02761       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02762       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02763       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02764         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02765       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02766       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02767         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02768       if (input->precision[i] != DEFAULT_PRECISION)
02769         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02770       if (input->algorithm == ALGORITHM_SWEEP)
02771         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02772       else if (input->algorithm == ALGORITHM_GENETIC)
02773         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02774     }
02775
02776   // Saving the XML file
02777   xmlSaveFormatFile (filename, doc, 1);
02778
02779   // Freeing memory
02780   xmlFreeDoc (doc);
02781 }
02782
02787 void
02788 options_new ()
02789 {
02790   options->label_seed = (GtkLabel *)
02791     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02792   options->spin_seed = (GtkSpinButton *)
02793     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02794   gtk_widget_set_tooltip_text
02795     (GTK_WIDGET (options->spin_seed),
02796      gettext ("Seed to init the pseudo-random numbers generator"));
02797   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02798   options->label_threads = (GtkLabel *)
02799     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
02800   options->spin_threads
02801     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02802   gtk_widget_set_tooltip_text
02803     (GTK_WIDGET (options->spin_threads),
02804      gettext ("Number of threads to perform the calibration/optimization for "
02805               "the stochastic algorithm"));
02806   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
02807   options->label_gradient = (GtkLabel *)
02808     gtk_label_new (gettext ("Threads number for the gradient based method"));
02809   options->spin_gradient
02810     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02811   gtk_widget_set_tooltip_text
02812     (GTK_WIDGET (options->spin_gradient),
02813      gettext ("Number of threads to perform the calibration/optimization for "
02814               "the gradient based method"));
02815   gtk_spin_button_set_value (options->spin_gradient,
02816                              (gdouble) nthreads_gradient);
02817   options->grid = (GtkGrid *) gtk_grid_new ();
02818   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
02819   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
02820   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
02821                    0, 1, 1, 1);
02822   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
02823                    1, 1, 1, 1);
02824   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_gradient),
02825                    0, 2, 1, 1);
02826   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_gradient),
02827                    1, 2, 1, 1);
02828   gtk_widget_show_all (GTK_WIDGET (options->grid));
02829   options->dialog = (GtkDialog *)
02830     gtk_dialog_new_with_buttons (gettext ("Options"),
02831                                  window->window,
02832                                  GTK_DIALOG_MODAL,
02833                                  gettext ("_OK"), GTK_RESPONSE_OK,
```

```
02834                                 gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02835                                 NULL);
02836   gtk_container_add
02837     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02838      GTK_WIDGET (options->grid));
02839   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02840     {
02841       input->seed
02842         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02843       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
02844       nthreads_gradient
02845         = gtk_spin_button_get_value_as_int (options->spin_gradient);
02846     }
02847   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02848 }
02849
02854 void
02855 running_new ()
02856 {
02857 #if DEBUG
02858   fprintf (stderr, "running_new: start\n");
02859 #endif
02860   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02861   running->dialog = (GtkDialog *)
02862     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02863                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
02864   gtk_container_add
02865     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02866      GTK_WIDGET (running->label));
02867   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02868 #if DEBUG
02869   fprintf (stderr, "running_new: end\n");
02870 #endif
02871 }
02872
02878 int
02879 window_get_algorithm ()
02880 {
02881   unsigned int i;
02882   for (i = 0; i < NALGORITHMS; ++i)
02883     if (gtk_toggle_button_get_active
02884         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02885       break;
02886   return i;
02887 }
02888
02894 int
02895 window_get_gradient ()
02896 {
02897   unsigned int i;
02898   for (i = 0; i < NGRADIENTS; ++i)
02899     if (gtk_toggle_button_get_active
02900         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02901       break;
02902   return i;
02903 }
02904
02909 void
02910 window_save_gradient ()
02911 {
02912 #if DEBUG
02913   fprintf (stderr, "window_save_gradient: start\n");
02914 #endif
02915   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
02916     {
02917       input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
02918       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
02919       switch (window_get_gradient ())
02920         {
02921         case GRADIENT_METHOD_COORDINATES:
02922           input->gradient_method = GRADIENT_METHOD_COORDINATES;
02923           break;
02924         default:
02925           input->gradient_method = GRADIENT_METHOD_RANDOM;
02926           input->nestimates
02927             = gtk_spin_button_get_value_as_int (window->spin_estimates);
02928         }
02929     }
02930   else
02931     input->nsteps = 0;
02932 #if DEBUG
02933   fprintf (stderr, "window_save_gradient: end\n");
02934 #endif
02935 }
02936
02942 int
```

```
02943 window_save ()
02944 {
02945   char *buffer;
02946   GtkFileChooserDialog *dlg;
02947
02948 #if DEBUG
02949   fprintf (stderr, "window_save: start\n");
02950 #endif
02951
02952   // Opening the saving dialog
02953   dlg = (GtkFileChooserDialog *)
02954     gtk_file_chooser_dialog_new (gettext ("Save file"),
02955                                  window->window,
02956                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02957                                  gettext ("_Cancel"),
02958                                  GTK_RESPONSE_CANCEL,
02959                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02960   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02961   buffer = g_build_filename (input->directory, input->name, NULL);
02962   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02963   g_free (buffer);
02964
02965   // If OK response then saving
02966   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02967     {
02968
02969       // Adding properties to the root XML node
02970       input->simulator = gtk_file_chooser_get_filename
02971         (GTK_FILE_CHOOSER (window->button_simulator));
02972       if (gtk_toggle_button_get_active
02973           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02974         input->evaluator = gtk_file_chooser_get_filename
02975           (GTK_FILE_CHOOSER (window->button_evaluator));
02976       else
02977         input->evaluator = NULL;
02978       input->result
02979         = (char *) xmlStrdup ((const xmlChar *)
02980                               gtk_entry_get_text (window->entry_result));
02981       input->variables
02982         = (char *) xmlStrdup ((const xmlChar *)
02983                               gtk_entry_get_text (window->entry_variables));
02984
02985       // Setting the algorithm
02986       switch (window_get_algorithm ())
02987         {
02988         case ALGORITHM_MONTE_CARLO:
02989           input->algorithm = ALGORITHM_MONTE_CARLO;
02990           input->nsimulations
02991             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02992           input->niterations
02993             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02994           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02995           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02996           window_save_gradient ();
02997           break;
02998         case ALGORITHM_SWEEP:
02999           input->algorithm = ALGORITHM_SWEEP;
03000           input->niterations
03001             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03003           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03004           window_save_gradient ();
03005           break;
03006         default:
03007           input->algorithm = ALGORITHM_GENETIC;
03008           input->nsimulations
03009             = gtk_spin_button_get_value_as_int (window->spin_population);
03010           input->niterations
03011             = gtk_spin_button_get_value_as_int (window->spin_generations);
03012           input->mutation_ratio
03013             = gtk_spin_button_get_value (window->spin_mutation);
03014           input->reproduction_ratio
03015             = gtk_spin_button_get_value (window->spin_reproduction);
03016           input->adaptation_ratio
03017             = gtk_spin_button_get_value (window->spin_adaptation);
03018           break;
03019         }
03020
03021       // Saving the XML file
03022       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03023       input_save (buffer);
03024
03025       // Closing and freeing memory
```

```
03026       g_free (buffer);
03027       gtk_widget_destroy (GTK_WIDGET (dlg));
03028 #if DEBUG
03029       fprintf (stderr, "window_save: end\n");
03030 #endif
03031       return 1;
03032     }
03033
03034   // Closing and freeing memory
03035   gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037   fprintf (stderr, "window_save: end\n");
03038 #endif
03039   return 0;
03040 }
03041
03046 void
03047 window_run ()
03048 {
03049   unsigned int i;
03050   char *msg, *msg2, buffer[64], buffer2[64];
03051 #if DEBUG
03052   fprintf (stderr, "window_run: start\n");
03053 #endif
03054   if (!window_save ())
03055     {
03056 #if DEBUG
03057       fprintf (stderr, "window_run: end\n");
03058 #endif
03059       return;
03060     }
03061   running_new ();
03062   while (gtk_events_pending ())
03063     gtk_main_iteration ();
03064   calibrate_open ();
03065   gtk_widget_destroy (GTK_WIDGET (running->dialog));
03066   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
03067   msg2 = g_strdup (buffer);
03068   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
03069     {
03070       snprintf (buffer, 64, "%s = %s\n",
03071                 calibrate->label[i], format[calibrate->precision[i]]);
03072       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
03073       msg = g_strconcat (msg2, buffer2, NULL);
03074       g_free (msg2);
03075     }
03076   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
03077             calibrate->calculation_time);
03078   msg = g_strconcat (msg2, buffer, NULL);
03079   g_free (msg2);
03080   show_message (gettext ("Best result"), msg, INFO_TYPE);
03081   g_free (msg);
03082   calibrate_free ();
03083 #if DEBUG
03084   fprintf (stderr, "window_run: end\n");
03085 #endif
03086 }
03087
03092 void
03093 window_help ()
03094 {
03095   char *buffer, *buffer2;
03096   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
03097                               gettext ("user-manual.pdf"), NULL);
03098   buffer = g_filename_to_uri (buffer2, NULL, NULL);
03099   g_free (buffer2);
03100   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
03101   g_free (buffer);
03102 }
03103
03108 void
03109 window_about ()
03110 {
03111   static const gchar *authors[] = {
03112     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03113     "Borja Latorre Garcés <borja.latorre@csic.es>",
03114     NULL
03115   };
03116   gtk_show_about_dialog
03117     (window->window,
03118      "program_name", "Calibrator",
03119      "comments",
03120      gettext ("A software to perform calibrations/optimizations of empirical "
03121               "parameters"),
03122      "authors", authors,
03123      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
03124      "version", "1.3.6",
```

```
03125       "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
03126       "logo", window->logo,
03127       "website", "https://github.com/jburguete/calibrator",
03128       "license-type", GTK_LICENSE_BSD, NULL);
03129 }
03130
03136 void
03137 window_update_gradient ()
03138 {
03139   gtk_widget_show (GTK_WIDGET (window->check_gradient));
03140   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_gradient)))
03141     gtk_widget_show (GTK_WIDGET (window->grid_gradient));
03142   switch (window_get_gradient ())
03143     {
03144     case GRADIENT_METHOD_COORDINATES:
03145       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
03146       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
03147       break;
03148     default:
03149       gtk_widget_show (GTK_WIDGET (window->label_estimates));
03150       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
03151     }
03152 }
03153
03158 void
03159 window_update ()
03160 {
03161   unsigned int i;
03162   gtk_widget_set_sensitive
03163     (GTK_WIDGET (window->button_evaluator),
03164      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03165                                    (window->check_evaluator)));
03166   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
03167   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
03168   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
03169   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
03170   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
03171   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
03172   gtk_widget_hide (GTK_WIDGET (window->label_bests));
03173   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
03174   gtk_widget_hide (GTK_WIDGET (window->label_population));
03175   gtk_widget_hide (GTK_WIDGET (window->spin_population));
03176   gtk_widget_hide (GTK_WIDGET (window->label_generations));
03177   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
03178   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
03179   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
03180   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
03181   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
03182   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
03183   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
03184   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
03185   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
03186   gtk_widget_hide (GTK_WIDGET (window->label_bits));
03187   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
03188   gtk_widget_hide (GTK_WIDGET (window->check_gradient));
03189   gtk_widget_hide (GTK_WIDGET (window->grid_gradient));
03190   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
03191   switch (window_get_algorithm ())
03192     {
03193     case ALGORITHM_MONTE_CARLO:
03194       gtk_widget_show (GTK_WIDGET (window->label_simulations));
03195       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
03196       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03197       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03198       if (i > 1)
03199         {
03200           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03201           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03202           gtk_widget_show (GTK_WIDGET (window->label_bests));
03203           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03204         }
03205       window_update_gradient ();
03206       break;
03207     case ALGORITHM_SWEEP:
03208       gtk_widget_show (GTK_WIDGET (window->label_iterations));
03209       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
03210       if (i > 1)
03211         {
03212           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
03213           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
03214           gtk_widget_show (GTK_WIDGET (window->label_bests));
03215           gtk_widget_show (GTK_WIDGET (window->spin_bests));
03216         }
03217       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
03218       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
03219       gtk_widget_show (GTK_WIDGET (window->check_gradient));
03220       window_update_gradient ();
```

```
03221       break;
03222     default:
03223       gtk_widget_show (GTK_WIDGET (window->label_population));
03224       gtk_widget_show (GTK_WIDGET (window->spin_population));
03225       gtk_widget_show (GTK_WIDGET (window->label_generations));
03226       gtk_widget_show (GTK_WIDGET (window->spin_generations));
03227       gtk_widget_show (GTK_WIDGET (window->label_mutation));
03228       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
03229       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
03230       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
03231       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
03232       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
03233       gtk_widget_show (GTK_WIDGET (window->label_bits));
03234       gtk_widget_show (GTK_WIDGET (window->spin_bits));
03235     }
03236   gtk_widget_set_sensitive
03237     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
03238   gtk_widget_set_sensitive
03239     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
03240   for (i = 0; i < input->ninputs; ++i)
03241     {
03242       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03243       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03244       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
03245       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
03246       g_signal_handler_block
03247         (window->check_template[i], window->id_template[i]);
03248       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
03249       gtk_toggle_button_set_active
03250         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
03251       g_signal_handler_unblock
03252         (window->button_template[i], window->id_input[i]);
03253       g_signal_handler_unblock
03254         (window->check_template[i], window->id_template[i]);
03255     }
03256   if (i > 0)
03257     {
03258       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
03259       gtk_widget_set_sensitive
03260         (GTK_WIDGET (window->button_template[i - 1]),
03261          gtk_toggle_button_get_active
03262          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
03263     }
03264   if (i < MAX_NINPUTS)
03265     {
03266       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
03267       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
03268       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
03269       gtk_widget_set_sensitive
03270         (GTK_WIDGET (window->button_template[i]),
03271          gtk_toggle_button_get_active
03272          GTK_TOGGLE_BUTTON (window->check_template[i]));
03273       g_signal_handler_block
03274         (window->check_template[i], window->id_template[i]);
03275       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
03276       gtk_toggle_button_set_active
03277         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
03278       g_signal_handler_unblock
03279         (window->button_template[i], window->id_input[i]);
03280       g_signal_handler_unblock
03281         (window->check_template[i], window->id_template[i]);
03282     }
03283   while (++i < MAX_NINPUTS)
03284     {
03285       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
03286       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
03287     }
03288   gtk_widget_set_sensitive
03289     (GTK_WIDGET (window->spin_minabs),
03290      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
03291   gtk_widget_set_sensitive
03292     (GTK_WIDGET (window->spin_maxabs),
03293      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
03294 }
03295
03300 void
03301 window_set_algorithm ()
03302 {
03303   int i;
03304 #if DEBUG
03305   fprintf (stderr, "window_set_algorithm: start\n");
03306 #endif
03307   i = window_get_algorithm ();
```

```
03308   switch (i)
03309     {
03310     case ALGORITHM_SWEEP:
03311       input->nsweeps = (unsigned int *) g_realloc
03312         (input->nsweeps, input->nvariables * sizeof (unsigned int));
03313       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03314       if (i < 0)
03315         i = 0;
03316       gtk_spin_button_set_value (window->spin_sweeps,
03317                                  (gdouble) input->nsweeps[i]);
03318       break;
03319     case ALGORITHM_GENETIC:
03320       input->nbits = (unsigned int *) g_realloc
03321         (input->nbits, input->nvariables * sizeof (unsigned int));
03322       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03323       if (i < 0)
03324         i = 0;
03325       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03326     }
03327   window_update ();
03328 #if DEBUG
03329   fprintf (stderr, "window_set_algorithm: end\n");
03330 #endif
03331 }
03332
03337 void
03338 window_set_experiment ()
03339 {
03340   unsigned int i, j;
03341   char *buffer1, *buffer2;
03342 #if DEBUG
03343   fprintf (stderr, "window_set_experiment: start\n");
03344 #endif
03345   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03346   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
03347   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
03348   buffer2 = g_build_filename (input->directory, buffer1, NULL);
03349   g_free (buffer1);
03350   g_signal_handler_block
03351     (window->button_experiment, window->id_experiment_name);
03352   gtk_file_chooser_set_filename
03353     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
03354   g_signal_handler_unblock
03355     (window->button_experiment, window->id_experiment_name);
03356   g_free (buffer2);
03357   for (j = 0; j < input->ninputs; ++j)
03358     {
03359       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03360       buffer2
03361         = g_build_filename (input->directory, input->template[j][i], NULL);
03362       gtk_file_chooser_set_filename
03363         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
03364       g_free (buffer2);
03365       g_signal_handler_unblock
03366         (window->button_template[j], window->id_input[j]);
03367     }
03368 #if DEBUG
03369   fprintf (stderr, "window_set_experiment: end\n");
03370 #endif
03371 }
03372
03377 void
03378 window_remove_experiment ()
03379 {
03380   unsigned int i, j;
03381   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03382   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03383   gtk_combo_box_text_remove (window->combo_experiment, i);
03384   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03385   xmlFree (input->experiment[i]);
03386   --input->nexperiments;
03387   for (j = i; j < input->nexperiments; ++j)
03388     {
03389       input->experiment[j] = input->experiment[j + 1];
03390       input->weight[j] = input->weight[j + 1];
03391     }
03392   j = input->nexperiments - 1;
03393   if (i > j)
03394     i = j;
03395   for (j = 0; j < input->ninputs; ++j)
03396     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03397   g_signal_handler_block
```

```
03398     (window->button_experiment, window->id_experiment_name);
03399   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03400   g_signal_handler_unblock
03401     (window->button_experiment, window->id_experiment_name);
03402   for (j = 0; j < input->ninputs; ++j)
03403     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03404   window_update ();
03405 }
03406
03411 void
03412 window_add_experiment ()
03413 {
03414   unsigned int i, j;
03415   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03416   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03417   gtk_combo_box_text_insert_text
03418     (window->combo_experiment, i, input->experiment[i]);
03419   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03420   input->experiment = (char **) g_realloc
03421     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
03422   input->weight = (double *) g_realloc
03423     (input->weight, (input->nexperiments + 1) * sizeof (double));
03424   for (j = input->nexperiments - 1; j > i; --j)
03425     {
03426       input->experiment[j + 1] = input->experiment[j];
03427       input->weight[j + 1] = input->weight[j];
03428     }
03429   input->experiment[j + 1]
03430     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
03431   input->weight[j + 1] = input->weight[j];
03432   ++input->nexperiments;
03433   for (j = 0; j < input->ninputs; ++j)
03434     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
03435   g_signal_handler_block
03436     (window->button_experiment, window->id_experiment_name);
03437   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
03438   g_signal_handler_unblock
03439     (window->button_experiment, window->id_experiment_name);
03440   for (j = 0; j < input->ninputs; ++j)
03441     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
03442   window_update ();
03443 }
03444
03449 void
03450 window_name_experiment ()
03451 {
03452   unsigned int i;
03453   char *buffer;
03454   GFile *file1, *file2;
03455 #if DEBUG
03456   fprintf (stderr, "window_name_experiment: start\n");
03457 #endif
03458   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03459   file1
03460     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
03461   file2 = g_file_new_for_path (input->directory);
03462   buffer = g_file_get_relative_path (file2, file1);
03463   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03464   gtk_combo_box_text_remove (window->combo_experiment, i);
03465   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
03466   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
03467   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03468   g_free (buffer);
03469   g_object_unref (file2);
03470   g_object_unref (file1);
03471 #if DEBUG
03472   fprintf (stderr, "window_name_experiment: end\n");
03473 #endif
03474 }
03475
03480 void
03481 window_weight_experiment ()
03482 {
03483   unsigned int i;
03484 #if DEBUG
03485   fprintf (stderr, "window_weight_experiment: start\n");
03486 #endif
03487   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03488   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
03489 #if DEBUG
```

```
03490   fprintf (stderr, "window_weight_experiment: end\n");
03491 #endif
03492 }
03493
03499 void
03500 window_inputs_experiment ()
03501 {
03502   unsigned int j;
03503 #if DEBUG
03504   fprintf (stderr, "window_inputs_experiment: start\n");
03505 #endif
03506   j = input->ninputs - 1;
03507   if (j
03508       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03509                                         (window->check_template[j])))
03510     --input->ninputs;
03511   if (input->ninputs < MAX_NINPUTS
03512       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
03513                                        (window->check_template[j])))
03514     {
03515       ++input->ninputs;
03516       for (j = 0; j < input->ninputs; ++j)
03517         {
03518           input->template[j] = (char **)
03519             g_realloc (input->template[j], input->nvariables * sizeof (char *));
03520         }
03521     }
03522   window_update ();
03523 #if DEBUG
03524   fprintf (stderr, "window_inputs_experiment: end\n");
03525 #endif
03526 }
03527
03535 void
03536 window_template_experiment (void *data)
03537 {
03538   unsigned int i, j;
03539   char *buffer;
03540   GFile *file1, *file2;
03541 #if DEBUG
03542   fprintf (stderr, "window_template_experiment: start\n");
03543 #endif
03544   i = (size_t) data;
03545   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03546   file1
03547     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03548   file2 = g_file_new_for_path (input->directory);
03549   buffer = g_file_get_relative_path (file2, file1);
03550   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03551   g_free (buffer);
03552   g_object_unref (file2);
03553   g_object_unref (file1);
03554 #if DEBUG
03555   fprintf (stderr, "window_template_experiment: end\n");
03556 #endif
03557 }
03558
03563 void
03564 window_set_variable ()
03565 {
03566   unsigned int i;
03567 #if DEBUG
03568   fprintf (stderr, "window_set_variable: start\n");
03569 #endif
03570   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03571   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03572   gtk_entry_set_text (window->entry_variable, input->label[i]);
03573   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03574   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
03575   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03576   if (input->rangeminabs[i] != -G_MAXDOUBLE)
03577     {
03578       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
03579       gtk_toggle_button_set_active
03580         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03581     }
03582   else
03583     {
03584       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03585       gtk_toggle_button_set_active
03586         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03587     }
03588   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03589     {
```

```
03590       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
03591       gtk_toggle_button_set_active
03592         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03593     }
03594   else
03595     {
03596       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03597       gtk_toggle_button_set_active
03598         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03599     }
03600   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
03601 #if DEBUG
03602   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03603            input->precision[i]);
03604 #endif
03605   switch (window_get_algorithm ())
03606     {
03607     case ALGORITHM_SWEEP:
03608       gtk_spin_button_set_value (window->spin_sweeps,
03609                                  (gdouble) input->nsweeps[i]);
03610 #if DEBUG
03611       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03612                input->nsweeps[i]);
03613 #endif
03614       break;
03615     case ALGORITHM_GENETIC:
03616       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
03617 #if DEBUG
03618       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03619                input->nbits[i]);
03620 #endif
03621       break;
03622     }
03623   window_update ();
03624 #if DEBUG
03625   fprintf (stderr, "window_set_variable: end\n");
03626 #endif
03627 }
03628
03633 void
03634 window_remove_variable ()
03635 {
03636   unsigned int i, j;
03637   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03638   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03639   gtk_combo_box_text_remove (window->combo_variable, i);
03640   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03641   xmlFree (input->label[i]);
03642   --input->nvariables;
03643   for (j = i; j < input->nvariables; ++j)
03644     {
03645       input->label[j] = input->label[j + 1];
03646       input->rangemin[j] = input->rangemin[j + 1];
03647       input->rangemax[j] = input->rangemax[j + 1];
03648       input->rangeminabs[j] = input->rangeminabs[j + 1];
03649       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03650       input->precision[j] = input->precision[j + 1];
03651       switch (window_get_algorithm ())
03652         {
03653         case ALGORITHM_SWEEP:
03654           input->nsweeps[j] = input->nsweeps[j + 1];
03655           break;
03656         case ALGORITHM_GENETIC:
03657           input->nbits[j] = input->nbits[j + 1];
03658         }
03659     }
03660   j = input->nvariables - 1;
03661   if (i > j)
03662     i = j;
03663   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03664   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03665   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03666   window_update ();
03667 }
03668
03673 void
03674 window_add_variable ()
03675 {
03676   unsigned int i, j;
03677 #if DEBUG
```

```
03678   fprintf (stderr, "window_add_variable: start\n");
03679 #endif
03680   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03681   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03682   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03683   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03684   input->label = (char **) g_realloc
03685     (input->label, (input->nvariables + 1) * sizeof (char *));
03686   input->rangemin = (double *) g_realloc
03687     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03688   input->rangemax = (double *) g_realloc
03689     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03690   input->rangeminabs = (double *) g_realloc
03691     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03692   input->rangemaxabs = (double *) g_realloc
03693     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03694   input->precision = (unsigned int *) g_realloc
03695     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03696   for (j = input->nvariables - 1; j > i; --j)
03697     {
03698       input->label[j + 1] = input->label[j];
03699       input->rangemin[j + 1] = input->rangemin[j];
03700       input->rangemax[j + 1] = input->rangemax[j];
03701       input->rangeminabs[j + 1] = input->rangeminabs[j];
03702       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03703       input->precision[j + 1] = input->precision[j];
03704     }
03705   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03706   input->rangemin[j + 1] = input->rangemin[j];
03707   input->rangemax[j + 1] = input->rangemax[j];
03708   input->rangeminabs[j + 1] = input->rangeminabs[j];
03709   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03710   input->precision[j + 1] = input->precision[j];
03711   switch (window_get_algorithm ())
03712     {
03713     case ALGORITHM_SWEEP:
03714       input->nsweeps = (unsigned int *) g_realloc
03715         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03716       for (j = input->nvariables - 1; j > i; --j)
03717         input->nsweeps[j + 1] = input->nsweeps[j];
03718       input->nsweeps[j + 1] = input->nsweeps[j];
03719       break;
03720     case ALGORITHM_GENETIC:
03721       input->nbits = (unsigned int *) g_realloc
03722         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03723       for (j = input->nvariables - 1; j > i; --j)
03724         input->nbits[j + 1] = input->nbits[j];
03725       input->nbits[j + 1] = input->nbits[j];
03726     }
03727   ++input->nvariables;
03728   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03729   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03730   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03731   window_update ();
03732 #if DEBUG
03733   fprintf (stderr, "window_add_variable: end\n");
03734 #endif
03735 }
03736
03741 void
03742 window_label_variable ()
03743 {
03744   unsigned int i;
03745   const char *buffer;
03746 #if DEBUG
03747   fprintf (stderr, "window_label_variable: start\n");
03748 #endif
03749   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03750   buffer = gtk_entry_get_text (window->entry_variable);
03751   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03752   gtk_combo_box_text_remove (window->combo_variable, i);
03753   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03754   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03755   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03756 #if DEBUG
03757   fprintf (stderr, "window_label_variable: end\n");
03758 #endif
03759 }
03760
03765 void
```

```
03766 window_precision_variable ()
03767 {
03768   unsigned int i;
03769 #if DEBUG
03770   fprintf (stderr, "window_precision_variable: start\n");
03771 #endif
03772   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03773   input->precision[i]
03774     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03775   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03776   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03777   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03778   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03779 #if DEBUG
03780   fprintf (stderr, "window_precision_variable: end\n");
03781 #endif
03782 }
03783
03788 void
03789 window_rangemin_variable ()
03790 {
03791   unsigned int i;
03792 #if DEBUG
03793   fprintf (stderr, "window_rangemin_variable: start\n");
03794 #endif
03795   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03796   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03797 #if DEBUG
03798   fprintf (stderr, "window_rangemin_variable: end\n");
03799 #endif
03800 }
03801
03806 void
03807 window_rangemax_variable ()
03808 {
03809   unsigned int i;
03810 #if DEBUG
03811   fprintf (stderr, "window_rangemax_variable: start\n");
03812 #endif
03813   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03814   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03815 #if DEBUG
03816   fprintf (stderr, "window_rangemax_variable: end\n");
03817 #endif
03818 }
03819
03824 void
03825 window_rangeminabs_variable ()
03826 {
03827   unsigned int i;
03828 #if DEBUG
03829   fprintf (stderr, "window_rangeminabs_variable: start\n");
03830 #endif
03831   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03832   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03833 #if DEBUG
03834   fprintf (stderr, "window_rangeminabs_variable: end\n");
03835 #endif
03836 }
03837
03842 void
03843 window_rangemaxabs_variable ()
03844 {
03845   unsigned int i;
03846 #if DEBUG
03847   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03848 #endif
03849   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03850   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03851 #if DEBUG
03852   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03853 #endif
03854 }
03855
03860 void
03861 window_update_variable ()
03862 {
03863   int i;
03864 #if DEBUG
03865   fprintf (stderr, "window_update_variable: start\n");
03866 #endif
03867   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03868   if (i < 0)
03869     i = 0;
03870   switch (window_get_algorithm ())
```

```
03871     {
03872     case ALGORITHM_SWEEP:
03873       input->nsweeps[i]
03874         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03875 #if DEBUG
03876       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03877                input->nsweeps[i]);
03878 #endif
03879       break;
03880     case ALGORITHM_GENETIC:
03881       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03882 #if DEBUG
03883       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
03884                input->nbits[i]);
03885 #endif
03886     }
03887 #if DEBUG
03888   fprintf (stderr, "window_update_variable: end\n");
03889 #endif
03890 }
03891
03899 int
03900 window_read (char *filename)
03901 {
03902   unsigned int i;
03903   char *buffer;
03904 #if DEBUG
03905   fprintf (stderr, "window_read: start\n");
03906 #endif
03907
03908   // Reading new input file
03909   input_free ();
03910   if (!input_open (filename))
03911     return 0;
03912
03913   // Setting GTK+ widgets data
03914   gtk_entry_set_text (window->entry_result, input->result);
03915   gtk_entry_set_text (window->entry_variables, input->variables);
03916   buffer = g_build_filename (input->directory, input->simulator, NULL);
03917   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03918                                  (window->button_simulator), buffer);
03919   g_free (buffer);
03920   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03921                                 (size_t) input->evaluator);
03922   if (input->evaluator)
03923     {
03924       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03925       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03926                                      (window->button_evaluator), buffer);
03927       g_free (buffer);
03928     }
03929   gtk_toggle_button_set_active
03930     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03931   switch (input->algorithm)
03932     {
03933     case ALGORITHM_MONTE_CARLO:
03934       gtk_spin_button_set_value (window->spin_simulations,
03935                                  (gdouble) input->nsimulations);
03936     case ALGORITHM_SWEEP:
03937       gtk_spin_button_set_value (window->spin_iterations,
03938                                  (gdouble) input->niterations);
03939       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03940       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03941       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_gradient),
03942                                     input->nsteps);
03943       if (input->nsteps)
03944         {
03945           gtk_toggle_button_set_active
03946             (GTK_TOGGLE_BUTTON (window->button_gradient
03947                                 [input->gradient_method]), TRUE);
03948           gtk_spin_button_set_value (window->spin_steps,
03949                                      (gdouble) input->nsteps);
03950           gtk_spin_button_set_value (window->spin_relaxation,
03951                                      (gdouble) input->relaxation);
03952           switch (input->gradient_method)
03953             {
03954             case GRADIENT_METHOD_RANDOM:
03955               gtk_spin_button_set_value (window->spin_estimates,
03956                                          (gdouble) input->nestimates);
03957             }
03958         }
03959       break;
03960     default:
03961       gtk_spin_button_set_value (window->spin_population,
```

```
03962                                 (gdouble) input->nsimulations);
03963       gtk_spin_button_set_value (window->spin_generations,
03964                                  (gdouble) input->niterations);
03965       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03966       gtk_spin_button_set_value (window->spin_reproduction,
03967                                  input->reproduction_ratio);
03968       gtk_spin_button_set_value (window->spin_adaptation,
03969                                  input->adaptation_ratio);
03970     }
03971   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03972   g_signal_handler_block (window->button_experiment,
03973                           window->id_experiment_name);
03974   gtk_combo_box_text_remove_all (window->combo_experiment);
03975   for (i = 0; i < input->nexperiments; ++i)
03976     gtk_combo_box_text_append_text (window->combo_experiment,
03977                                     input->experiment[i]);
03978   g_signal_handler_unblock
03979     (window->button_experiment, window->id_experiment_name);
03980   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03981   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03982   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03983   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03984   gtk_combo_box_text_remove_all (window->combo_variable);
03985   for (i = 0; i < input->nvariables; ++i)
03986     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03987   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03988   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03989   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03990   window_set_variable ();
03991   window_update ();
03992
03993 #if DEBUG
03994   fprintf (stderr, "window_read: end\n");
03995 #endif
03996   return 1;
03997 }
03998
04003 void
04004 window_open ()
04005 {
04006   char *buffer, *directory, *name;
04007   GtkFileChooserDialog *dlg;
04008
04009 #if DEBUG
04010   fprintf (stderr, "window_open: start\n");
04011 #endif
04012
04013   // Saving a backup of the current input file
04014   directory = g_strdup (input->directory);
04015   name = g_strdup (input->name);
04016
04017   // Opening dialog
04018   dlg = (GtkFileChooserDialog *)
04019     gtk_file_chooser_dialog_new (gettext ("Open input file"),
04020                                  window->window,
04021                                  GTK_FILE_CHOOSER_ACTION_OPEN,
04022                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
04023                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
04024   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
04025     {
04026
04027       // Traying to open the input file
04028       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
04029       if (!window_read (buffer))
04030         {
04031 #if DEBUG
04032           fprintf (stderr, "window_open: error reading input file\n");
04033 #endif
04034           g_free (buffer);
04035
04036           // Reading backup file on error
04037           buffer = g_build_filename (directory, name, NULL);
04038           if (!input_open (buffer))
04039             {
04040
04041               // Closing on backup file reading error
04042 #if DEBUG
04043               fprintf (stderr, "window_read: error reading backup file\n");
04044 #endif
```

```
04045               g_free (buffer);
04046               break;
04047             }
04048           g_free (buffer);
04049         }
04050       else
04051         {
04052           g_free (buffer);
04053           break;
04054         }
04055     }
04056
04057   // Freeing and closing
04058   g_free (name);
04059   g_free (directory);
04060   gtk_widget_destroy (GTK_WIDGET (dlg));
04061 #if DEBUG
04062   fprintf (stderr, "window_open: end\n");
04063 #endif
04064 }
04065
04070 void
04071 window_new ()
04072 {
04073   unsigned int i;
04074   char *buffer, *buffer2, buffer3[64];
04075   GtkViewport *viewport;
04076   char *label_algorithm[NALGORITHMS] = {
04077     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
04078   };
04079   char *tip_algorithm[NALGORITHMS] = {
04080     gettext ("Monte-Carlo brute force algorithm"),
04081     gettext ("Sweep brute force algorithm"),
04082     gettext ("Genetic algorithm")
04083   };
04084   char *label_gradient[NGRADIENTS] = {
04085     gettext ("_Coordinates descent"), gettext ("_Random")
04086   };
04087   char *tip_gradient[NGRADIENTS] = {
04088     gettext ("Coordinates descent gradient estimate method"),
04089     gettext ("Random gradient estimate method")
04090   };
04091
04092   // Creating the window
04093   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
04094
04095   // Finish when closing the window
04096   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
04097
04098   // Setting the window title
04099   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
04100
04101   // Creating the open button
04102   window->button_open = (GtkToolButton *) gtk_tool_button_new
04103     (gtk_image_new_from_icon_name ("document-open",
04104                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04105      gettext ("Open"));
04106   g_signal_connect (window->button_open, "clicked", window_open, NULL);
04107
04108   // Creating the save button
04109   window->button_save = (GtkToolButton *) gtk_tool_button_new
04110     (gtk_image_new_from_icon_name ("document-save",
04111                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04112      gettext ("Save"));
04113   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
04114                     NULL);
04115
04116   // Creating the run button
04117   window->button_run = (GtkToolButton *) gtk_tool_button_new
04118     (gtk_image_new_from_icon_name ("system-run",
04119                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04120      gettext ("Run"));
04121   g_signal_connect (window->button_run, "clicked", window_run, NULL);
04122
04123   // Creating the options button
04124   window->button_options = (GtkToolButton *) gtk_tool_button_new
04125     (gtk_image_new_from_icon_name ("preferences-system",
04126                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04127      gettext ("Options"));
04128   g_signal_connect (window->button_options, "clicked", options_new, NULL);
04129
04130   // Creating the help button
04131   window->button_help = (GtkToolButton *) gtk_tool_button_new
04132     (gtk_image_new_from_icon_name ("help-browser",
04133                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04134      gettext ("Help"));
```

```
04135   g_signal_connect (window->button_help, "clicked", window_help, NULL);
04136
04137   // Creating the about button
04138   window->button_about = (GtkToolButton *) gtk_tool_button_new
04139     (gtk_image_new_from_icon_name ("help-about",
04140                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04141      gettext ("About"));
04142   g_signal_connect (window->button_about, "clicked", window_about, NULL);
04143
04144   // Creating the exit button
04145   window->button_exit = (GtkToolButton *) gtk_tool_button_new
04146     (gtk_image_new_from_icon_name ("application-exit",
04147                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
04148      gettext ("Exit"));
04149   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
04150
04151   // Creating the buttons bar
04152   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
04153   gtk_toolbar_insert
04154     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
04155   gtk_toolbar_insert
04156     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
04157   gtk_toolbar_insert
04158     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
04159   gtk_toolbar_insert
04160     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
04161   gtk_toolbar_insert
04162     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
04163   gtk_toolbar_insert
04164     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
04165   gtk_toolbar_insert
04166     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
04167   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
04168
04169   // Creating the simulator program label and entry
04170   window->label_simulator
04171     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
04172   window->button_simulator = (GtkFileChooserButton *)
04173     gtk_file_chooser_button_new (gettext ("Simulator program"),
04174                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04175   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
04176                                gettext ("Simulator program executable file"));
04177
04178   // Creating the evaluator program label and entry
04179   window->check_evaluator = (GtkCheckButton *)
04180     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
04181   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
04182   window->button_evaluator = (GtkFileChooserButton *)
04183     gtk_file_chooser_button_new (gettext ("Evaluator program"),
04184                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04185   gtk_widget_set_tooltip_text
04186     (GTK_WIDGET (window->button_evaluator),
04187      gettext ("Optional evaluator program executable file"));
04188
04189   // Creating the results files labels and entries
04190   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
04191   window->entry_result = (GtkEntry *) gtk_entry_new ();
04192   gtk_widget_set_tooltip_text
04193     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
04194   window->label_variables
04195     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
04196   window->entry_variables = (GtkEntry *) gtk_entry_new ();
04197   gtk_widget_set_tooltip_text
04198     (GTK_WIDGET (window->entry_variables),
04199      gettext ("All simulated results file"));
04200
04201   // Creating the files grid and attaching widgets
04202   window->grid_files = (GtkGrid *) gtk_grid_new ();
04203   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
04204                    0, 0, 1, 1);
04205   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
04206                    1, 0, 1, 1);
04207   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
04208                    2, 0, 1, 1);
04209   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
04210                    3, 0, 1, 1);
04211   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
04212                    0, 1, 1, 1);
04213   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
04214                    1, 1, 1, 1);
```

```
04215   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
04216                    2, 1, 1, 1);
04217   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
04218                    3, 1, 1, 1);
04219
04220   // Creating the algorithm properties
04221   window->label_simulations = (GtkLabel *) gtk_label_new
04222     (gettext ("Simulations number"));
04223   window->spin_simulations
04224     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04225   gtk_widget_set_tooltip_text
04226     (GTK_WIDGET (window->spin_simulations),
04227      gettext ("Number of simulations to perform for each iteration"));
04228   window->label_iterations = (GtkLabel *)
04229     gtk_label_new (gettext ("Iterations number"));
04230   window->spin_iterations
04231     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04232   gtk_widget_set_tooltip_text
04233     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
04234   g_signal_connect
04235     (window->spin_iterations, "value-changed", window_update, NULL);
04236   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
04237   window->spin_tolerance
04238     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04239   gtk_widget_set_tooltip_text
04240     (GTK_WIDGET (window->spin_tolerance),
04241      gettext ("Tolerance to set the variable interval on the next iteration"));
04242   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
04243   window->spin_bests
04244     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04245   gtk_widget_set_tooltip_text
04246     (GTK_WIDGET (window->spin_bests),
04247      gettext ("Number of best simulations used to set the variable interval "
04248               "on the next iteration"));
04249   window->label_population
04250     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
04251   window->spin_population
04252     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04253   gtk_widget_set_tooltip_text
04254     (GTK_WIDGET (window->spin_population),
04255      gettext ("Number of population for the genetic algorithm"));
04256   window->label_generations
04257     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
04258   window->spin_generations
04259     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
04260   gtk_widget_set_tooltip_text
04261     (GTK_WIDGET (window->spin_generations),
04262      gettext ("Number of generations for the genetic algorithm"));
04263   window->label_mutation
04264     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
04265   window->spin_mutation
04266     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04267   gtk_widget_set_tooltip_text
04268     (GTK_WIDGET (window->spin_mutation),
04269      gettext ("Ratio of mutation for the genetic algorithm"));
04270   window->label_reproduction
04271     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
04272   window->spin_reproduction
04273     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04274   gtk_widget_set_tooltip_text
04275     (GTK_WIDGET (window->spin_reproduction),
04276      gettext ("Ratio of reproduction for the genetic algorithm"));
04277   window->label_adaptation
04278     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
04279   window->spin_adaptation
04280     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04281   gtk_widget_set_tooltip_text
04282     (GTK_WIDGET (window->spin_adaptation),
04283      gettext ("Ratio of adaptation for the genetic algorithm"));
04284
04285   // Creating the gradient based method properties
04286   window->check_gradient = (GtkCheckButton *)
04287     gtk_check_button_new_with_mnemonic (gettext ("_Gradient based method"));
04288   g_signal_connect (window->check_gradient, "clicked",
      window_update, NULL);
04289   window->grid_gradient = (GtkGrid *) gtk_grid_new ();
04290   window->button_gradient[0] = (GtkRadioButton *)
04291     gtk_radio_button_new_with_mnemonic (NULL, label_gradient[0]);
04292   gtk_grid_attach (window->grid_gradient,
04293                    GTK_WIDGET (window->button_gradient[0]), 0, 0, 1, 1);
04294   g_signal_connect (window->button_gradient[0], "clicked",
      window_update, NULL);
04295   for (i = 0; ++i < NGRADIENTS;)
04296     {
04297       window->button_gradient[i] = (GtkRadioButton *)
```

```
04298         gtk_radio_button_new_with_mnemonic
04299         (gtk_radio_button_get_group (window->button_gradient[0]),
04300          label_gradient[i]);
04301       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_gradient[i]),
04302                                    tip_gradient[i]);
04303       gtk_grid_attach (window->grid_gradient,
04304                        GTK_WIDGET (window->button_gradient[i]), 0, i, 1, 1);
04305       g_signal_connect (window->button_gradient[i], "clicked",
04306                         window_update, NULL);
04307     }
04308   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
04309   window->spin_steps = (GtkSpinButton *)
04310     gtk_spin_button_new_with_range (1., 1.e12, 1.);
04311   window->label_estimates
04312     = (GtkLabel *) gtk_label_new (gettext ("Gradient estimates number"));
04313   window->spin_estimates = (GtkSpinButton *)
04314     gtk_spin_button_new_with_range (1., 1.e3, 1.);
04315   window->label_relaxation
04316     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
04317   window->spin_relaxation = (GtkSpinButton *)
04318     gtk_spin_button_new_with_range (0., 2., 0.001);
04319   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     label_steps),
04320                    0, NGRADIENTS, 1, 1);
04321   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     spin_steps),
04322                    1, NGRADIENTS, 1, 1);
04323   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     label_estimates),
04324                    0, NGRADIENTS + 1, 1, 1);
04325   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     spin_estimates),
04326                    1, NGRADIENTS + 1, 1, 1);
04327   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     label_relaxation),
04328                    0, NGRADIENTS + 2, 1, 1);
04329   gtk_grid_attach (window->grid_gradient, GTK_WIDGET (window->
     spin_relaxation),
04330                    1, NGRADIENTS + 2, 1, 1);
04331
04332   // Creating the array of algorithms
04333   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
04334   window->button_algorithm[0] = (GtkRadioButton *)
04335     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
04336   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
04337                                tip_algorithm[0]);
04338   gtk_grid_attach (window->grid_algorithm,
04339                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
04340   g_signal_connect (window->button_algorithm[0], "clicked",
04341                     window_set_algorithm, NULL);
04342   for (i = 0; ++i < NALGORITHMS;)
04343     {
04344       window->button_algorithm[i] = (GtkRadioButton *)
04345         gtk_radio_button_new_with_mnemonic
04346         (gtk_radio_button_get_group (window->button_algorithm[0]),
04347          label_algorithm[i]);
04348       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
04349                                    tip_algorithm[i]);
04350       gtk_grid_attach (window->grid_algorithm,
04351                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
04352       g_signal_connect (window->button_algorithm[i], "clicked",
04353                         window_set_algorithm, NULL);
04354     }
04355   gtk_grid_attach (window->grid_algorithm,
04356                    GTK_WIDGET (window->label_simulations), 0,
04357                    NALGORITHMS, 1, 1);
04358   gtk_grid_attach (window->grid_algorithm,
04359                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
04360   gtk_grid_attach (window->grid_algorithm,
04361                    GTK_WIDGET (window->label_iterations), 0,
04362                    NALGORITHMS + 1, 1, 1);
04363   gtk_grid_attach (window->grid_algorithm,
04364                    GTK_WIDGET (window->spin_iterations), 1,
04365                    NALGORITHMS + 1, 1, 1);
04366   gtk_grid_attach (window->grid_algorithm,
04367                    GTK_WIDGET (window->label_tolerance), 0,
04368                    NALGORITHMS + 2, 1, 1);
04369   gtk_grid_attach (window->grid_algorithm,
04370                    GTK_WIDGET (window->spin_tolerance), 1,
04371                    NALGORITHMS + 2, 1, 1);
04372   gtk_grid_attach (window->grid_algorithm,
04373                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
04374   gtk_grid_attach (window->grid_algorithm,
04375                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
04376   gtk_grid_attach (window->grid_algorithm,
04377                    GTK_WIDGET (window->label_population), 0,
04378                    NALGORITHMS + 4, 1, 1);
```

```
04379   gtk_grid_attach (window->grid_algorithm,
04380                    GTK_WIDGET (window->spin_population), 1,
04381                    NALGORITHMS + 4, 1, 1);
04382   gtk_grid_attach (window->grid_algorithm,
04383                    GTK_WIDGET (window->label_generations), 0,
04384                    NALGORITHMS + 5, 1, 1);
04385   gtk_grid_attach (window->grid_algorithm,
04386                    GTK_WIDGET (window->spin_generations), 1,
04387                    NALGORITHMS + 5, 1, 1);
04388   gtk_grid_attach (window->grid_algorithm,
04389                    GTK_WIDGET (window->label_mutation), 0,
04390                    NALGORITHMS + 6, 1, 1);
04391   gtk_grid_attach (window->grid_algorithm,
04392                    GTK_WIDGET (window->spin_mutation), 1,
04393                    NALGORITHMS + 6, 1, 1);
04394   gtk_grid_attach (window->grid_algorithm,
04395                    GTK_WIDGET (window->label_reproduction), 0,
04396                    NALGORITHMS + 7, 1, 1);
04397   gtk_grid_attach (window->grid_algorithm,
04398                    GTK_WIDGET (window->spin_reproduction), 1,
04399                    NALGORITHMS + 7, 1, 1);
04400   gtk_grid_attach (window->grid_algorithm,
04401                    GTK_WIDGET (window->label_adaptation), 0,
04402                    NALGORITHMS + 8, 1, 1);
04403   gtk_grid_attach (window->grid_algorithm,
04404                    GTK_WIDGET (window->spin_adaptation), 1,
04405                    NALGORITHMS + 8, 1, 1);
04406   gtk_grid_attach (window->grid_algorithm,
04407                    GTK_WIDGET (window->check_gradient), 0,
04408                    NALGORITHMS + 9, 2, 1);
04409   gtk_grid_attach (window->grid_algorithm,
04410                    GTK_WIDGET (window->grid_gradient), 0,
04411                    NALGORITHMS + 10, 2, 1);
04412   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
04413   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
04414                      GTK_WIDGET (window->grid_algorithm));
04415
04416   // Creating the variable widgets
04417   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
04418   gtk_widget_set_tooltip_text
04419     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
04420   window->id_variable = g_signal_connect
04421     (window->combo_variable, "changed", window_set_variable, NULL);
04422   window->button_add_variable
04423     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04424                                                    GTK_ICON_SIZE_BUTTON);
04425   g_signal_connect
04426     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
04427   gtk_widget_set_tooltip_text
04428     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
04429   window->button_remove_variable
04430     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04431                                                    GTK_ICON_SIZE_BUTTON);
04432   g_signal_connect
04433     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
04434   gtk_widget_set_tooltip_text
04435     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
04436   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
04437   window->entry_variable = (GtkEntry *) gtk_entry_new ();
04438   gtk_widget_set_tooltip_text
04439     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
04440   window->id_variable_label = g_signal_connect
04441     (window->entry_variable, "changed", window_label_variable, NULL);
04442   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
04443   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
04444     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04445   gtk_widget_set_tooltip_text
04446     (GTK_WIDGET (window->spin_min),
04447      gettext ("Minimum initial value of the variable"));
04448   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04449   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
04450   window->scrolled_min
04451     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04452   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
04453                      GTK_WIDGET (viewport));
04454   g_signal_connect (window->spin_min, "value-changed",
04455                     window_rangemin_variable, NULL);
04456   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
04457   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
04458     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04459   gtk_widget_set_tooltip_text
04460     (GTK_WIDGET (window->spin_max),
04461      gettext ("Maximum initial value of the variable"));
04462   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04463   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
```

```
04464   window->scrolled_max
04465     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04466   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
04467                      GTK_WIDGET (viewport));
04468   g_signal_connect (window->spin_max, "value-changed",
04469                     window_rangemax_variable, NULL);
04470   window->check_minabs = (GtkCheckButton *)
04471     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
04472   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
04473   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04474     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04475   gtk_widget_set_tooltip_text
04476     (GTK_WIDGET (window->spin_minabs),
04477      gettext ("Minimum allowed value of the variable"));
04478   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04479   gtk_container_add (GTK_CONTAINER (viewport),
04480                      GTK_WIDGET (window->spin_minabs));
04481   window->scrolled_minabs
04482     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04483   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
04484                      GTK_WIDGET (viewport));
04485   g_signal_connect (window->spin_minabs, "value-changed",
04486                     window_rangeminabs_variable, NULL);
04487   window->check_maxabs = (GtkCheckButton *)
04488     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
04489   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
04490   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
04491     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
04492   gtk_widget_set_tooltip_text
04493     (GTK_WIDGET (window->spin_maxabs),
04494      gettext ("Maximum allowed value of the variable"));
04495   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
04496   gtk_container_add (GTK_CONTAINER (viewport),
04497                      GTK_WIDGET (window->spin_maxabs));
04498   window->scrolled_maxabs
04499     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
04500   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
04501                      GTK_WIDGET (viewport));
04502   g_signal_connect (window->spin_maxabs, "value-changed",
04503                     window_rangemaxabs_variable, NULL);
04504   window->label_precision
04505     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
04506   window->spin_precision = (GtkSpinButton *)
04507     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
04508   gtk_widget_set_tooltip_text
04509     (GTK_WIDGET (window->spin_precision),
04510      gettext ("Number of precision floating point digits\n"
04511               "0 is for integer numbers"));
04512   g_signal_connect (window->spin_precision, "value-changed",
04513                     window_precision_variable, NULL);
04514   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
04515   window->spin_sweeps
04516     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
04517   gtk_widget_set_tooltip_text
04518     (GTK_WIDGET (window->spin_sweeps),
04519      gettext ("Number of steps sweeping the variable"));
04520   g_signal_connect
04521     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
04522   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
04523   window->spin_bits
04524     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
04525   gtk_widget_set_tooltip_text
04526     (GTK_WIDGET (window->spin_bits),
04527      gettext ("Number of bits to encode the variable"));
04528   g_signal_connect
04529     (window->spin_bits, "value-changed", window_update_variable, NULL);
04530   window->grid_variable = (GtkGrid *) gtk_grid_new ();
04531   gtk_grid_attach (window->grid_variable,
04532                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
04533   gtk_grid_attach (window->grid_variable,
04534                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
04535   gtk_grid_attach (window->grid_variable,
04536                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
04537   gtk_grid_attach (window->grid_variable,
04538                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
04539   gtk_grid_attach (window->grid_variable,
04540                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
04541   gtk_grid_attach (window->grid_variable,
04542                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
04543   gtk_grid_attach (window->grid_variable,
04544                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
04545   gtk_grid_attach (window->grid_variable,
04546                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
04547   gtk_grid_attach (window->grid_variable,
04548                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
04549   gtk_grid_attach (window->grid_variable,
04550                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
```

```
04551   gtk_grid_attach (window->grid_variable,
04552                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
04553   gtk_grid_attach (window->grid_variable,
04554                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
04555   gtk_grid_attach (window->grid_variable,
04556                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
04557   gtk_grid_attach (window->grid_variable,
04558                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
04559   gtk_grid_attach (window->grid_variable,
04560                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
04561   gtk_grid_attach (window->grid_variable,
04562                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
04563   gtk_grid_attach (window->grid_variable,
04564                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
04565   gtk_grid_attach (window->grid_variable,
04566                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
04567   gtk_grid_attach (window->grid_variable,
04568                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
04569   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
04570   gtk_container_add (GTK_CONTAINER (window->frame_variable),
04571                      GTK_WIDGET (window->grid_variable));
04572
04573   // Creating the experiment widgets
04574   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
04575   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
04576                                gettext ("Experiment selector"));
04577   window->id_experiment = g_signal_connect
04578     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
04579   window->button_add_experiment
04580     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
04581                                                    GTK_ICON_SIZE_BUTTON);
04582   g_signal_connect
04583     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
04584   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
04585                                gettext ("Add experiment"));
04586   window->button_remove_experiment
04587     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
04588                                                    GTK_ICON_SIZE_BUTTON);
04589   g_signal_connect (window->button_remove_experiment, "clicked",
04590                     window_remove_experiment, NULL);
04591   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
04592                                gettext ("Remove experiment"));
04593   window->label_experiment
04594     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
04595   window->button_experiment = (GtkFileChooserButton *)
04596     gtk_file_chooser_button_new (gettext ("Experimental data file"),
04597                                  GTK_FILE_CHOOSER_ACTION_OPEN);
04598   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
04599                                gettext ("Experimental data file"));
04600   window->id_experiment_name
04601     = g_signal_connect (window->button_experiment, "selection-changed",
04602                         window_name_experiment, NULL);
04603   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
04604   window->spin_weight
04605     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
04606   gtk_widget_set_tooltip_text
04607     (GTK_WIDGET (window->spin_weight),
04608      gettext ("Weight factor to build the objective function"));
04609   g_signal_connect
04610     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
04611   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
04612   gtk_grid_attach (window->grid_experiment,
04613                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
04614   gtk_grid_attach (window->grid_experiment,
04615                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
04616   gtk_grid_attach (window->grid_experiment,
04617                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
04618   gtk_grid_attach (window->grid_experiment,
04619                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
04620   gtk_grid_attach (window->grid_experiment,
04621                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
04622   gtk_grid_attach (window->grid_experiment,
04623                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
04624   gtk_grid_attach (window->grid_experiment,
04625                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
04626   for (i = 0; i < MAX_NINPUTS; ++i)
04627     {
04628       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
04629       window->check_template[i] = (GtkCheckButton *)
04630         gtk_check_button_new_with_label (buffer3);
04631       window->id_template[i]
04632         = g_signal_connect (window->check_template[i], "toggled",
04633                             window_inputs_experiment, NULL);
04634       gtk_grid_attach (window->grid_experiment,
```

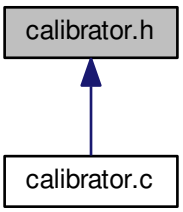
```
04635                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
04636       window->button_template[i] = (GtkFileChooserButton *)
04637         gtk_file_chooser_button_new (gettext ("Input template"),
04638                                      GTK_FILE_CHOOSER_ACTION_OPEN);
04639       gtk_widget_set_tooltip_text
04640         (GTK_WIDGET (window->button_template[i]),
04641          gettext ("Experimental input template file"));
04642       window->id_input[i]
04643         = g_signal_connect_swapped (window->button_template[i],
04644                                     "selection-changed",
04645                                     (void (*)) window_template_experiment,
04646                                     (void *) (size_t) i);
04647       gtk_grid_attach (window->grid_experiment,
04648                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
04649     }
04650   window->frame_experiment
04651     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
04652   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04653                      GTK_WIDGET (window->grid_experiment));
04654
04655   // Creating the grid and attaching the widgets to the grid
04656   window->grid = (GtkGrid *) gtk_grid_new ();
04657   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04658   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04659   gtk_grid_attach (window->grid,
04660                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04661   gtk_grid_attach (window->grid,
04662                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04663   gtk_grid_attach (window->grid,
04664                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04665   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
04666
04667   // Setting the window logo
04668   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04669   gtk_window_set_icon (window->window, window->logo);
04670
04671   // Showing the window
04672   gtk_widget_show_all (GTK_WIDGET (window->window));
04673
04674   // In GTK+ 3.18 the default scrolled size is wrong
04675 #if GTK_MINOR_VERSION >= 18
04676   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04677   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04678   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04679   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04680 #endif
04681
04682   // Reading initial example
04683   input_new ();
04684   buffer2 = g_get_current_dir ();
04685   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04686   g_free (buffer2);
04687   window_read (buffer);
04688   g_free (buffer);
04689 }
04690
04691 #endif
04692
04698 int
04699 cores_number ()
04700 {
04701 #ifdef G_OS_WIN32
04702   SYSTEM_INFO sysinfo;
04703   GetSystemInfo (&sysinfo);
04704   return sysinfo.dwNumberOfProcessors;
04705 #else
04706   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04707 #endif
04708 }
04709
04719 int
04720 main (int argn, char **argc)
04721 {
04722 #if HAVE_GTK
04723   char *buffer;
04724 #endif
04725
04726   // Starting pseudo-random numbers generator
04727   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04728   calibrate->seed = DEFAULT_RANDOM_SEED;
04729
04730   // Allowing spaces in the XML data file
04731   xmlKeepBlanksDefault (0);
04732
04733   // Starting MPI
04734 #if HAVE_MPI
```

```
04735   MPI_Init (&argn, &argc);
04736   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04737   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04738   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04739 #else
04740   ntasks = 1;
04741 #endif
04742
04743 #if HAVE_GTK
04744
04745   // Getting threads number
04746   nthreads_gradient = nthreads = cores_number ();
04747
04748   // Setting local language and international floating point numbers notation
04749   setlocale (LC_ALL, "");
04750   setlocale (LC_NUMERIC, "C");
04751   window->application_directory = g_get_current_dir ();
04752   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
04753   bindtextdomain (PROGRAM_INTERFACE, buffer);
04754   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04755   textdomain (PROGRAM_INTERFACE);
04756
04757   // Initing GTK+
04758   gtk_disable_setlocale ();
04759   gtk_init (&argn, &argc);
04760
04761   // Opening the main window
04762   window_new ();
04763   gtk_main ();
04764
04765   // Freeing memory
04766   input_free ();
04767   g_free (buffer);
04768   gtk_widget_destroy (GTK_WIDGET (window->window));
04769   g_free (window->application_directory);
04770
04771 #else
04772
04773   // Checking syntax
04774   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04775     {
04776       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
04777       return 1;
04778     }
04779
04780   // Getting threads number
04781   if (argn == 2)
04782     nthreads_gradient = nthreads = cores_number ();
04783   else
04784     {
04785       nthreads_gradient = nthreads = atoi (argc[2]);
04786       if (!nthreads)
04787         {
04788           printf ("Bad threads number\n");
04789           return 2;
04790         }
04791     }
04792   printf ("nthreads=%u\n", nthreads);
04793
04794   // Making calibration
04795   if (input_open (argc[argn - 1]))
04796     calibrate_open ();
04797
04798   // Freeing memory
04799   calibrate_free ();
04800
04801 #endif
04802
04803   // Closing MPI
04804 #if HAVE_MPI
04805   MPI_Finalize ();
04806 #endif
04807
04808   // Freeing memory
04809   gsl_rng_free (calibrate->rng);
04810
04811   // Closing
04812   return 0;
04813 }
```

5.3 calibrator.h File Reference

Header file of the calibrator.

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.

5.3.1 Detailed Description

Header file of the calibrator.

Authors

Javier Burguete.

Copyright

Definition in file calibrator.h.

5.3.2 Enumeration Type Documentation

5.3.2.1 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 calibrator.h.

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

5.3.2.2 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 calibrator.h.

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

5.3.3 Function Documentation

5.3.3.1 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 1423 of file calibrator.c.

```
01424 {
01425   unsigned int i, j;
01426   double e;
01427 #if DEBUG
01428   fprintf (stderr, "calibrate_best: start\n");
01429   fprintf (stderr, "calibrate_best: nsaveds=%u nbest=%u\n",
01430            calibrate->nsaveds, calibrate->nbest);
01431 #endif
01432   if (calibrate->nsaveds < calibrate->nbest
01433       || value < calibrate->error_best[calibrate->nsaveds - 1])
01434     {
01435       if (calibrate->nsaveds < calibrate->nbest)
01436         ++calibrate->nsaveds;
01437       calibrate->error_best[calibrate->nsaveds - 1] = value;
01438       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01439       for (i = calibrate->nsaveds; --i;)
01440         {
01441           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01442             {
01443               j = calibrate->simulation_best[i];
01444               e = calibrate->error_best[i];
01445               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01446               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01447               calibrate->simulation_best[i - 1] = j;
01448               calibrate->error_best[i - 1] = e;
01449             }
01450           else
01451             break;
01452         }
01453     }
01454 #if DEBUG
01455   fprintf (stderr, "calibrate_best: end\n");
01456 #endif
01457 }
```

5.3.3.2 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 1736 of file calibrator.c.

```
01737 {
01738 #if DEBUG
01739   fprintf (stderr, "calibrate_best_gradient: start\n");
01740   fprintf (stderr,
01741            "calibrate_best_gradient: simulation=%u value=%.14le best=%.14le\n",
01742            simulation, value, calibrate->error_best[0]);
01743 #endif
01744   if (value < calibrate->error_best[0])
01745     {
01746       calibrate->error_best[0] = value;
01747       calibrate->simulation_best[0] = simulation;
01748 #if DEBUG
01749       fprintf (stderr,
01750                "calibrate_best_gradient: BEST simulation=%u value=%.14le\n",
01751                simulation, value);
01752 #endif
01753     }
01754 #if DEBUG
01755   fprintf (stderr, "calibrate_best_gradient: end\n");
01756 #endif
01757 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

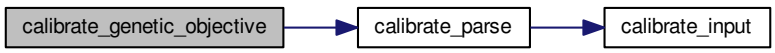
Returns

objective function value.

Definition at line 2028 of file calibrator.c.

```
02029 {
02030   unsigned int j;
02031   double objective;
02032   char buffer[64];
02033 #if DEBUG
02034   fprintf (stderr, "calibrate_genetic_objective: start\n");
02035 #endif
02036   for (j = 0; j < calibrate->nvariables; ++j)
02037     {
02038       calibrate->value[entity->id * calibrate->nvariables + j]
02039         = genetic_get_variable (entity, calibrate->genetic_variable + j);
02040     }
02041   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
02042     objective += calibrate_parse (entity->id, j);
02043   g_mutex_lock (mutex);
02044   for (j = 0; j < calibrate->nvariables; ++j)
02045     {
02046       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
02047       fprintf (calibrate->file_variables, buffer,
02048                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
02049     }
02050   fprintf (calibrate->file_variables, "%.14le\n", objective);
02051   g_mutex_unlock (mutex);
02052 #if DEBUG
02053   fprintf (stderr, "calibrate_genetic_objective: end\n");
02054 #endif
02055   return objective;
02056 }
```

Here is the call graph for this function:



5.3.3.4 void* calibrate_gradient_thread (ParallelData * *data*)

Function to estimate the gradient on a thread.

Parameters

data	Function data.

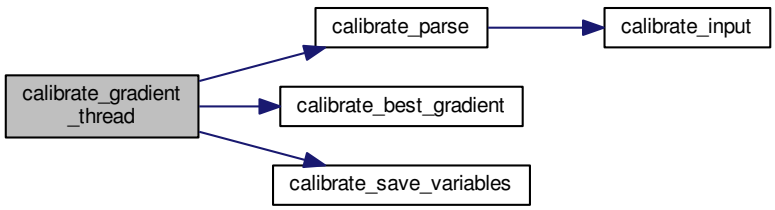
Returns

NULL

Definition at line 1801 of file calibrator.c.

```
01802 {
01803   unsigned int i, j, thread;
01804   double e;
01805 #if DEBUG
01806   fprintf (stderr, "calibrate_gradient_thread: start\n");
01807 #endif
01808   thread = data->thread;
01809 #if DEBUG
01810   fprintf (stderr, "calibrate_gradient_thread: thread=%u start=%u end=%u\n",
01811            thread,
01812            calibrate->thread_gradient[thread],
01813            calibrate->thread_gradient[thread + 1]);
01814 #endif
01815   for (i = calibrate->thread_gradient[thread];
01816        i < calibrate->thread_gradient[thread + 1]; ++i)
01817     {
01818       e = 0.;
01819       for (j = 0; j < calibrate->nexperiments; ++j)
01820         e += calibrate_parse (i, j);
01821       g_mutex_lock (mutex);
01822       calibrate_best_gradient (i, e);
01823       calibrate_save_variables (i, e);
01824       g_mutex_unlock (mutex);
01825 #if DEBUG
01826       fprintf (stderr, "calibrate_gradient_thread: i=%u e=%lg\n", i, e);
01827 #endif
01828     }
01829 #if DEBUG
01830   fprintf (stderr, "calibrate_gradient_thread: end\n");
01831 #endif
01832   g_thread_exit (NULL);
01833   return NULL;
01834 }
```

Here is the call graph for this function:



5.3.3.5 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 1176 of file calibrator.c.

```
01177 {
01178   unsigned int i;
01179   char buffer[32], value[32], *buffer2, *buffer3, *content;
01180   FILE *file;
01181   gsize length;
01182   GRegex *regex;
01183
01184 #if DEBUG
01185   fprintf (stderr, "calibrate_input: start\n");
01186 #endif
```

```
01187
01188   // Checking the file
01189   if (!template)
01190     goto calibrate_input_end;
01191
01192   // Opening template
01193   content = g_mapped_file_get_contents (template);
01194   length = g_mapped_file_get_length (template);
01195 #if DEBUG
01196   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01197            content);
01198 #endif
01199   file = g_fopen (input, "w");
01200
01201   // Parsing template
01202   for (i = 0; i < calibrate->nvariables; ++i)
01203     {
01204 #if DEBUG
01205       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01206 #endif
01207       snprintf (buffer, 32, "@variable%u@", i + 1);
01208       regex = g_regex_new (buffer, 0, 0, NULL);
01209       if (i == 0)
01210         {
01211           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01212                                              calibrate->label[i], 0, NULL);
01213 #if DEBUG
01214           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01215 #endif
01216         }
01217       else
01218         {
01219           length = strlen (buffer3);
01220           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01221                                              calibrate->label[i], 0, NULL);
01222           g_free (buffer3);
01223         }
01224       g_regex_unref (regex);
01225       length = strlen (buffer2);
01226       snprintf (buffer, 32, "@value%u@", i + 1);
01227       regex = g_regex_new (buffer, 0, 0, NULL);
01228       snprintf (value, 32, format[calibrate->precision[i]],
01229                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01230
01231 #if DEBUG
01232       fprintf (stderr, "calibrate_input: value=%s\n", value);
01233 #endif
01234       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01235                                          0, NULL);
01236       g_free (buffer2);
01237       g_regex_unref (regex);
01238     }
01239
01240   // Saving input file
01241   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01242   g_free (buffer3);
01243   fclose (file);
01244
01245 calibrate_input_end:
01246 #if DEBUG
01247   fprintf (stderr, "calibrate_input: end\n");
01248 #endif
01249   return;
01250 }
```

5.3.3.6 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 1541 of file calibrator.c.

```
01543 {
01544   unsigned int i, j, k, s[calibrate->nbest];
```

```
01545   double e[calibrate->nbest];
01546 #if DEBUG
01547   fprintf (stderr, "calibrate_merge: start\n");
01548 #endif
01549   i = j = k = 0;
01550   do
01551     {
01552       if (i == calibrate->nsaveds)
01553         {
01554           s[k] = simulation_best[j];
01555           e[k] = error_best[j];
01556           ++j;
01557           ++k;
01558           if (j == nsaveds)
01559             break;
01560         }
01561       else if (j == nsaveds)
01562         {
01563           s[k] = calibrate->simulation_best[i];
01564           e[k] = calibrate->error_best[i];
01565           ++i;
01566           ++k;
01567           if (i == calibrate->nsaveds)
01568             break;
01569         }
01570       else if (calibrate->error_best[i] > error_best[j])
01571         {
01572           s[k] = simulation_best[j];
01573           e[k] = error_best[j];
01574           ++j;
01575           ++k;
01576         }
01577       else
01578         {
01579           s[k] = calibrate->simulation_best[i];
01580           e[k] = calibrate->error_best[i];
01581           ++i;
01582           ++k;
01583         }
01584     }
01585   while (k < calibrate->nbest);
01586   calibrate->nsaveds = k;
01587   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01588   memcpy (calibrate->error_best, e, k * sizeof (double));
01589 #if DEBUG
01590   fprintf (stderr, "calibrate_merge: end\n");
01591 #endif
01592 }
```

5.3.3.7 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 1263 of file calibrator.c.

```
01264 {
01265   unsigned int i;
01266   double e;
01267   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01268     *buffer3, *buffer4;
01269   FILE *file_result;
01270
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_parse: start\n");
01273   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01274            experiment);
01275 #endif
01276
01277   // Opening input files
```

```
01278   for (i = 0; i < calibrate->ninputs; ++i)
01279     {
01280       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01281 #if DEBUG
01282       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01283 #endif
01284       calibrate_input (simulation, &input[i][0],
01285                        calibrate->file[i][experiment]);
01286     }
01287   for (; i < MAX_NINPUTS; ++i)
01288     strcpy (&input[i][0], "");
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_parse: parsing end\n");
01291 #endif
01292
01293   // Performing the simulation
01294   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01295   buffer2 = g_path_get_dirname (calibrate->simulator);
01296   buffer3 = g_path_get_basename (calibrate->simulator);
01297   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01298   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01299             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01300             input[6], input[7], output);
01301   g_free (buffer4);
01302   g_free (buffer3);
01303   g_free (buffer2);
01304 #if DEBUG
01305   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01306 #endif
01307   system (buffer);
01308
01309   // Checking the objective value function
01310   if (calibrate->evaluator)
01311     {
01312       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01313       buffer2 = g_path_get_dirname (calibrate->evaluator);
01314       buffer3 = g_path_get_basename (calibrate->evaluator);
01315       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01316       snprintf (buffer, 512, "\"%s\" %s %s %s",
01317                 buffer4, output, calibrate->experiment[experiment], result);
01318       g_free (buffer4);
01319       g_free (buffer3);
01320       g_free (buffer2);
01321 #if DEBUG
01322       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01323 #endif
01324       system (buffer);
01325       file_result = g_fopen (result, "r");
01326       e = atof (fgets (buffer, 512, file_result));
01327       fclose (file_result);
01328     }
01329   else
01330     {
01331       strcpy (result, "");
01332       file_result = g_fopen (output, "r");
01333       e = atof (fgets (buffer, 512, file_result));
01334       fclose (file_result);
01335     }
01336
01337   // Removing files
01338 #if !DEBUG
01339   for (i = 0; i < calibrate->ninputs; ++i)
01340     {
01341       if (calibrate->file[i][0])
01342         {
01343           snprintf (buffer, 512, RM " %s", &input[i][0]);
01344           system (buffer);
01345         }
01346     }
01347   snprintf (buffer, 512, RM " %s %s", output, result);
01348   system (buffer);
01349 #endif
01350
01351 #if DEBUG
01352   fprintf (stderr, "calibrate_parse: end\n");
01353 #endif
01354
01355   // Returning the objective function
01356   return e * calibrate->weight[experiment];
01357 }
```

Here is the call graph for this function:

5.3.3.8 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 1395 of file calibrator.c.

```
01396 {
01397   unsigned int i;
01398   char buffer[64];
01399 #if DEBUG
01400   fprintf (stderr, "calibrate_save_variables: start\n");
01401 #endif
01402   for (i = 0; i < calibrate->nvariables; ++i)
01403     {
01404       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01405       fprintf (calibrate->file_variables, buffer,
01406                calibrate->value[simulation * calibrate->
      nvariables + i]);
01407     }
01408   fprintf (calibrate->file_variables, "%.14le\n", error);
01409 #if DEBUG
01410   fprintf (stderr, "calibrate_save_variables: end\n");
01411 #endif
01412 }
```

5.3.3.9 void calibrate_step_gradient (unsigned int *simulation*)

Function to do a step of the gradient based method.

Parameters

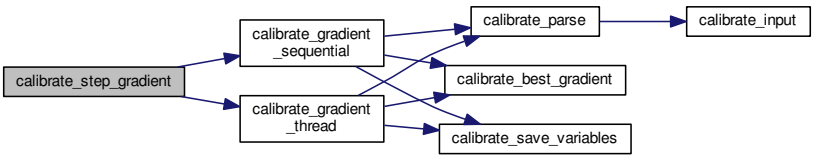
simulation	Simulation number.

Definition at line 1903 of file calibrator.c.

```
01904 {
01905   GThread *thread[nthreads_gradient];
01906   ParallelData data[nthreads_gradient];
01907   unsigned int i, j, k, b;
01908 #if DEBUG
01909   fprintf (stderr, "calibrate_step_gradient: start\n");
01910 #endif
01911   for (i = 0; i < calibrate->nestimates; ++i)
01912     {
01913       k = (simulation + i) * calibrate->nvariables;
01914       b = calibrate->simulation_best[0] * calibrate->
      nvariables;
01915 #if DEBUG
01916       fprintf (stderr, "calibrate_step_gradient: simulation=%u best=%u\n",
01917                simulation + i, calibrate->simulation_best[0]);
01918 #endif
```

```
01919       for (j = 0; j < calibrate->nvariables; ++j, ++k, ++b)
01920         {
01921 #if DEBUG
01922           fprintf (stderr,
01923                    "calibrate_step_gradient: estimate=%u best%u=%.14le\n",
01924                    i, j, calibrate->value[b]);
01925 #endif
01926           calibrate->value[k]
01927             = calibrate->value[b] + calibrate_estimate_gradient (j
      , i);
01928           calibrate->value[k] = fmin (fmax (calibrate->
      value[k],
01929                                             calibrate->rangeminabs[j]),
01930                                       calibrate->rangemaxabs[j]);
01931 #if DEBUG
01932           fprintf (stderr,
01933                    "calibrate_step_gradient: estimate=%u variable%u=%.14le\n",
01934                    i, j, calibrate->value[k]);
01935 #endif
01936         }
01937     }
01938   if (nthreads_gradient == 1)
01939     calibrate_gradient_sequential (simulation);
01940   else
01941     {
01942       for (i = 0; i <= nthreads_gradient; ++i)
01943         {
01944           calibrate->thread_gradient[i]
01945             = simulation + calibrate->nstart_gradient
01946             + i * (calibrate->nend_gradient - calibrate->
      nstart_gradient)
01947             / nthreads_gradient;
01948 #if DEBUG
01949           fprintf (stderr,
01950                    "calibrate_step_gradient: i=%u thread_gradient=%u\n",
01951                    i, calibrate->thread_gradient[i]);
01952 #endif
01953         }
01954       for (i = 0; i < nthreads_gradient; ++i)
01955         {
01956           data[i].thread = i;
01957           thread[i] = g_thread_new
01958             (NULL, (void (*)) calibrate_gradient_thread, &data[i]);
01959         }
01960       for (i = 0; i < nthreads_gradient; ++i)
01961         g_thread_join (thread[i]);
01962     }
01963 #if DEBUG
01964   fprintf (stderr, "calibrate_step_gradient: end\n");
01965 #endif
01966 }
```

Here is the call graph for this function:



5.3.3.10 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

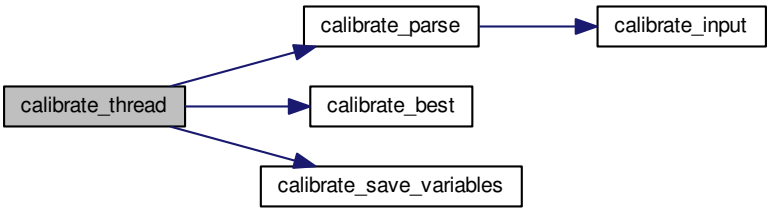
Returns

NULL

Definition at line 1497 of file calibrator.c.

```
01498 {
01499   unsigned int i, j, thread;
01500   double e;
01501 #if DEBUG
01502   fprintf (stderr, "calibrate_thread: start\n");
01503 #endif
01504   thread = data->thread;
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01507            calibrate->thread[thread], calibrate->thread[thread + 1]);
01508 #endif
01509   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01510     {
01511       e = 0.;
01512       for (j = 0; j < calibrate->nexperiments; ++j)
01513         e += calibrate_parse (i, j);
01514       g_mutex_lock (mutex);
01515       calibrate_best (i, e);
01516       calibrate_save_variables (i, e);
01517       g_mutex_unlock (mutex);
01518 #if DEBUG
01519       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01520 #endif
01521     }
01522 #if DEBUG
01523   fprintf (stderr, "calibrate_thread: end\n");
01524 #endif
01525   g_thread_exit (NULL);
01526   return NULL;
01527 }
```

Here is the call graph for this function:



5.3.3.11 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 488 of file calibrator.c.

```
00489 {
00490   char buffer2[64];
00491   char *buffert[MAX_NINPUTS] =
00492     { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
00493   xmlDoc *doc;
00494   xmlNode *node, *child;
00495   xmlChar *buffer;
00496   char *msg;
00497   int error_code;
00498   unsigned int i;
00499
00500 #if DEBUG
00501   fprintf (stderr, "input_open: start\n");
00502 #endif
00503
00504   // Resetting input data
00505   buffer = NULL;
00506   input_new ();
00507
00508   // Parsing the input file
00509 #if DEBUG
00510   fprintf (stderr, "input_open: parsing the input file %s\n", filename);
00511 #endif
00512   doc = xmlParseFile (filename);
00513   if (!doc)
00514     {
00515       msg = gettext ("Unable to parse the input file");
00516       goto exit_on_error;
00517     }
00518
00519   // Getting the root node
00520 #if DEBUG
00521   fprintf (stderr, "input_open: getting the root node\n");
00522 #endif
00523   node = xmlDocGetRootElement (doc);
00524   if (xmlStrcmp (node->name, XML_CALIBRATE))
00525     {
00526       msg = gettext ("Bad root XML node");
00527       goto exit_on_error;
00528     }
00529
00530   // Getting results file names
00531   input->result = (char *) xmlGetProp (node, XML_RESULT);
00532   if (!input->result)
00533     input->result = (char *) xmlStrdup (result_name);
00534   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00535   if (!input->variables)
00536     input->variables = (char *) xmlStrdup (variables_name);
00537
00538   // Opening simulator program name
00539   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00540   if (!input->simulator)
00541     {
00542       msg = gettext ("Bad simulator program");
00543       goto exit_on_error;
00544     }
00545
00546   // Opening evaluator program name
00547   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00548
00549   // Obtaining pseudo-random numbers generator seed
00550   if (!xmlHasProp (node, XML_SEED))
00551     input->seed = DEFAULT_RANDOM_SEED;
00552   else
00553     {
00554       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00555       if (error_code)
00556         {
00557           msg = gettext ("Bad pseudo-random numbers generator seed");
00558           goto exit_on_error;
00559         }
00560     }
00561
00562   // Opening algorithm
00563   buffer = xmlGetProp (node, XML_ALGORITHM);
00564   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00565     {
00566       input->algorithm = ALGORITHM_MONTE_CARLO;
00567
```

```
00568       // Obtaining simulations number
00569       input->nsimulations
00570         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00571       if (error_code)
00572         {
00573           msg = gettext ("Bad simulations number");
00574           goto exit_on_error;
00575         }
00576     }
00577   else if (!xmlStrcmp (buffer, XML_SWEEP))
00578     {
00579       input->algorithm = ALGORITHM_SWEEP;
00580     }
00581   else if (!xmlStrcmp (buffer, XML_GENETIC))
00582     {
00583       input->algorithm = ALGORITHM_GENETIC;
00584
00585       // Obtaining population
00586       if (xmlHasProp (node, XML_NPOPULATION))
00587         {
00588           input->nsimulations
00589             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00590           if (error_code || input->nsimulations < 3)
00591             {
00592               msg = gettext ("Invalid population number");
00593               goto exit_on_error;
00594             }
00595         }
00596       else
00597         {
00598           msg = gettext ("No population number");
00599           goto exit_on_error;
00600         }
00601
00602       // Obtaining generations
00603       if (xmlHasProp (node, XML_NGENERATIONS))
00604         {
00605           input->niterations
00606             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00607           if (error_code || !input->niterations)
00608             {
00609               msg = gettext ("Invalid generations number");
00610               goto exit_on_error;
00611             }
00612         }
00613       else
00614         {
00615           msg = gettext ("No generations number");
00616           goto exit_on_error;
00617         }
00618
00619       // Obtaining mutation probability
00620       if (xmlHasProp (node, XML_MUTATION))
00621         {
00622           input->mutation_ratio
00623             = xml_node_get_float (node, XML_MUTATION, &error_code);
00624           if (error_code || input->mutation_ratio < 0.
00625               || input->mutation_ratio >= 1.)
00626             {
00627               msg = gettext ("Invalid mutation probability");
00628               goto exit_on_error;
00629             }
00630         }
00631       else
00632         {
00633           msg = gettext ("No mutation probability");
00634           goto exit_on_error;
00635         }
00636
00637       // Obtaining reproduction probability
00638       if (xmlHasProp (node, XML_REPRODUCTION))
00639         {
00640           input->reproduction_ratio
00641             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00642           if (error_code || input->reproduction_ratio < 0.
00643               || input->reproduction_ratio >= 1.0)
00644             {
00645               msg = gettext ("Invalid reproduction probability");
00646               goto exit_on_error;
00647             }
00648         }
00649       else
00650         {
00651           msg = gettext ("No reproduction probability");
00652           goto exit_on_error;
00653         }
00654
```

```
00655       // Obtaining adaptation probability
00656       if (xmlHasProp (node, XML_ADAPTATION))
00657         {
00658           input->adaptation_ratio
00659             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00660           if (error_code || input->adaptation_ratio < 0.
00661               || input->adaptation_ratio >= 1.)
00662             {
00663               msg = gettext ("Invalid adaptation probability");
00664               goto exit_on_error;
00665             }
00666         }
00667       else
00668         {
00669           msg = gettext ("No adaptation probability");
00670           goto exit_on_error;
00671         }
00672
00673       // Checking survivals
00674       i = input->mutation_ratio * input->nsimulations;
00675       i += input->reproduction_ratio * input->
      nsimulations;
00676       i += input->adaptation_ratio * input->
      nsimulations;
00677       if (i > input->nsimulations - 2)
00678         {
00679           msg = gettext
00680             ("No enough survival entities to reproduce the population");
00681           goto exit_on_error;
00682         }
00683     }
00684   else
00685     {
00686       msg = gettext ("Unknown algorithm");
00687       goto exit_on_error;
00688     }
00689   xmlFree (buffer);
00690   buffer = NULL;
00691
00692   if (input->algorithm == ALGORITHM_MONTE_CARLO
00693       || input->algorithm == ALGORITHM_SWEEP)
00694     {
00695
00696       // Obtaining iterations number
00697       input->niterations
00698         = xml_node_get_uint (node, XML_NITERATIONS, &error_code);
00699       if (error_code == 1)
00700         input->niterations = 1;
00701       else if (error_code)
00702         {
00703           msg = gettext ("Bad iterations number");
00704           goto exit_on_error;
00705         }
00706
00707       // Obtaining best number
00708       if (xmlHasProp (node, XML_NBEST))
00709         {
00710           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00711           if (error_code || !input->nbest)
00712             {
00713               msg = gettext ("Invalid best number");
00714               goto exit_on_error;
00715             }
00716         }
00717       else
00718         input->nbest = 1;
00719
00720       // Obtaining tolerance
00721       if (xmlHasProp (node, XML_TOLERANCE))
00722         {
00723           input->tolerance
00724             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00725           if (error_code || input->tolerance < 0.)
00726             {
00727               msg = gettext ("Invalid tolerance");
00728               goto exit_on_error;
00729             }
00730         }
00731       else
00732         input->tolerance = 0.;
00733
00734       // Getting gradient method parameters
00735       if (xmlHasProp (node, XML_NSTEPS))
00736         {
00737           input->nsteps = xml_node_get_uint (node,
      XML_NSTEPS, &error_code);
```

```
00738           if (error_code || !input->nsteps)
00739             {
00740               msg = gettext ("Invalid steps number");
00741               goto exit_on_error;
00742             }
00743           buffer = xmlGetProp (node, XML_GRADIENT_METHOD);
00744           if (!xmlStrcmp (buffer, XML_COORDINATES))
00745             input->gradient_method =
      GRADIENT_METHOD_COORDINATES;
00746           else if (!xmlStrcmp (buffer, XML_RANDOM))
00747             {
00748               input->gradient_method =
      GRADIENT_METHOD_RANDOM;
00749               input->nestimates
00750                 = xml_node_get_uint (node, XML_NESTIMATES, &error_code);
00751               if (error_code || !input->nestimates)
00752                 {
00753                   msg = gettext ("Invalid estimates number");
00754                   goto exit_on_error;
00755                 }
00756             }
00757           else
00758             {
00759               msg = gettext ("Unknown method to estimate the gradient");
00760               goto exit_on_error;
00761             }
00762           xmlFree (buffer);
00763           buffer = NULL;
00764           if (xmlHasProp (node, XML_RELAXATION))
00765             {
00766               input->relaxation
00767                 = xml_node_get_float (node, XML_RELAXATION, &error_code);
00768               if (error_code || input->relaxation < 0.
00769                   || input->relaxation > 2.)
00770                 {
00771                   msg = gettext ("Invalid relaxation parameter");
00772                   goto exit_on_error;
00773                 }
00774             }
00775           else
00776             input->relaxation = DEFAULT_RELAXATION;
00777         }
00778       else
00779         input->nsteps = 0;
00780     }
00781
00782   // Reading the experimental data
00783   for (child = node->children; child; child = child->next)
00784     {
00785       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00786         break;
00787 #if DEBUG
00788       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00789 #endif
00790       if (xmlHasProp (child, XML_NAME))
00791         buffer = xmlGetProp (child, XML_NAME);
00792       else
00793         {
00794           snprintf (buffer2, 64, "%s %u: %s",
00795                     gettext ("Experiment"),
00796                     input->nexperiments + 1, gettext ("no data file name"));
00797           msg = buffer2;
00798           goto exit_on_error;
00799         }
00800 #if DEBUG
00801       fprintf (stderr, "input_open: experiment=%s\n", buffer);
00802 #endif
00803       input->weight = g_realloc (input->weight,
00804                                  (1 + input->nexperiments) * sizeof (double));
00805       if (xmlHasProp (child, XML_WEIGHT))
00806         {
00807           input->weight[input->nexperiments]
00808             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00809           if (error_code)
00810             {
00811               snprintf (buffer2, 64, "%s %s: %s",
00812                         gettext ("Experiment"), buffer, gettext ("bad weight"));
00813               msg = buffer2;
00814               goto exit_on_error;
00815             }
00816         }
00817       else
00818         input->weight[input->nexperiments] = 1.;
00819 #if DEBUG
00820       fprintf (stderr, "input_open: weight=%lg\n",
00821                input->weight[input->nexperiments]);
00822 #endif
```

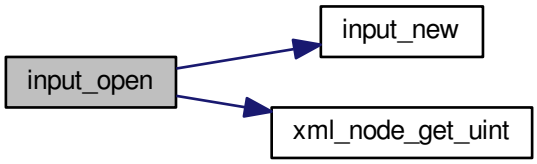
```
00823       if (!input->nexperiments)
00824         input->ninputs = 0;
00825 #if DEBUG
00826       fprintf (stderr, "input_open: template[0]\n");
00827 #endif
00828       if (xmlHasProp (child, XML_TEMPLATE1))
00829         {
00830           input->template[0]
00831             = (char **) g_realloc (input->template[0],
00832                                    (1 + input->nexperiments) * sizeof (char *));
00833           buffert[0] = (char *) xmlGetProp (child, template[0]);
00834 #if DEBUG
00835           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00836                    input->nexperiments,
00837                    buffert[0]);
00838 #endif
00839           if (!input->nexperiments)
00840             ++input->ninputs;
00841 #if DEBUG
00842           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00843 #endif
00844         }
00845       else
00846         {
00847           snprintf (buffer2, 64, "%s %s: %s",
00848                     gettext ("Experiment"), buffer, gettext ("no template"));
00849           msg = buffer2;
00850           goto exit_on_error;
00851         }
00852       for (i = 1; i < MAX_NINPUTS; ++i)
00853         {
00854 #if DEBUG
00855           fprintf (stderr, "input_open: template%u\n", i + 1);
00856 #endif
00857           if (xmlHasProp (child, template[i]))
00858             {
00859               if (input->nexperiments && input->ninputs <= i)
00860                 {
00861                   snprintf (buffer2, 64, "%s %s: %s",
00862                             gettext ("Experiment"),
00863                             buffer, gettext ("bad templates number"));
00864                   msg = buffer2;
00865                   while (i-- > 0)
00866                     xmlFree (buffert[i]);
00867                   goto exit_on_error;
00868                 }
00869               input->template[i] = (char **)
00870                 g_realloc (input->template[i],
00871                            (1 + input->nexperiments) * sizeof (char *));
00872               buffert[i] = (char *) xmlGetProp (child, template[i]);
00873 #if DEBUG
00874               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00875                        input->nexperiments, i + 1,
00876                        input->template[i][input->nexperiments]);
00877 #endif
00878               if (!input->nexperiments)
00879                 ++input->ninputs;
00880 #if DEBUG
00881               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00882 #endif
00883             }
00884           else if (input->nexperiments && input->ninputs >= i)
00885             {
00886               snprintf (buffer2, 64, "%s %s: %s%u",
00887                         gettext ("Experiment"),
00888                         buffer, gettext ("no template"), i + 1);
00889               msg = buffer2;
00890               while (i-- > 0)
00891                 xmlFree (buffert[i]);
00892               goto exit_on_error;
00893             }
00894           else
00895             break;
00896         }
00897       input->experiment
00898         = g_realloc (input->experiment,
00899                      (1 + input->nexperiments) * sizeof (char *));
00900       input->experiment[input->nexperiments] = (char *) buffer;
00901       for (i = 0; i < input->ninputs; ++i)
00902         input->template[i][input->nexperiments] = buffert[i];
00903       ++input->nexperiments;
00904 #if DEBUG
00905       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00906 #endif
00907     }
00908   if (!input->nexperiments)
00909     {
```

```
00910       msg = gettext ("No calibration experiments");
00911       goto exit_on_error;
00912     }
00913   buffer = NULL;
00914
00915   // Reading the variables data
00916   for (; child; child = child->next)
00917     {
00918       if (xmlStrcmp (child->name, XML_VARIABLE))
00919         {
00920           snprintf (buffer2, 64, "%s %u: %s",
00921                     gettext ("Variable"),
00922                     input->nvariables + 1, gettext ("bad XML node"));
00923           msg = buffer2;
00924           goto exit_on_error;
00925         }
00926       if (xmlHasProp (child, XML_NAME))
00927         buffer = xmlGetProp (child, XML_NAME);
00928       else
00929         {
00930           snprintf (buffer2, 64, "%s %u: %s",
00931                     gettext ("Variable"),
00932                     input->nvariables + 1, gettext ("no name"));
00933           msg = buffer2;
00934           goto exit_on_error;
00935         }
00936       if (xmlHasProp (child, XML_MINIMUM))
00937         {
00938           input->rangemin = g_realloc
00939             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00940           input->rangeminabs = g_realloc
00941             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00942           input->rangemin[input->nvariables]
00943             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00944           if (error_code)
00945             {
00946               snprintf (buffer2, 64, "%s %s: %s",
00947                         gettext ("Variable"), buffer, gettext ("bad minimum"));
00948               msg = buffer2;
00949               goto exit_on_error;
00950             }
00951           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00952             {
00953               input->rangeminabs[input->nvariables]
00954                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00955               if (error_code)
00956                 {
00957                   snprintf (buffer2, 64, "%s %s: %s",
00958                             gettext ("Variable"),
00959                             buffer, gettext ("bad absolute minimum"));
00960                   msg = buffer2;
00961                   goto exit_on_error;
00962                 }
00963             }
00964           else
00965             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00966           if (input->rangemin[input->nvariables]
00967               < input->rangeminabs[input->nvariables])
00968             {
00969               snprintf (buffer2, 64, "%s %s: %s",
00970                         gettext ("Variable"),
00971                         buffer, gettext ("minimum range not allowed"));
00972               msg = buffer2;
00973               goto exit_on_error;
00974             }
00975         }
00976       else
00977         {
00978           snprintf (buffer2, 64, "%s %s: %s",
00979                     gettext ("Variable"), buffer, gettext ("no minimum range"));
00980           msg = buffer2;
00981           goto exit_on_error;
00982         }
00983       if (xmlHasProp (child, XML_MAXIMUM))
00984         {
00985           input->rangemax = g_realloc
00986             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00987           input->rangemaxabs = g_realloc
00988             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00989           input->rangemax[input->nvariables]
00990             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00991           if (error_code)
00992             {
00993               snprintf (buffer2, 64, "%s %s: %s",
00994                         gettext ("Variable"), buffer, gettext ("bad maximum"));
00995               msg = buffer2;
```

```
00996               goto exit_on_error;
00997             }
00998           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00999             {
01000               input->rangemaxabs[input->nvariables]
01001                 = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
01002               if (error_code)
01003                 {
01004                   snprintf (buffer2, 64, "%s %s: %s",
01005                             gettext ("Variable"),
01006                             buffer, gettext ("bad absolute maximum"));
01007                   msg = buffer2;
01008                   goto exit_on_error;
01009                 }
01010             }
01011           else
01012             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
01013           if (input->rangemax[input->nvariables]
01014               > input->rangemaxabs[input->nvariables])
01015             {
01016               snprintf (buffer2, 64, "%s %s: %s",
01017                         gettext ("Variable"),
01018                         buffer, gettext ("maximum range not allowed"));
01019               msg = buffer2;
01020               goto exit_on_error;
01021             }
01022         }
01023       else
01024         {
01025           snprintf (buffer2, 64, "%s %s: %s",
01026                     gettext ("Variable"), buffer, gettext ("no maximum range"));
01027           msg = buffer2;
01028           goto exit_on_error;
01029         }
01030       if (input->rangemax[input->nvariables]
01031           < input->rangemin[input->nvariables])
01032         {
01033           snprintf (buffer2, 64, "%s %s: %s",
01034                     gettext ("Variable"), buffer, gettext ("bad range"));
01035           msg = buffer2;
01036           goto exit_on_error;
01037         }
01038       input->precision = g_realloc
01039         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
01040       if (xmlHasProp (child, XML_PRECISION))
01041         {
01042           input->precision[input->nvariables]
01043             = xml_node_get_uint (child, XML_PRECISION, &error_code);
01044           if (error_code || input->precision[input->
      nvariables] >= NPRECISIONS)
01045             {
01046               snprintf (buffer2, 64, "%s %s: %s",
01047                         gettext ("Variable"),
01048                         buffer, gettext ("bad precision"));
01049               msg = buffer2;
01050               goto exit_on_error;
01051             }
01052         }
01053       else
01054         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
01055       if (input->algorithm == ALGORITHM_SWEEP)
01056         {
01057           if (xmlHasProp (child, XML_NSWEEPS))
01058             {
01059               input->nsweeps = (unsigned int *)
01060                 g_realloc (input->nsweeps,
01061                            (1 + input->nvariables) * sizeof (unsigned int));
01062               input->nsweeps[input->nvariables]
01063                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
01064               if (error_code || !input->nsweeps[input->
      nvariables])
01065                 {
01066                   snprintf (buffer2, 64, "%s %s: %s",
01067                             gettext ("Variable"),
01068                             buffer, gettext ("bad sweeps"));
01069                   msg = buffer2;
01070                   goto exit_on_error;
01071                 }
01072             }
01073           else
01074             {
01075               snprintf (buffer2, 64, "%s %s: %s",
01076                         gettext ("Variable"),
01077                         buffer, gettext ("no sweeps number"));
01078               msg = buffer2;
```

```
01079               goto exit_on_error;
01080             }
01081 #if DEBUG
01082           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
01083                    input->nsweeps[input->nvariables],
      input->nsimulations);
01084 #endif
01085         }
01086       if (input->algorithm == ALGORITHM_GENETIC)
01087         {
01088           // Obtaining bits representing each variable
01089           if (xmlHasProp (child, XML_NBITS))
01090             {
01091               input->nbits = (unsigned int *)
01092                 g_realloc (input->nbits,
01093                            (1 + input->nvariables) * sizeof (unsigned int));
01094               i = xml_node_get_uint (child, XML_NBITS, &error_code);
01095               if (error_code || !i)
01096                 {
01097                   snprintf (buffer2, 64, "%s %s: %s",
01098                             gettext ("Variable"),
01099                             buffer, gettext ("invalid bits number"));
01100                   msg = buffer2;
01101                   goto exit_on_error;
01102                 }
01103               input->nbits[input->nvariables] = i;
01104             }
01105           else
01106             {
01107               snprintf (buffer2, 64, "%s %s: %s",
01108                         gettext ("Variable"),
01109                         buffer, gettext ("no bits number"));
01110               msg = buffer2;
01111               goto exit_on_error;
01112             }
01113         }
01114       else if (input->nsteps)
01115         {
01116           input->step = (double *)
01117             g_realloc (input->step, (1 + input->nvariables) * sizeof (double));
01118           input->step[input->nvariables]
01119             = xml_node_get_float (child, XML_STEP, &error_code);
01120           if (error_code || input->step[input->nvariables] < 0.)
01121             {
01122               snprintf (buffer2, 64, "%s %s: %s",
01123                         gettext ("Variable"),
01124                         buffer, gettext ("bad step size"));
01125               msg = buffer2;
01126               goto exit_on_error;
01127             }
01128         }
01129       input->label = g_realloc
01130         (input->label, (1 + input->nvariables) * sizeof (char *));
01131       input->label[input->nvariables] = (char *) buffer;
01132       ++input->nvariables;
01133     }
01134   if (!input->nvariables)
01135     {
01136       msg = gettext ("No calibration variables");
01137       goto exit_on_error;
01138     }
01139   buffer = NULL;
01140
01141   // Getting the working directory
01142   input->directory = g_path_get_dirname (filename);
01143   input->name = g_path_get_basename (filename);
01144
01145   // Closing the XML document
01146   xmlFreeDoc (doc);
01147
01148 #if DEBUG
01149   fprintf (stderr, "input_open: end\n");
01150 #endif
01151   return 1;
01152
01153 exit_on_error:
01154   xmlFree (buffer);
01155   xmlFreeDoc (doc);
01156   show_error (msg);
01157   input_free ();
01158 #if DEBUG
01159   fprintf (stderr, "input_open: end\n");
01160 #endif
01161   return 0;
01162 }
```

Here is the call graph for this function:



5.3.3.12 void show_error (char * *msg*)

Function to show a dialog with an error message.

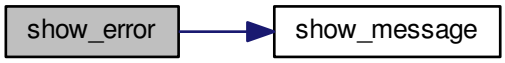
Parameters

msg	Error message.

Definition at line 256 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 226 of file calibrator.c.

```
00227 {
00228 #if HAVE_GTK
00229   GtkMessageDialog *dlg;
00230
00231   // Creating the dialog
```

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

5.3.3.14 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 336 of file calibrator.c.

```
00337 {
00338   double x = 0.;
00339   xmlChar *buffer;
00340   buffer = xmlGetProp (node, prop);
00341   if (!buffer)
00342     *error_code = 1;
00343   else
00344     {
00345       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00346         *error_code = 2;
00347       else
00348         *error_code = 0;
00349       xmlFree (buffer);
00350     }
00351   return x;
00352 }
```

5.3.3.15 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 274 of file calibrator.c.

```
00275 {
00276   int i = 0;
00277   xmlChar *buffer;
00278   buffer = xmlGetProp (node, prop);
00279   if (!buffer)
00280     *error_code = 1;
00281   else
00282     {
00283       if (sscanf ((char *) buffer, "%d", &i) != 1)
00284         *error_code = 2;
00285       else
00286         *error_code = 0;
00287       xmlFree (buffer);
00288     }
00289   return i;
00290 }
```

5.3.3.16 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 305 of file calibrator.c.

```
00306 {
00307   unsigned int i = 0;
00308   xmlChar *buffer;
00309   buffer = xmlGetProp (node, prop);
00310   if (!buffer)
00311     *error_code = 1;
00312   else
00313     {
00314       if (sscanf ((char *) buffer, "%u", &i) != 1)
00315         *error_code = 2;
00316       else
00317         *error_code = 0;
00318       xmlFree (buffer);
00319     }
00320   return i;
00321 }
```

5.3.3.17 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 403 of file calibrator.c.

```
00404 {
00405   xmlChar buffer[64];
00406   snprintf ((char *) buffer, 64, "%.14lg", value);
00407   xmlSetProp (node, prop, buffer);
00408 }
```

5.3.3.18 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 365 of file calibrator.c.

```
00366 {
00367   xmlChar buffer[64];
00368   snprintf ((char *) buffer, 64, "%d", value);
00369   xmlSetProp (node, prop, buffer);
00370 }
```

5.3.3.19 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 384 of file calibrator.c.

```
00385 {
00386   xmlChar buffer[64];
00387   snprintf ((char *) buffer, 64, "%u", value);
00388   xmlSetProp (node, prop, buffer);
00389 }
```

5.4 calibrator.h

```
00001 /*
00002 Calibrator: 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
```

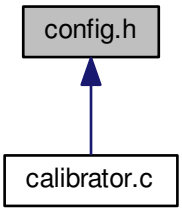
```
00054 enum GradientMethod
00055 {
00056   GRADIENT_METHOD_COORDINATES = 0,
00057   GRADIENT_METHOD_RANDOM = 1,
00058 };
00059
00064 typedef struct
00065 {
00066   char **template[MAX_NINPUTS];
00067   char **experiment;
00068   char **label;
00069   char *result;
00070   char *variables;
00071   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
```

5.5 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 "calibrator"

  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.

5.5.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file config.h.

5.6 config.h

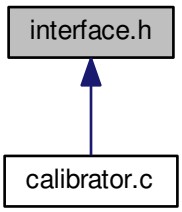
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary 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 "calibrator"
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
```

5.7 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.

5.7.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.7.2 Function Documentation

5.7.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 4699 of file calibrator.c.

```
04700 {
04701 #ifdef G_OS_WIN32
04702   SYSTEM_INFO sysinfo;
04703   GetSystemInfo (&sysinfo);
04704   return sysinfo.dwNumberOfProcessors;
04705 #else
04706   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04707 #endif
04708 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

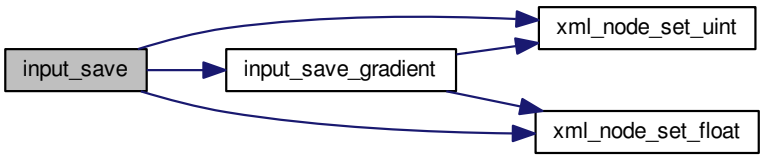
Definition at line 2662 of file calibrator.c.

```
02663 {
02664   unsigned int i, j;
02665   char *buffer;
02666   xmlDoc *doc;
02667   xmlNode *node, *child;
02668   GFile *file, *file2;
02669
02670   // Getting the input file directory
02671   input->name = g_path_get_basename (filename);
02672   input->directory = g_path_get_dirname (filename);
02673   file = g_file_new_for_path (input->directory);
```

```
02674
02675   // Opening the input file
02676   doc = xmlNewDoc ((const xmlChar *) "1.0");
02677
02678   // Setting root XML node
02679   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02680   xmlDocSetRootElement (doc, node);
02681
02682   // Adding properties to the root XML node
02683   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02684     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02685   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02686     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02687   file2 = g_file_new_for_path (input->simulator);
02688   buffer = g_file_get_relative_path (file, file2);
02689   g_object_unref (file2);
02690   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02691   g_free (buffer);
02692   if (input->evaluator)
02693     {
02694       file2 = g_file_new_for_path (input->evaluator);
02695       buffer = g_file_get_relative_path (file, file2);
02696       g_object_unref (file2);
02697       if (xmlStrlen ((xmlChar *) buffer))
02698         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02699       g_free (buffer);
02700     }
02701   if (input->seed != DEFAULT_RANDOM_SEED)
02702     xml_node_set_uint (node, XML_SEED, input->seed);
02703
02704   // Setting the algorithm
02705   buffer = (char *) g_malloc (64);
02706   switch (input->algorithm)
02707     {
02708     case ALGORITHM_MONTE_CARLO:
02709       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02710       snprintf (buffer, 64, "%u", input->nsimulations);
02711       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02712       snprintf (buffer, 64, "%u", input->niterations);
02713       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02714       snprintf (buffer, 64, "%.3lg", input->tolerance);
02715       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02716       snprintf (buffer, 64, "%u", input->nbest);
02717       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02718       input_save_gradient (node);
02719       break;
02720     case ALGORITHM_SWEEP:
02721       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02722       snprintf (buffer, 64, "%u", input->niterations);
02723       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02724       snprintf (buffer, 64, "%.3lg", input->tolerance);
02725       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02726       snprintf (buffer, 64, "%u", input->nbest);
02727       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02728       input_save_gradient (node);
02729       break;
02730     default:
02731       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02732       snprintf (buffer, 64, "%u", input->nsimulations);
02733       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02734       snprintf (buffer, 64, "%u", input->niterations);
02735       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02736       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02737       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02738       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02739       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02740       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02741       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02742       break;
02743     }
02744   g_free (buffer);
02745
02746   // Setting the experimental data
02747   for (i = 0; i < input->nexperiments; ++i)
02748     {
02749       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02750       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02751       if (input->weight[i] != 1.)
02752         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02753       for (j = 0; j < input->ninputs; ++j)
02754         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02755     }
02756
02757   // Setting the variables data
```

```
02758   for (i = 0; i < input->nvariables; ++i)
02759     {
02760       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02761       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02762       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02763       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02764         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02765       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02766       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02767         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02768       if (input->precision[i] != DEFAULT_PRECISION)
02769         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02770       if (input->algorithm == ALGORITHM_SWEEP)
02771         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02772       else if (input->algorithm == ALGORITHM_GENETIC)
02773         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02774     }
02775
02776   // Saving the XML file
02777   xmlSaveFormatFile (filename, doc, 1);
02778
02779   // Freeing memory
02780   xmlFreeDoc (doc);
02781 }
```

Here is the call graph for this function:



5.7.2.3 int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 2879 of file calibrator.c.

```
02880 {
02881   unsigned int i;
02882   for (i = 0; i < NALGORITHMS; ++i)
02883     if (gtk_toggle_button_get_active
02884         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02885       break;
02886   return i;
02887 }
```

5.7.2.4 int window_get_gradient ( )

Function to get the gradient base method number.

Returns

Gradient base method number.

Definition at line 2895 of file calibrator.c.

```
02896 {
02897   unsigned int i;
02898   for (i = 0; i < NGRADIENTS; ++i)
02899     if (gtk_toggle_button_get_active
02900         (GTK_TOGGLE_BUTTON (window->button_gradient[i])))
02901       break;
02902   return i;
02903 }
```

5.7.2.5 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

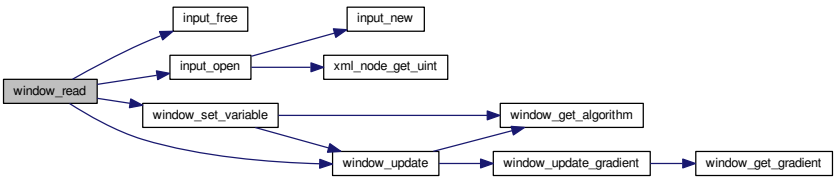
1 on succes, 0 on error.

Definition at line 3900 of file calibrator.c.

```
03901 {
03902   unsigned int i;
03903   char *buffer;
03904 #if DEBUG
03905   fprintf (stderr, "window_read: start\n");
03906 #endif
03907
03908   // Reading new input file
03909   input_free ();
03910   if (!input_open (filename))
03911     return 0;
03912
03913   // Setting GTK+ widgets data
03914   gtk_entry_set_text (window->entry_result, input->result);
03915   gtk_entry_set_text (window->entry_variables, input->
      variables);
03916   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03917   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03918                                  (window->button_simulator), buffer);
03919   g_free (buffer);
03920   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03921                                 (size_t) input->evaluator);
03922   if (input->evaluator)
03923     {
03924       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03925       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03926                                      (window->button_evaluator), buffer);
03927       g_free (buffer);
03928     }
03929   gtk_toggle_button_set_active
03930     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03931   switch (input->algorithm)
03932     {
03933     case ALGORITHM_MONTE_CARLO:
03934       gtk_spin_button_set_value (window->spin_simulations,
03935                                  (gdouble) input->nsimulations);
03936     case ALGORITHM_SWEEP:
03937       gtk_spin_button_set_value (window->spin_iterations,
03938                                  (gdouble) input->niterations);
03939       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03940       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03941       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
```

```
    check_gradient),
03942                                 input->nsteps);
03943       if (input->nsteps)
03944         {
03945           gtk_toggle_button_set_active
03946             (GTK_TOGGLE_BUTTON (window->button_gradient
03947                                 [input->gradient_method]), TRUE);
03948           gtk_spin_button_set_value (window->spin_steps,
03949                                      (gdouble) input->nsteps);
03950           gtk_spin_button_set_value (window->spin_relaxation,
03951                                      (gdouble) input->relaxation);
03952           switch (input->gradient_method)
03953             {
03954             case GRADIENT_METHOD_RANDOM:
03955               gtk_spin_button_set_value (window->spin_estimates,
03956                                          (gdouble) input->nestimates);
03957             }
03958         }
03959       break;
03960     default:
03961       gtk_spin_button_set_value (window->spin_population,
03962                                  (gdouble) input->nsimulations);
03963       gtk_spin_button_set_value (window->spin_generations,
03964                                  (gdouble) input->niterations);
03965       gtk_spin_button_set_value (window->spin_mutation, input->
    mutation_ratio);
03966       gtk_spin_button_set_value (window->spin_reproduction,
03967                                  input->reproduction_ratio);
03968       gtk_spin_button_set_value (window->spin_adaptation,
03969                                  input->adaptation_ratio);
03970     }
03971   g_signal_handler_block (window->combo_experiment, window->
    id_experiment);
03972   g_signal_handler_block (window->button_experiment,
03973                           window->id_experiment_name);
03974   gtk_combo_box_text_remove_all (window->combo_experiment);
03975   for (i = 0; i < input->nexperiments; ++i)
03976     gtk_combo_box_text_append_text (window->combo_experiment,
03977                                     input->experiment[i]);
03978   g_signal_handler_unblock
03979     (window->button_experiment, window->
    id_experiment_name);
03980   g_signal_handler_unblock (window->combo_experiment,
    window->id_experiment);
03981   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03982   g_signal_handler_block (window->combo_variable, window->
    id_variable);
03983   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
03984   gtk_combo_box_text_remove_all (window->combo_variable);
03985   for (i = 0; i < input->nvariables; ++i)
03986     gtk_combo_box_text_append_text (window->combo_variable,
    input->label[i]);
03987   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
03988   g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
03989   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03990   window_set_variable ();
03991   window_update ();
03992
03993 #if DEBUG
03994   fprintf (stderr, "window_read: end\n");
03995 #endif
03996   return 1;
03997 }
```

Here is the call graph for this function:



5.7.2.6 int window_save ( )

Function to save the input file.

Returns

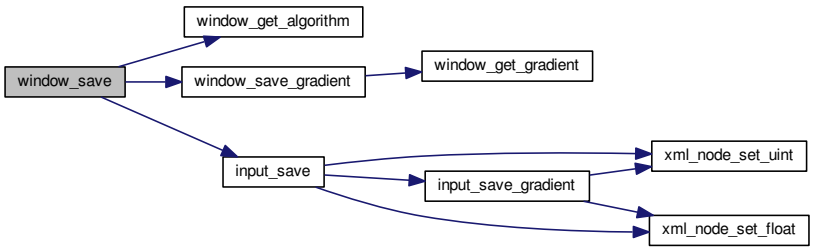
1 on OK, 0 on Cancel.

Definition at line 2943 of file calibrator.c.

```
02944 {
02945   char *buffer;
02946   GtkFileChooserDialog *dlg;
02947
02948 #if DEBUG
02949   fprintf (stderr, "window_save: start\n");
02950 #endif
02951
02952   // Opening the saving dialog
02953   dlg = (GtkFileChooserDialog *)
02954     gtk_file_chooser_dialog_new (gettext ("Save file"),
02955                                  window->window,
02956                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02957                                  gettext ("_Cancel"),
02958                                  GTK_RESPONSE_CANCEL,
02959                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02960   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02961   buffer = g_build_filename (input->directory, input->name, NULL);
02962   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02963   g_free (buffer);
02964
02965   // If OK response then saving
02966   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02967     {
02968
02969       // Adding properties to the root XML node
02970       input->simulator = gtk_file_chooser_get_filename
02971         (GTK_FILE_CHOOSER (window->button_simulator));
02972       if (gtk_toggle_button_get_active
02973           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02974         input->evaluator = gtk_file_chooser_get_filename
02975           (GTK_FILE_CHOOSER (window->button_evaluator));
02976       else
02977         input->evaluator = NULL;
02978       input->result
02979         = (char *) xmlStrdup ((const xmlChar *)
02980                               gtk_entry_get_text (window->entry_result));
02981       input->variables
02982         = (char *) xmlStrdup ((const xmlChar *)
02983                               gtk_entry_get_text (window->entry_variables));
02984
02985       // Setting the algorithm
02986       switch (window_get_algorithm ())
02987         {
02988         case ALGORITHM_MONTE_CARLO:
02989           input->algorithm = ALGORITHM_MONTE_CARLO;
02990           input->nsimulations
02991             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02992           input->niterations
02993             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02994           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02995           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02996           window_save_gradient ();
02997           break;
02998         case ALGORITHM_SWEEP:
02999           input->algorithm = ALGORITHM_SWEEP;
03000           input->niterations
03001             = gtk_spin_button_get_value_as_int (window->spin_iterations);
03002           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
03003           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
03004           window_save_gradient ();
03005           break;
03006         default:
03007           input->algorithm = ALGORITHM_GENETIC;
03008           input->nsimulations
03009             = gtk_spin_button_get_value_as_int (window->spin_population);
03010           input->niterations
03011             = gtk_spin_button_get_value_as_int (window->spin_generations);
03012           input->mutation_ratio
```

```
03013             = gtk_spin_button_get_value (window->spin_mutation);
03014           input->reproduction_ratio
03015             = gtk_spin_button_get_value (window->spin_reproduction);
03016           input->adaptation_ratio
03017             = gtk_spin_button_get_value (window->spin_adaptation);
03018           break;
03019         }
03020
03021       // Saving the XML file
03022       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03023       input_save (buffer);
03024
03025       // Closing and freeing memory
03026       g_free (buffer);
03027       gtk_widget_destroy (GTK_WIDGET (dlg));
03028 #if DEBUG
03029       fprintf (stderr, "window_save: end\n");
03030 #endif
03031       return 1;
03032     }
03033
03034   // Closing and freeing memory
03035   gtk_widget_destroy (GTK_WIDGET (dlg));
03036 #if DEBUG
03037   fprintf (stderr, "window_save: end\n");
03038 #endif
03039   return 0;
03040 }
```

Here is the call graph for this function:



5.7.2.7 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 3536 of file calibrator.c.

```
03537 {
03538   unsigned int i, j;
03539   char *buffer;
03540   GFile *file1, *file2;
03541 #if DEBUG
03542   fprintf (stderr, "window_template_experiment: start\n");
03543 #endif
03544   i = (size_t) data;
03545   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
03546   file1
03547     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
03548   file2 = g_file_new_for_path (input->directory);
03549   buffer = g_file_get_relative_path (file2, file1);
03550   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
03551   g_free (buffer);
03552   g_object_unref (file2);
03553   g_object_unref (file1);
03554 #if DEBUG
```

```
03555   fprintf (stderr, "window_template_experiment: end\n");
03556 #endif
03557 }
```

5.8 interface.h

```
00001 /*
00002 Calibrator: 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   unsigned int precision;
00066   unsigned int nsweeps;
00067   unsigned int nbits;
00068 } Variable;
00069
00074 typedef struct
00075 {
00076   GtkDialog *dialog;
00077   GtkGrid *grid;
00078   GtkLabel *label_seed;
00080   GtkSpinButton *spin_seed;
00082   GtkLabel *label_threads;
00083   GtkSpinButton *spin_threads;
00084   GtkLabel *label_gradient;
00085   GtkSpinButton *spin_gradient;
00086 } Options;
00087
00092 typedef struct
00093 {
00094   GtkDialog *dialog;
00095   GtkLabel *label;
00096 } Running;
00097
00102 typedef struct
00103 {
00104   GtkWindow *window;
00105   GtkGrid *grid;
00106   GtkToolbar *bar_buttons;
00107   GtkToolButton *button_open;
```

```
00108   GtkToolButton *button_save;
00109   GtkToolButton *button_run;
00110   GtkToolButton *button_options;
00111   GtkToolButton *button_help;
00112   GtkToolButton *button_about;
00113   GtkToolButton *button_exit;
00114   GtkGrid *grid_files;
00115   GtkLabel *label_simulator;
00116   GtkFileChooserButton *button_simulator;
00118   GtkCheckButton *check_evaluator;
00119   GtkFileChooserButton *button_evaluator;
00121   GtkLabel *label_result;
00122   GtkEntry *entry_result;
00123   GtkLabel *label_variables;
00124   GtkEntry *entry_variables;
00125   GtkFrame *frame_algorithm;
00126   GtkGrid *grid_algorithm;
00127   GtkRadioButton *button_algorithm[NALGORITHMS];
00129   GtkLabel *label_simulations;
00130   GtkSpinButton *spin_simulations;
00132   GtkLabel *label_iterations;
00133   GtkSpinButton *spin_iterations;
00135   GtkLabel *label_tolerance;
00136   GtkSpinButton *spin_tolerance;
00137   GtkLabel *label_bests;
00138   GtkSpinButton *spin_bests;
00139   GtkLabel *label_population;
00140   GtkSpinButton *spin_population;
00142   GtkLabel *label_generations;
00143   GtkSpinButton *spin_generations;
00145   GtkLabel *label_mutation;
00146   GtkSpinButton *spin_mutation;
00147   GtkLabel *label_reproduction;
00148   GtkSpinButton *spin_reproduction;
00150   GtkLabel *label_adaptation;
00151   GtkSpinButton *spin_adaptation;
00153   GtkCheckButton *check_gradient;
00155   GtkGrid *grid_gradient;
00157   GtkRadioButton *button_gradient[NGRADIENTS];
00159   GtkLabel *label_steps;
00160   GtkSpinButton *spin_steps;
00161   GtkLabel *label_estimates;
00162   GtkSpinButton *spin_estimates;
00164   GtkLabel *label_relaxation;
00166   GtkSpinButton *spin_relaxation;
00168   GtkFrame *frame_variable;
00169   GtkGrid *grid_variable;
00170   GtkComboBoxText *combo_variable;
00172   GtkButton *button_add_variable;
00173   GtkButton *button_remove_variable;
00174   GtkLabel *label_variable;
00175   GtkEntry *entry_variable;
00176   GtkLabel *label_min;
00177   GtkSpinButton *spin_min;
00178   GtkScrolledWindow *scrolled_min;
00179   GtkLabel *label_max;
00180   GtkSpinButton *spin_max;
00181   GtkScrolledWindow *scrolled_max;
00182   GtkCheckButton *check_minabs;
00183   GtkSpinButton *spin_minabs;
00184   GtkScrolledWindow *scrolled_minabs;
00185   GtkCheckButton *check_maxabs;
00186   GtkSpinButton *spin_maxabs;
00187   GtkScrolledWindow *scrolled_maxabs;
00188   GtkLabel *label_precision;
00189   GtkSpinButton *spin_precision;
00190   GtkLabel *label_sweeps;
00191   GtkSpinButton *spin_sweeps;
00192   GtkLabel *label_bits;
00193   GtkSpinButton *spin_bits;
00194   GtkFrame *frame_experiment;
00195   GtkGrid *grid_experiment;
00196   GtkComboBoxText *combo_experiment;
00197   GtkButton *button_add_experiment;
00198   GtkButton *button_remove_experiment;
00199   GtkLabel *label_experiment;
00200   GtkFileChooserButton *button_experiment;
00202   GtkLabel *label_weight;
00203   GtkSpinButton *spin_weight;
00204   GtkCheckButton *check_template[MAX_NINPUTS];
00206   GtkFileChooserButton *button_template[MAX_NINPUTS];
00208   GdkPixbuf *logo;
00209   Experiment *experiment;
00210   Variable *variable;
00211   char *application_directory;
00212   gulong id_experiment;
00213   gulong id_experiment_name;
```

```
00214   gulong id_variable;
00215   gulong id_variable_label;
00216   gulong id_template[MAX_NINPUTS];
00218   gulong id_input[MAX_NINPUTS];
00220   unsigned int nexperiments;
00221   unsigned int nvariables;
00222 } Window;
00223
00224 // Public functions
00225 void input_save (char *filename);
00226 void options_new ();
00227 void running_new ();
00228 int window_get_algorithm ();
00229 int window_get_gradient ();
00230 void window_save_gradient ();
00231 int window_save ();
00232 void window_run ();
00233 void window_help ();
00234 void window_update_gradient ();
00235 void window_update ();
00236 void window_set_algorithm ();
00237 void window_set_experiment ();
00238 void window_remove_experiment ();
00239 void window_add_experiment ();
00240 void window_name_experiment ();
00241 void window_weight_experiment ();
00242 void window_inputs_experiment ();
00243 void window_template_experiment (void *data);
00244 void window_set_variable ();
00245 void window_remove_variable ();
00246 void window_add_variable ();
00247 void window_label_variable ();
00248 void window_precision_variable ();
00249 void window_rangemin_variable ();
00250 void window_rangemax_variable ();
00251 void window_rangeminabs_variable ();
00252 void window_rangemaxabs_variable ();
00253 void window_update_variable ();
00254 int window_read (char *filename);
00255 void window_open ();
00256 void window_new ();
00257 int cores_number ();
00258
00259 #endif
```

Index