Calibrator

1.0.1

Generated by Doxygen 1.8.8

Tue Nov 3 2015 12:23:20

Contents

1 CALIBRATOR **1**

2 Data Structure Index **9**

2.1 Data Structures 9

3 File Index **11**

3.1 File List 11

4 Data Structure Documentation **13**

4.1 Calibrate Struct Reference 13

4.1.1 Detailed Description 15

4.2 Experiment Struct Reference 15

4.2.1 Detailed Description 15

4.3 Input Struct Reference 15

4.3.1 Detailed Description 16

4.4 Options Struct Reference 17

4.4.1 Detailed Description 17

4.5 ParallelData Struct Reference 17

4.5.1 Detailed Description 17

4.6 Running Struct Reference 18

4.6.1 Detailed Description 18

4.7 Variable Struct Reference 18

4.7.1 Detailed Description 19

4.8 Window Struct Reference 19

4.8.1 Detailed Description 22

5 File Documentation **23**

5.1 calibrator.c File Reference 23

5.1.1 Detailed Description 27

5.1.2 Function Documentation 27

5.1.2.1 calibrate_best_sequential 27

5.1.2.2 calibrate_best_thread 28

5.1.2.3 calibrate_genetic_objective 28

5.1.2.4 calibrate_input . . . 29
5.1.2.5 calibrate_merge . . . 30
5.1.2.6 calibrate_parse . . . 31
5.1.2.7 calibrate_save_variables . . . 33
5.1.2.8 calibrate_thread . . . 33
5.1.2.9 cores_number . . . 34
5.1.2.10 input_open . . . 34
5.1.2.11 input_save . . . 41
5.1.2.12 main . . . 43
5.1.2.13 show_error . . . 45
5.1.2.14 show_message . . . 46
5.1.2.15 window_get_algorithm . . . 46
5.1.2.16 window_read . . . 47
5.1.2.17 window_save . . . 49
5.1.2.18 window_template_experiment . . . 51
5.1.2.19 xml_node_get_float . . . 51
5.1.2.20 xml_node_get_int . . . 51
5.1.2.21 xml_node_get_uint . . . 52
5.1.2.22 xml_node_set_float . . . 52
5.1.2.23 xml_node_set_int . . . 53
5.1.2.24 xml_node_set_uint . . . 53
5.1.3 Variable Documentation . . . 53
5.1.3.1 format . . . 53
5.1.3.2 precision . . . 54
5.1.3.3 template . . . 54
5.2 calibrator.c . . . 54
5.3 calibrator.h File Reference . . . 97
5.3.1 Detailed Description . . . 99
5.3.2 Enumeration Type Documentation . . . 99
5.3.2.1 Algorithm . . . 99
5.3.3 Function Documentation . . . 99
5.3.3.1 calibrate_best_sequential . . . 99
5.3.3.2 calibrate_best_thread . . . 100
5.3.3.3 calibrate_genetic_objective . . . 101
5.3.3.4 calibrate_input . . . 101
5.3.3.5 calibrate_merge . . . 103
5.3.3.6 calibrate_parse . . . 104
5.3.3.7 calibrate_save_variables . . . 106
5.3.3.8 calibrate_thread . . . 106
5.3.3.9 input_open . . . 107

5.3.3.10 show_error . . . 114
5.3.3.11 show_message . . . 115
5.3.3.12 xml_node_get_float . . . 115
5.3.3.13 xml_node_get_int . . . 115
5.3.3.14 xml_node_get_uint . . . 116
5.3.3.15 xml_node_set_float . . . 116
5.3.3.16 xml_node_set_int . . . 117
5.3.3.17 xml_node_set_uint . . . 117
5.4 calibrator.h . . . 117
5.5 config.h File Reference . . . 119
5.5.1 Detailed Description . . . 122
5.6 config.h . . . 122
5.7 interface.h File Reference . . . 123
5.7.1 Detailed Description . . . 125
5.7.2 Function Documentation . . . 125
5.7.2.1 cores_number . . . 125
5.7.2.2 input_save . . . 125
5.7.2.3 window_get_algorithm . . . 127
5.7.2.4 window_read . . . 128
5.7.2.5 window_save . . . 129
5.7.2.6 window_template_experiment . . . 131
5.8 interface.h . . . 131

Index **134**

Chapter 1

CALIBRATOR

A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 1.0.0: Stable and recommended version.
- 1.1.33: 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

- 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:

- configure.ac: configure generator.
- Makefile.in: Makefile generator.
- config.h.in: config header generator.
- calibrator.c: main source code.
- calibrator.h: main header code.
- interface.h: interface header code.
- build: script to build all.
- logo.png: logo figure.
- logo2.png: alternative logo figure.
- 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/∗.png: manual figures.
- 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

Fedora Linux 23

FreeBSD 10.2

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/1.1.33
   $ 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

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

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

```
<?xml version="1.0"/>
<calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm_type" nsimulations="simu
    <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="s
    ...
    <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="s
</calibrate>
```

- ∗"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 brutal 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 brutal 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 brutal 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 brutal 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, c-string format and sweeps number to perform are:

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

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

_

```
<?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" format="%.2lf" nsweeps="5"/>
    <variable name="alpha2" minimum="179.30" maximum="179.60" format="%.2lf" nsweeps="5"/>
    <variable name="random" minimum="0.00" maximum="0.20" format="%.2lf" nsweeps="5"/>
    <variable name="boot-time" minimum="0.0" maximum="3.0" format="%.1lf" nsweeps="5"/>
</calibrate>
```

- A template file as *template1.js*:

_

```
{
  "towers" :
  [
    {
      "length"      : 50.11,
      "velocity"    : 0.02738,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.02824,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.03008,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.03753,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    }
  ],
  "cycle-time"     : 71.0,
  "plot-time"     : 1.0,
  "comp-time-step": 0.1,
  "active-percent" : 27.48
}
```

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

—

```
{
  "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:

Calibrate
: Struct to define the calibration data 13

Experiment
: Struct to define experiment data 15

Input
: Struct to define the calibration input file 15

Options
: Struct to define the options dialog 17

ParallelData
: Struct to pass to the GThreads parallelized function 17

Running
: Struct to define the running dialog 18

Variable
: Struct to define variable data 18

Window
: Struct to define the main window 19

Chapter 3

File Index

3.1 File List

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

calibrator.c
Source file of the calibrator 23
calibrator.h
Header file of the calibrator 97
config.h
Configuration header file 119
interface.h
Header file of the interface 123

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

`#include <calibrator.h>`

Data Fields

- char * simulator
  Name of the simulator program.
- char * evaluator
  Name of the program to evaluate the objective function.
- char ** experiment
  Array of experimental data file names.
- char ** template [MAX_NINPUTS]
  Matrix of template names of input files.
- char ** label
  Array of variable names.
- 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 * precision
  Array of variable precisions.
- unsigned int * nsweeps
  Array of sweeps of the sweep algorithm.
- unsigned int nstart
  Beginning simulation number of the task.
- unsigned int nend
  Ending simulation number of the task.
- unsigned int * thread

Array of simulation numbers to calculate on the thread.

- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- unsigned int * simulation_best

  Array of best simulation numbers.
- unsigned long int seed

  Seed of the pseudo-random numbers generator.
- 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 * 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.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- FILE * file_result

  Result file.
- FILE * file_variables

  Variables file.
- gsl_rng * rng

  GSL random number generator.
- GMappedFile ** file [MAX_NINPUTS]

  Matrix of input template files.
- GeneticVariable * genetic_variable

  Array of variables for the genetic algorithm.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 92 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 * simulator
  Name of the simulator program.
- char * evaluator
  Name of the program to evaluate the objective function.
- char ** experiment
  Array of experimental data file names.
- char ** template [MAX_NINPUTS]
  Matrix of template names of input files.
- char ** label
  Array of variable names.

- 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 tolerance
  Algorithm tolerance.
- double mutation_ratio
  Mutation probability.
- double reproduction_ratio
  Reproduction probability.
- double adaptation_ratio
  Adaptation probability.
- 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 * 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.
- 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 54 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_processors

  Processors number GtkLabel.
- GtkSpinButton * spin_processors

  Processors number GtkSpinButton.
- GtkLabel * label_seed

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

  Pseudo-random numbers generator seed 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 147 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 90 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:

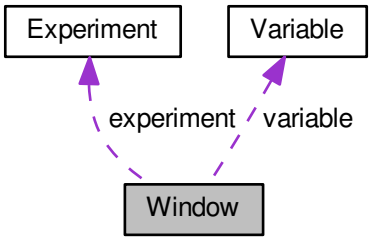
- interface.h

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.

- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- 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.

- 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 100 of file interface.h.

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

- interface.h

Chapter 5

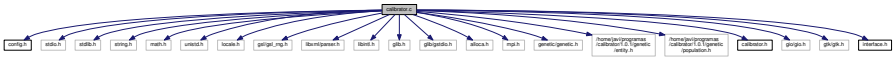
File Documentation

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_thread (unsigned int simulation, double value)

  Function to save the best simulations of a thread.
- void calibrate_best_sequential (unsigned int simulation, double value)

  Function to save the best simulations.
- void * calibrate_thread (ParallelData *data)

  Function to calibrate on a thread.
- void calibrate_sequential ()

  Function to calibrate sequentially.
- 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.
- 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_iterate ()

  Function to iterate the algorithm.
- void calibrate_free ()

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

  Function to open and perform a calibration.
- 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_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.
- int window_get_algorithm ()

  Function to get the algorithm number.
- 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.
- GMutex mutex [1]

  Mutex struct.
- void(* calibrate_step)()

  Pointer to the function to perform a calibration algorithm step.
- Input input [1]

  Input struct to define the input file to calibrator.
- Calibrate calibrate [1]

  Calibration data.
- 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_sequential (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1330 of file calibrator.c.

```
01331 {
01332   unsigned int i, j;
01333   double e;
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_best_sequential: start\n");
01336 #endif
01337   if (calibrate->nsaveds < calibrate->nbest
01338       || value < calibrate->error_best[calibrate->nsaveds - 1])
01339     {
01340       if (calibrate->nsaveds < calibrate->nbest)
01341         ++calibrate->nsaveds;
01342       calibrate->error_best[calibrate->nsaveds - 1] = value;
01343       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01344       for (i = calibrate->nsaveds; --i;)
01345         {
01346           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01347             {
01348               j = calibrate->simulation_best[i];
01349               e = calibrate->error_best[i];
01350               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01351               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01352               calibrate->simulation_best[i - 1] = j;
01353               calibrate->error_best[i - 1] = e;
```

```
01354             }
01355           else
01356             break;
01357         }
01358     }
01359 #if DEBUG
01360   fprintf (stderr, "calibrate_best_sequential: end\n");
01361 #endif
01362 }
```

5.1.2.2 void calibrate_best_thread (unsigned int *simulation*, double *value*)

Function to save the best simulations of a thread.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1285 of file calibrator.c.

```
01286 {
01287   unsigned int i, j;
01288   double e;
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_best_thread: start\n");
01291 #endif
01292   if (calibrate->nsaveds < calibrate->nbest
01293       || value < calibrate->error_best[calibrate->nsaveds - 1])
01294     {
01295       g_mutex_lock (mutex);
01296       if (calibrate->nsaveds < calibrate->nbest)
01297         ++calibrate->nsaveds;
01298       calibrate->error_best[calibrate->nsaveds - 1] = value;
01299       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01300       for (i = calibrate->nsaveds; --i;)
01301         {
01302           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01303             {
01304               j = calibrate->simulation_best[i];
01305               e = calibrate->error_best[i];
01306               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01307               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01308               calibrate->simulation_best[i - 1] = j;
01309               calibrate->error_best[i - 1] = e;
01310             }
01311           else
01312             break;
01313         }
01314       g_mutex_unlock (mutex);
01315     }
01316 #if DEBUG
01317   fprintf (stderr, "calibrate_best_thread: end\n");
01318 #endif
01319 }
```

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

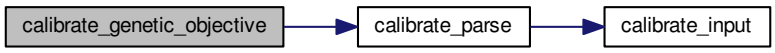
Returns

objective function value.

Definition at line 1639 of file calibrator.c.

```
01640 {
01641   unsigned int j;
01642   double objective;
01643   char buffer[64];
01644 #if DEBUG
01645   fprintf (stderr, "calibrate_genetic_objective: start\n");
01646 #endif
01647   for (j = 0; j < calibrate->nvariables; ++j)
01648     {
01649       calibrate->value[entity->id * calibrate->nvariables + j]
01650         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01651     }
01652   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01653     objective += calibrate_parse (entity->id, j);
01654   g_mutex_lock (mutex);
01655   for (j = 0; j < calibrate->nvariables; ++j)
01656     {
01657       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01658       fprintf (calibrate->file_variables, buffer,
01659                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01660     }
01661   fprintf (calibrate->file_variables, "%.14le\n", objective);
01662   g_mutex_unlock (mutex);
01663 #if DEBUG
01664   fprintf (stderr, "calibrate_genetic_objective: end\n");
01665 #endif
01666   return objective;
01667 }
```

Here is the call graph for this function:



5.1.2.4 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 1034 of file calibrator.c.

```
01035 {
01036   unsigned int i;
01037   char buffer[32], value[32], *buffer2, *buffer3, *content;
01038   FILE *file;
01039   gsize length;
01040   GRegex *regex;
01041
01042 #if DEBUG
01043   fprintf (stderr, "calibrate_input: start\n");
01044 #endif
01045
01046   // Checking the file
01047   if (!template)
01048     goto calibrate_input_end;
01049
01050   // Opening template
01051   content = g_mapped_file_get_contents (template);
01052   length = g_mapped_file_get_length (template);
01053 #if DEBUG
01054   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01055            content);
```

```
01056 #endif
01057   file = g_fopen (input, "w");
01058
01059   // Parsing template
01060   for (i = 0; i < calibrate->nvariables; ++i)
01061     {
01062 #if DEBUG
01063       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01064 #endif
01065       snprintf (buffer, 32, "@variable%u@", i + 1);
01066       regex = g_regex_new (buffer, 0, 0, NULL);
01067       if (i == 0)
01068         {
01069           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01070                                              calibrate->label[i], 0, NULL);
01071 #if DEBUG
01072           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01073 #endif
01074         }
01075       else
01076         {
01077           length = strlen (buffer3);
01078           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01079                                              calibrate->label[i], 0, NULL);
01080           g_free (buffer3);
01081         }
01082       g_regex_unref (regex);
01083       length = strlen (buffer2);
01084       snprintf (buffer, 32, "@value%u@", i + 1);
01085       regex = g_regex_new (buffer, 0, 0, NULL);
01086       snprintf (value, 32, format[calibrate->precision[i]],
01087                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01088
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_input: value=%s\n", value);
01091 #endif
01092       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01093                                          0, NULL);
01094       g_free (buffer2);
01095       g_regex_unref (regex);
01096     }
01097
01098   // Saving input file
01099   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01100   g_free (buffer3);
01101   fclose (file);
01102
01103 calibrate_input_end:
01104 #if DEBUG
01105   fprintf (stderr, "calibrate_input: end\n");
01106 #endif
01107   return;
01108 }
```

5.1.2.5 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 1446 of file calibrator.c.

```
01448 {
01449   unsigned int i, j, k, s[calibrate->nbest];
01450   double e[calibrate->nbest];
01451 #if DEBUG
01452   fprintf (stderr, "calibrate_merge: start\n");
01453 #endif
01454   i = j = k = 0;
01455   do
01456     {
01457       if (i == calibrate->nsaveds)
01458         {
01459           s[k] = simulation_best[j];
01460           e[k] = error_best[j];
```

```
01461           ++j;
01462           ++k;
01463           if (j == nsaveds)
01464             break;
01465         }
01466       else if (j == nsaveds)
01467         {
01468           s[k] = calibrate->simulation_best[i];
01469           e[k] = calibrate->error_best[i];
01470           ++i;
01471           ++k;
01472           if (i == calibrate->nsaveds)
01473             break;
01474         }
01475       else if (calibrate->error_best[i] > error_best[j])
01476         {
01477           s[k] = simulation_best[j];
01478           e[k] = error_best[j];
01479           ++j;
01480           ++k;
01481         }
01482       else
01483         {
01484           s[k] = calibrate->simulation_best[i];
01485           e[k] = calibrate->error_best[i];
01486           ++i;
01487           ++k;
01488         }
01489     }
01490   while (k < calibrate->nbest);
01491   calibrate->nsaveds = k;
01492   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01493   memcpy (calibrate->error_best, e, k * sizeof (double));
01494 #if DEBUG
01495   fprintf (stderr, "calibrate_merge: end\n");
01496 #endif
01497 }
```

5.1.2.6 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 1121 of file calibrator.c.

```
01122 {
01123   unsigned int i;
01124   double e;
01125   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01126     *buffer3, *buffer4;
01127   FILE *file_result;
01128
01129 #if DEBUG
01130   fprintf (stderr, "calibrate_parse: start\n");
01131   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01132            experiment);
01133 #endif
01134
01135   // Opening input files
01136   for (i = 0; i < calibrate->ninputs; ++i)
01137     {
01138       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01139 #if DEBUG
01140       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01141 #endif
01142       calibrate_input (simulation, &input[i][0],
01143                        calibrate->file[i][experiment]);
01144     }
01145   for (; i < MAX_NINPUTS; ++i)
01146     strcpy (&input[i][0], "");
```

```
01147 #if DEBUG
01148   fprintf (stderr, "calibrate_parse: parsing end\n");
01149 #endif
01150
01151   // Performing the simulation
01152   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01153   buffer2 = g_path_get_dirname (calibrate->simulator);
01154   buffer3 = g_path_get_basename (calibrate->simulator);
01155   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01156   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01157             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01158             input[6], input[7], output);
01159   g_free (buffer4);
01160   g_free (buffer3);
01161   g_free (buffer2);
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01164 #endif
01165   system (buffer);
01166
01167   // Checking the objective value function
01168   if (calibrate->evaluator)
01169     {
01170       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01171       buffer2 = g_path_get_dirname (calibrate->evaluator);
01172       buffer3 = g_path_get_basename (calibrate->evaluator);
01173       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01174       snprintf (buffer, 512, "\"%s\" %s %s %s",
01175                 buffer4, output, calibrate->experiment[experiment], result);
01176       g_free (buffer4);
01177       g_free (buffer3);
01178       g_free (buffer2);
01179 #if DEBUG
01180       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01181 #endif
01182       system (buffer);
01183       file_result = g_fopen (result, "r");
01184       e = atof (fgets (buffer, 512, file_result));
01185       fclose (file_result);
01186     }
01187   else
01188     {
01189       strcpy (result, "");
01190       file_result = g_fopen (output, "r");
01191       e = atof (fgets (buffer, 512, file_result));
01192       fclose (file_result);
01193     }
01194
01195   // Removing files
01196 #if !DEBUG
01197   for (i = 0; i < calibrate->ninputs; ++i)
01198     {
01199       if (calibrate->file[i][0])
01200         {
01201           snprintf (buffer, 512, RM " %s", &input[i][0]);
01202           system (buffer);
01203         }
01204     }
01205   snprintf (buffer, 512, RM " %s %s", output, result);
01206   system (buffer);
01207 #endif
01208
01209 #if DEBUG
01210   fprintf (stderr, "calibrate_parse: end\n");
01211 #endif
01212
01213   // Returning the objective function
01214   return e * calibrate->weight[experiment];
01215 }
```

Here is the call graph for this function:

5.1.2.7 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 1257 of file calibrator.c.

```
01258 {
01259   unsigned int i;
01260   char buffer[64];
01261 #if DEBUG
01262   fprintf (stderr, "calibrate_save_variables: start\n");
01263 #endif
01264   for (i = 0; i < calibrate->nvariables; ++i)
01265     {
01266       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01267       fprintf (calibrate->file_variables, buffer,
01268                calibrate->value[simulation * calibrate->
      nvariables + i]);
01269     }
01270   fprintf (calibrate->file_variables, "%.14le\n", error);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_save_variables: end\n");
01273 #endif
01274 }
```

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

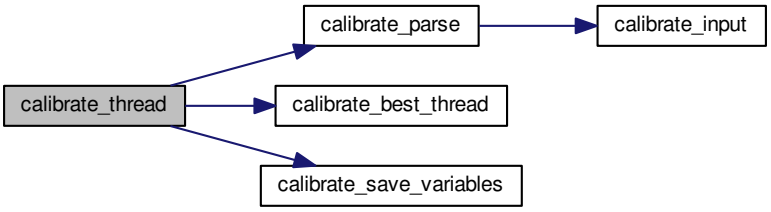
Returns

NULL

Definition at line 1372 of file calibrator.c.

```
01373 {
01374   unsigned int i, j, thread;
01375   double e;
01376 #if DEBUG
01377   fprintf (stderr, "calibrate_thread: start\n");
01378 #endif
01379   thread = data->thread;
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01382            calibrate->thread[thread], calibrate->thread[thread + 1]);
01383 #endif
01384   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01385     {
01386       e = 0.;
01387       for (j = 0; j < calibrate->nexperiments; ++j)
01388         e += calibrate_parse (i, j);
01389       calibrate_best_thread (i, e);
01390       g_mutex_lock (mutex);
01391       calibrate_save_variables (i, e);
01392       g_mutex_unlock (mutex);
01393 #if DEBUG
01394       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01395 #endif
01396     }
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_thread: end\n");
01399 #endif
01400   g_thread_exit (NULL);
01401   return NULL;
01402 }
```

Here is the call graph for this function:



5.1.2.9 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 3906 of file calibrator.c.

```
03907 {
03908 #ifdef G_OS_WIN32
03909   SYSTEM_INFO sysinfo;
03910   GetSystemInfo (&sysinfo);
03911   return sysinfo.dwNumberOfProcessors;
03912 #else
03913   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03914 #endif
03915 }
```

5.1.2.10 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 472 of file calibrator.c.

```
00473 {
00474   char buffer2[64];
00475   xmlDoc *doc;
00476   xmlNode *node, *child;
00477   xmlChar *buffer;
00478   char *msg;
00479   int error_code;
00480   unsigned int i;
00481
00482 #if DEBUG
00483   fprintf (stderr, "input_open: start\n");
00484 #endif
00485
00486   // Resetting input data
```

```
00487   input_new ();
00488
00489   // Parsing the input file
00490   doc = xmlParseFile (filename);
00491   if (!doc)
00492     {
00493       msg = gettext ("Unable to parse the input file");
00494       goto exit_on_error;
00495     }
00496
00497   // Getting the root node
00498   node = xmlDocGetRootElement (doc);
00499   if (xmlStrcmp (node->name, XML_CALIBRATE))
00500     {
00501       msg = gettext ("Bad root XML node");
00502       goto exit_on_error;
00503     }
00504
00505   // Opening simulator program name
00506   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00507   if (!input->simulator)
00508     {
00509       msg = gettext ("Bad simulator program");
00510       goto exit_on_error;
00511     }
00512
00513   // Opening evaluator program name
00514   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00515
00516   // Obtaining pseudo-random numbers generator seed
00517   if (!xmlHasProp (node, XML_SEED))
00518     input->seed = DEFAULT_RANDOM_SEED;
00519   else
00520     {
00521       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00522       if (error_code)
00523         {
00524           msg = gettext ("Bad pseudo-random numbers generator seed");
00525           goto exit_on_error;
00526         }
00527     }
00528
00529   // Opening algorithm
00530   buffer = xmlGetProp (node, XML_ALGORITHM);
00531   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00532     {
00533       input->algorithm = ALGORITHM_MONTE_CARLO;
00534
00535       // Obtaining simulations number
00536       input->nsimulations
00537         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00538       if (error_code)
00539         {
00540           msg = gettext ("Bad simulations number");
00541           goto exit_on_error;
00542         }
00543     }
00544   else if (!xmlStrcmp (buffer, XML_SWEEP))
00545     input->algorithm = ALGORITHM_SWEEP;
00546   else if (!xmlStrcmp (buffer, XML_GENETIC))
00547     {
00548       input->algorithm = ALGORITHM_GENETIC;
00549
00550       // Obtaining population
00551       if (xmlHasProp (node, XML_NPOPULATION))
00552         {
00553           input->nsimulations
00554             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00555           if (error_code || input->nsimulations < 3)
00556             {
00557               msg = gettext ("Invalid population number");
00558               goto exit_on_error;
00559             }
00560         }
00561       else
00562         {
00563           msg = gettext ("No population number");
00564           goto exit_on_error;
00565         }
00566
00567       // Obtaining generations
00568       if (xmlHasProp (node, XML_NGENERATIONS))
00569         {
00570           input->niterations
00571             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00572           if (error_code || !input->niterations)
00573             {
```

```
00574               msg = gettext ("Invalid generations number");
00575               goto exit_on_error;
00576             }
00577         }
00578       else
00579         {
00580           msg = gettext ("No generations number");
00581           goto exit_on_error;
00582         }
00583
00584       // Obtaining mutation probability
00585       if (xmlHasProp (node, XML_MUTATION))
00586         {
00587           input->mutation_ratio
00588             = xml_node_get_float (node, XML_MUTATION, &error_code);
00589           if (error_code || input->mutation_ratio < 0.
00590               || input->mutation_ratio >= 1.)
00591             {
00592               msg = gettext ("Invalid mutation probability");
00593               goto exit_on_error;
00594             }
00595         }
00596       else
00597         {
00598           msg = gettext ("No mutation probability");
00599           goto exit_on_error;
00600         }
00601
00602       // Obtaining reproduction probability
00603       if (xmlHasProp (node, XML_REPRODUCTION))
00604         {
00605           input->reproduction_ratio
00606             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00607           if (error_code || input->reproduction_ratio < 0.
00608               || input->reproduction_ratio >= 1.0)
00609             {
00610               msg = gettext ("Invalid reproduction probability");
00611               goto exit_on_error;
00612             }
00613         }
00614       else
00615         {
00616           msg = gettext ("No reproduction probability");
00617           goto exit_on_error;
00618         }
00619
00620       // Obtaining adaptation probability
00621       if (xmlHasProp (node, XML_ADAPTATION))
00622         {
00623           input->adaptation_ratio
00624             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00625           if (error_code || input->adaptation_ratio < 0.
00626               || input->adaptation_ratio >= 1.)
00627             {
00628               msg = gettext ("Invalid adaptation probability");
00629               goto exit_on_error;
00630             }
00631         }
00632       else
00633         {
00634           msg = gettext ("No adaptation probability");
00635           goto exit_on_error;
00636         }
00637
00638       // Checking survivals
00639       i = input->mutation_ratio * input->nsimulations;
00640       i += input->reproduction_ratio * input->
     nsimulations;
00641       i += input->adaptation_ratio * input->
     nsimulations;
00642       if (i > input->nsimulations - 2)
00643         {
00644           msg = gettext
00645             ("No enough survival entities to reproduce the population");
00646           goto exit_on_error;
00647         }
00648     }
00649   else
00650     {
00651       msg = gettext ("Unknown algorithm");
00652       goto exit_on_error;
00653     }
00654
00655   if (input->algorithm == ALGORITHM_MONTE_CARLO
00656       || input->algorithm == ALGORITHM_SWEEP)
00657     {
00658
```

```
00659       // Obtaining iterations number
00660       input->niterations
00661         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00662       if (error_code == 1)
00663         input->niterations = 1;
00664       else if (error_code)
00665         {
00666           msg = gettext ("Bad iterations number");
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining best number
00671       if (xmlHasProp (node, XML_NBEST))
00672         {
00673           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00674           if (error_code || !input->nbest)
00675             {
00676               msg = gettext ("Invalid best number");
00677               goto exit_on_error;
00678             }
00679         }
00680       else
00681         input->nbest = 1;
00682
00683       // Obtaining tolerance
00684       if (xmlHasProp (node, XML_TOLERANCE))
00685         {
00686           input->tolerance
00687             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00688           if (error_code || input->tolerance < 0.)
00689             {
00690               msg = gettext ("Invalid tolerance");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         input->tolerance = 0.;
00696     }
00697
00698   // Reading the experimental data
00699   for (child = node->children; child; child = child->next)
00700     {
00701       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00702         break;
00703 #if DEBUG
00704       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00705 #endif
00706       if (xmlHasProp (child, XML_NAME))
00707         {
00708           input->experiment
00709             = g_realloc (input->experiment,
00710                          (1 + input->nexperiments) * sizeof (char *));
00711           input->experiment[input->nexperiments]
00712             = (char *) xmlGetProp (child, XML_NAME);
00713         }
00714       else
00715         {
00716           snprintf (buffer2, 64, "%s %u: %s",
00717                     gettext ("Experiment"),
00718                     input->nexperiments + 1, gettext ("no data file name"));
00719           msg = buffer2;
00720           goto exit_on_error;
00721         }
00722 #if DEBUG
00723       fprintf (stderr, "input_open: experiment=%s\n",
00724                input->experiment[input->nexperiments]);
00725 #endif
00726       input->weight = g_realloc (input->weight,
00727                                  (1 + input->nexperiments) * sizeof (double));
00728       if (xmlHasProp (child, XML_WEIGHT))
00729         {
00730           input->weight[input->nexperiments]
00731             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00732           if (error_code)
00733             {
00734               snprintf (buffer2, 64, "%s %u: %s",
00735                         gettext ("Experiment"),
00736                         input->nexperiments + 1, gettext ("bad weight"));
00737               msg = buffer2;
00738               goto exit_on_error;
00739             }
00740         }
00741       else
00742         input->weight[input->nexperiments] = 1.;
00743 #if DEBUG
00744       fprintf (stderr, "input_open: weight=%lg\n",
```

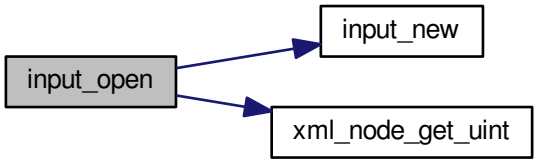
```
00745                 input->weight[input->nexperiments]);
00746 #endif
00747       if (!input->nexperiments)
00748         input->ninputs = 0;
00749 #if DEBUG
00750       fprintf (stderr, "input_open: template[0]\n");
00751 #endif
00752       if (xmlHasProp (child, XML_TEMPLATE1))
00753         {
00754           input->template[0]
00755             = (char **) g_realloc (input->template[0],
00756                                    (1 + input->nexperiments) * sizeof (char *));
00757           input->template[0][input->nexperiments]
00758             = (char *) xmlGetProp (child, template[0]);
00759 #if DEBUG
00760           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00761                    input->nexperiments,
00762                    input->template[0][input->nexperiments]);
00763 #endif
00764           if (!input->nexperiments)
00765             ++input->ninputs;
00766 #if DEBUG
00767           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00768 #endif
00769         }
00770       else
00771         {
00772           snprintf (buffer2, 64, "%s %u: %s",
00773                     gettext ("Experiment"),
00774                     input->nexperiments + 1, gettext ("no template"));
00775           msg = buffer2;
00776           goto exit_on_error;
00777         }
00778       for (i = 1; i < MAX_NINPUTS; ++i)
00779         {
00780 #if DEBUG
00781           fprintf (stderr, "input_open: template%u\n", i + 1);
00782 #endif
00783           if (xmlHasProp (child, template[i]))
00784             {
00785               if (input->nexperiments && input->ninputs <= i)
00786                 {
00787                   snprintf (buffer2, 64, "%s %u: %s",
00788                             gettext ("Experiment"),
00789                             input->nexperiments + 1,
00790                             gettext ("bad templates number"));
00791                   msg = buffer2;
00792                   goto exit_on_error;
00793                 }
00794               input->template[i] = (char **)
00795                 g_realloc (input->template[i],
00796                            (1 + input->nexperiments) * sizeof (char *));
00797               input->template[i][input->nexperiments]
00798                 = (char *) xmlGetProp (child, template[i]);
00799 #if DEBUG
00800               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00801                        input->nexperiments, i + 1,
00802                        input->template[i][input->nexperiments]);
00803 #endif
00804               if (!input->nexperiments)
00805                 ++input->ninputs;
00806 #if DEBUG
00807               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00808 #endif
00809             }
00810           else if (input->nexperiments && input->ninputs >= i)
00811             {
00812               snprintf (buffer2, 64, "%s %u: %s%u",
00813                         gettext ("Experiment"),
00814                         input->nexperiments + 1,
00815                         gettext ("no template"), i + 1);
00816               msg = buffer2;
00817               goto exit_on_error;
00818             }
00819           else
00820             break;
00821         }
00822       ++input->nexperiments;
00823 #if DEBUG
00824       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00825 #endif
00826     }
00827   if (!input->nexperiments)
00828     {
00829       msg = gettext ("No calibration experiments");
00830       goto exit_on_error;
00831     }
```

```
00832
00833   // Reading the variables data
00834   for (; child; child = child->next)
00835     {
00836       if (xmlStrcmp (child->name, XML_VARIABLE))
00837         {
00838           snprintf (buffer2, 64, "%s %u: %s",
00839                     gettext ("Variable"),
00840                     input->nvariables + 1, gettext ("bad XML node"));
00841           msg = buffer2;
00842           goto exit_on_error;
00843         }
00844       if (xmlHasProp (child, XML_NAME))
00845         {
00846           input->label = g_realloc
00847             (input->label, (1 + input->nvariables) * sizeof (char *));
00848           input->label[input->nvariables]
00849             = (char *) xmlGetProp (child, XML_NAME);
00850         }
00851       else
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("no name"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_MINIMUM))
00860         {
00861           input->rangemin = g_realloc
00862             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00863           input->rangeminabs = g_realloc
00864             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00865           input->rangemin[input->nvariables]
00866             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00867           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00868             {
00869               input->rangeminabs[input->nvariables]
00870                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00871             }
00872           else
00873             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00874           if (input->rangemin[input->nvariables]
00875               < input->rangeminabs[input->nvariables])
00876             {
00877               snprintf (buffer2, 64, "%s %u: %s",
00878                         gettext ("Variable"),
00879                         input->nvariables + 1,
00880                         gettext ("minimum range not allowed"));
00881               msg = buffer2;
00882               goto exit_on_error;
00883             }
00884         }
00885       else
00886         {
00887           snprintf (buffer2, 64, "%s %u: %s",
00888                     gettext ("Variable"),
00889                     input->nvariables + 1, gettext ("no minimum range"));
00890           msg = buffer2;
00891           goto exit_on_error;
00892         }
00893       if (xmlHasProp (child, XML_MAXIMUM))
00894         {
00895           input->rangemax = g_realloc
00896             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00897           input->rangemaxabs = g_realloc
00898             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00899           input->rangemax[input->nvariables]
00900             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00901           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00902             input->rangemaxabs[input->nvariables]
00903               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00904           else
00905             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00906           if (input->rangemax[input->nvariables]
00907               > input->rangemaxabs[input->nvariables])
00908             {
00909               snprintf (buffer2, 64, "%s %u: %s",
00910                         gettext ("Variable"),
00911                         input->nvariables + 1,
00912                         gettext ("maximum range not allowed"));
00913               msg = buffer2;
00914               goto exit_on_error;
00915             }
00916         }
```

```
00917       else
00918         {
00919           snprintf (buffer2, 64, "%s %u: %s",
00920                     gettext ("Variable"),
00921                     input->nvariables + 1, gettext ("no maximum range"));
00922           msg = buffer2;
00923           goto exit_on_error;
00924         }
00925       if (input->rangemax[input->nvariables]
00926           < input->rangemin[input->nvariables])
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("bad range"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       input->precision = g_realloc
00935         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00936       if (xmlHasProp (child, XML_PRECISION))
00937         input->precision[input->nvariables]
00938           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00939       else
00940         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00941       if (input->algorithm == ALGORITHM_SWEEP)
00942         {
00943           if (xmlHasProp (child, XML_NSWEEPS))
00944             {
00945               input->nsweeps = (unsigned int *)
00946                 g_realloc (input->nsweeps,
00947                            (1 + input->nvariables) * sizeof (unsigned int));
00948               input->nsweeps[input->nvariables]
00949                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00950             }
00951           else
00952             {
00953               snprintf (buffer2, 64, "%s %u: %s",
00954                         gettext ("Variable"),
00955                         input->nvariables + 1, gettext ("no sweeps number"));
00956               msg = buffer2;
00957               goto exit_on_error;
00958             }
00959 #if DEBUG
00960           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00961                    input->nsweeps[input->nvariables],
      input->nsimulations);
00962 #endif
00963         }
00964       if (input->algorithm == ALGORITHM_GENETIC)
00965         {
00966           // Obtaining bits representing each variable
00967           if (xmlHasProp (child, XML_NBITS))
00968             {
00969               input->nbits = (unsigned int *)
00970                 g_realloc (input->nbits,
00971                            (1 + input->nvariables) * sizeof (unsigned int));
00972               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00973               if (error_code || !i)
00974                 {
00975                   snprintf (buffer2, 64, "%s %u: %s",
00976                             gettext ("Variable"),
00977                             input->nvariables + 1,
00978                             gettext ("invalid bits number"));
00979                   msg = buffer2;
00980                   goto exit_on_error;
00981                 }
00982               input->nbits[input->nvariables] = i;
00983             }
00984           else
00985             {
00986               snprintf (buffer2, 64, "%s %u: %s",
00987                         gettext ("Variable"),
00988                         input->nvariables + 1, gettext ("no bits number"));
00989               msg = buffer2;
00990               goto exit_on_error;
00991             }
00992         }
00993       ++input->nvariables;
00994     }
00995   if (!input->nvariables)
00996     {
00997       msg = gettext ("No calibration variables");
00998       goto exit_on_error;
00999     }
01000
01001   // Getting the working directory
```

```
01002   input->directory = g_path_get_dirname (filename);
01003   input->name = g_path_get_basename (filename);
01004
01005   // Closing the XML document
01006   xmlFreeDoc (doc);
01007
01008 #if DEBUG
01009   fprintf (stderr, "input_open: end\n");
01010 #endif
01011   return 1;
01012
01013 exit_on_error:
01014   show_error (msg);
01015   input_free ();
01016 #if DEBUG
01017   fprintf (stderr, "input_open: end\n");
01018 #endif
01019   return 0;
01020 }
```

Here is the call graph for this function:



5.1.2.11 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

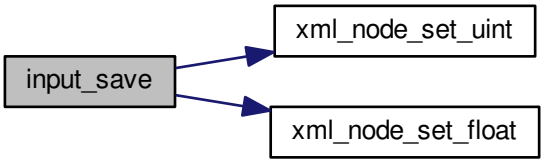
Definition at line 2144 of file calibrator.c.

```
02145 {
02146   unsigned int i, j;
02147   char *buffer;
02148   xmlDoc *doc;
02149   xmlNode *node, *child;
02150   GFile *file, *file2;
02151
02152   // Getting the input file directory
02153   input->name = g_path_get_basename (filename);
02154   input->directory = g_path_get_dirname (filename);
02155   file = g_file_new_for_path (input->directory);
02156
02157   // Opening the input file
02158   doc = xmlNewDoc ((const xmlChar *) "1.0");
02159
02160   // Setting root XML node
02161   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02162   xmlDocSetRootElement (doc, node);
02163
02164   // Adding properties to the root XML node
02165   file2 = g_file_new_for_path (input->simulator);
02166   buffer = g_file_get_relative_path (file, file2);
02167   g_object_unref (file2);
02168   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02169   g_free (buffer);
02170   if (input->evaluator)
02171     {
```

```
02172       file2 = g_file_new_for_path (input->evaluator);
02173       buffer = g_file_get_relative_path (file, file2);
02174       g_object_unref (file2);
02175       if (xmlStrlen ((xmlChar *) buffer))
02176         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02177       g_free (buffer);
02178     }
02179   if (input->seed != DEFAULT_RANDOM_SEED)
02180     xml_node_set_uint (node, XML_SEED, input->seed);
02181
02182   // Setting the algorithm
02183   buffer = (char *) g_malloc (64);
02184   switch (input->algorithm)
02185     {
02186     case ALGORITHM_MONTE_CARLO:
02187       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02188       snprintf (buffer, 64, "%u", input->nsimulations);
02189       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%u", input->niterations);
02191       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%.3lg", input->tolerance);
02193       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02194       snprintf (buffer, 64, "%u", input->nbest);
02195       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02196       break;
02197     case ALGORITHM_SWEEP:
02198       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02199       snprintf (buffer, 64, "%u", input->niterations);
02200       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%.3lg", input->tolerance);
02202       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02203       snprintf (buffer, 64, "%u", input->nbest);
02204       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02205       break;
02206     default:
02207       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02208       snprintf (buffer, 64, "%u", input->nsimulations);
02209       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%u", input->niterations);
02211       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02213       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02215       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02216       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02217       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02218       break;
02219     }
02220   g_free (buffer);
02221
02222   // Setting the experimental data
02223   for (i = 0; i < input->nexperiments; ++i)
02224     {
02225       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02226       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02227       if (input->weight[i] != 1.)
02228         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02229       for (j = 0; j < input->ninputs; ++j)
02230         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02231     }
02232
02233   // Setting the variables data
02234   for (i = 0; i < input->nvariables; ++i)
02235     {
02236       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02237       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02238       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02239       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02240         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02241       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02242       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02243         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02244       if (input->precision[i] != DEFAULT_PRECISION)
02245         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02246       if (input->algorithm == ALGORITHM_SWEEP)
02247         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02248       else if (input->algorithm == ALGORITHM_GENETIC)
02249         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02250     }
```

```
02251
02252   // Saving the XML file
02253   xmlSaveFormatFile (filename, doc, 1);
02254
02255   // Freeing memory
02256   xmlFreeDoc (doc);
02257 }
```

Here is the call graph for this function:



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

Main function.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Returns

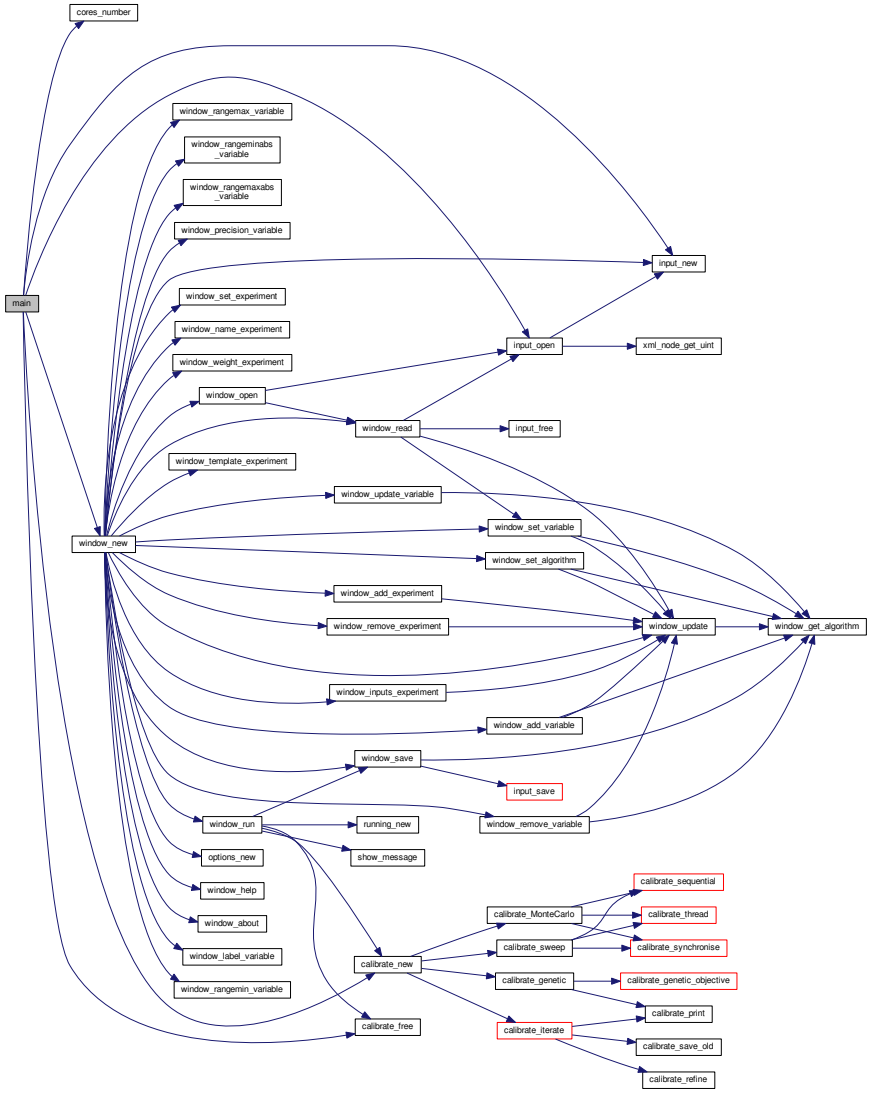
0 on success, >0 on error.

Definition at line 3927 of file calibrator.c.

```
03928 {
03929   // Starting pseudo-random numbers generator
03930   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03931   calibrate->seed = DEFAULT_RANDOM_SEED;
03932
03933   // Allowing spaces in the XML data file
03934   xmlKeepBlanksDefault (0);
03935
03936   // Starting MPI
03937 #if HAVE_MPI
03938   MPI_Init (&argn, &argc);
03939   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03940   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03941   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03942 #else
03943   ntasks = 1;
03944 #endif
03945
03946 #if HAVE_GTK
03947
03948   // Getting threads number
03949   nthreads = cores_number ();
03950
03951   // Setting local language and international floating point numbers notation
03952   setlocale (LC_ALL, "");
03953   setlocale (LC_NUMERIC, "C");
03954   window->application_directory = g_get_current_dir ();
03955   bindtextdomain (PROGRAM_INTERFACE,
03956                   g_build_filename (window->application_directory,
03957                                     LOCALE_DIR, NULL));
03958   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
```

```
03959   textdomain (PROGRAM_INTERFACE);
03960
03961   // Initing GTK+
03962   gtk_disable_setlocale ();
03963   gtk_init (&argn, &argc);
03964
03965   // Opening the main window
03966   window_new ();
03967   gtk_main ();
03968
03969   // Freeing memory
03970   gtk_widget_destroy (GTK_WIDGET (window->window));
03971   g_free (window->application_directory);
03972
03973 #else
03974
03975   // Checking syntax
03976   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03977     {
03978       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03979       return 1;
03980     }
03981
03982   // Getting threads number
03983   if (argn == 2)
03984     nthreads = cores_number ();
03985   else
03986     nthreads = atoi (argc[2]);
03987   printf ("nthreads=%u\n", nthreads);
03988
03989   // Making calibration
03990   input_new ();
03991   if (input_open (argc[argn - 1]))
03992     calibrate_new ();
03993
03994   // Freeing memory
03995   calibrate_free ();
03996
03997 #endif
03998
03999   // Closing MPI
04000 #if HAVE_MPI
04001   MPI_Finalize ();
04002 #endif
04003
04004   // Freeing memory
04005   gsl_rng_free (calibrate->rng);
04006
04007   // Closing
04008   return 0;
04009 }
```

Here is the call graph for this function:



5.1.2.13 void show_error (char * *msg*)

Function to show a dialog with an error message.

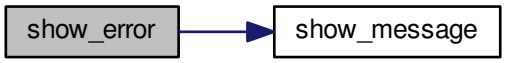
Parameters

msg	Error message.

Definition at line 246 of file calibrator.c.

```
00247 {
00248   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00249 }
```

Here is the call graph for this function:



5.1.2.14 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 216 of file calibrator.c.

```
00217 {
00218 #if HAVE_GTK
00219   GtkMessageDialog *dlg;
00220
00221   // Creating the dialog
00222   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00223     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00224
00225   // Setting the dialog title
00226   gtk_window_set_title (GTK_WINDOW (dlg), title);
00227
00228   // Showing the dialog and waiting response
00229   gtk_dialog_run (GTK_DIALOG (dlg));
00230
00231   // Closing and freeing memory
00232   gtk_widget_destroy (GTK_WIDGET (dlg));
00233
00234 #else
00235   printf ("%s: %s\n", title, msg);
00236 #endif
00237 }
```

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2517 of file calibrator.c.

```
02518 {
02519   unsigned int i;
02520   for (i = 0; i < NALGORITHMS; ++i)
02521     if (gtk_toggle_button_get_active
02522         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02523       break;
02524   return i;
02525 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

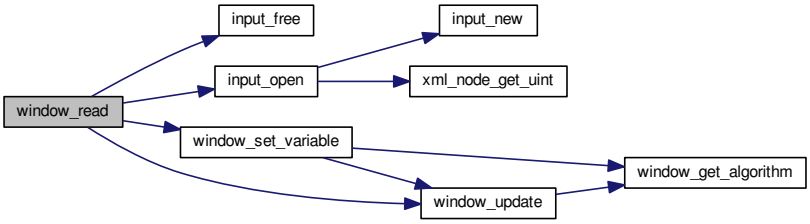
1 on succes, 0 on error.

Definition at line 3268 of file calibrator.c.

```
03269 {
03270   unsigned int i;
03271   char *buffer;
03272 #if DEBUG
03273   fprintf (stderr, "window_read: start\n");
03274 #endif
03275
03276   // Reading new input file
03277   input_free ();
03278   if (!input_open (filename))
03279     return 0;
03280
03281   // Setting GTK+ widgets data
03282   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03283   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03284                                  (window->button_simulator), buffer);
03285   g_free (buffer);
03286   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03287                                 (size_t) input->evaluator);
03288   if (input->evaluator)
03289     {
03290       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03291       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03292                                      (window->button_evaluator), buffer);
03293       g_free (buffer);
03294     }
03295   gtk_toggle_button_set_active
03296     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03297   switch (input->algorithm)
03298     {
03299     case ALGORITHM_MONTE_CARLO:
03300       gtk_spin_button_set_value (window->spin_simulations,
03301                                  (gdouble) input->nsimulations);
03302     case ALGORITHM_SWEEP:
03303       gtk_spin_button_set_value (window->spin_iterations,
03304                                  (gdouble) input->niterations);
03305       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03306       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03307       break;
03308     default:
03309       gtk_spin_button_set_value (window->spin_population,
03310                                  (gdouble) input->nsimulations);
03311       gtk_spin_button_set_value (window->spin_generations,
03312                                  (gdouble) input->niterations);
03313       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03314       gtk_spin_button_set_value (window->spin_reproduction,
03315                                  input->reproduction_ratio);
03316       gtk_spin_button_set_value (window->spin_adaptation,
03317                                  input->adaptation_ratio);
03318     }
03319   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03320   g_signal_handler_block (window->button_experiment,
03321                           window->id_experiment_name);
03322   gtk_combo_box_text_remove_all (window->combo_experiment);
03323   for (i = 0; i < input->nexperiments; ++i)
03324     gtk_combo_box_text_append_text (window->combo_experiment,
03325                                     input->experiment[i]);
03326   g_signal_handler_unblock
03327     (window->button_experiment, window->
      id_experiment_name);
03328   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03329   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03330   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03331   g_signal_handler_block (window->entry_variable, window->
```

```
      id_variable_label);
03332   gtk_combo_box_text_remove_all (window->combo_variable);
03333   for (i = 0; i < input->nvariables; ++i)
03334     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03335   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03336   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03337   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03338   window_set_variable ();
03339   window_update ();
03340
03341 #if DEBUG
03342   fprintf (stderr, "window_read: end\n");
03343 #endif
03344   return 1;
03345 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

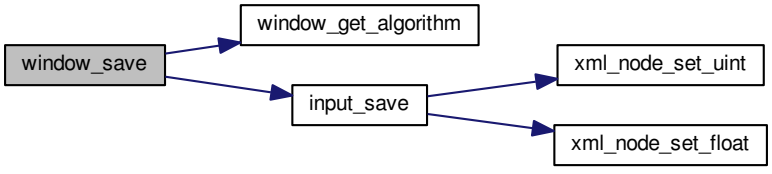
1 on OK, 0 on Cancel.

Definition at line 2332 of file calibrator.c.

```
02333 {
02334   char *buffer;
02335   GtkFileChooserDialog *dlg;
02336
02337 #if DEBUG
02338   fprintf (stderr, "window_save: start\n");
02339 #endif
02340
02341   // Opening the saving dialog
02342   dlg = (GtkFileChooserDialog *)
02343     gtk_file_chooser_dialog_new (gettext ("Save file"),
02344                                  window->window,
02345                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02346                                  gettext ("_Cancel"),
02347                                  GTK_RESPONSE_CANCEL,
02348                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02349   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02350   buffer = g_build_filename (input->directory, input->name, NULL);
02351   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02352   g_free (buffer);
02353
02354   // If OK response then saving
02355   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02356     {
02357
02358       // Adding properties to the root XML node
02359       input->simulator = gtk_file_chooser_get_filename
```

```
02360         (GTK_FILE_CHOOSER (window->button_simulator));
02361       if (gtk_toggle_button_get_active
02362           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02363         input->evaluator = gtk_file_chooser_get_filename
02364           (GTK_FILE_CHOOSER (window->button_evaluator));
02365       else
02366         input->evaluator = NULL;
02367
02368       // Setting the algorithm
02369       switch (window_get_algorithm ())
02370         {
02371         case ALGORITHM_MONTE_CARLO:
02372           input->algorithm = ALGORITHM_MONTE_CARLO;
02373           input->nsimulations
02374             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02375           input->niterations
02376             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02377           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02378           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02379           break;
02380         case ALGORITHM_SWEEP:
02381           input->algorithm = ALGORITHM_SWEEP;
02382           input->niterations
02383             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02384           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02385           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02386           break;
02387         default:
02388           input->algorithm = ALGORITHM_GENETIC;
02389           input->nsimulations
02390             = gtk_spin_button_get_value_as_int (window->spin_population);
02391           input->niterations
02392             = gtk_spin_button_get_value_as_int (window->spin_generations);
02393           input->mutation_ratio
02394             = gtk_spin_button_get_value (window->spin_mutation);
02395           input->reproduction_ratio
02396             = gtk_spin_button_get_value (window->spin_reproduction);
02397           input->adaptation_ratio
02398             = gtk_spin_button_get_value (window->spin_adaptation);
02399           break;
02400         }
02401
02402       // Saving the XML file
02403       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02404       input_save (buffer);
02405
02406       // Closing and freeing memory
02407       g_free (buffer);
02408       gtk_widget_destroy (GTK_WIDGET (dlg));
02409 #if DEBUG
02410       fprintf (stderr, "window_save: end\n");
02411 #endif
02412       return 1;
02413     }
02414
02415   // Closing and freeing memory
02416   gtk_widget_destroy (GTK_WIDGET (dlg));
02417 #if DEBUG
02418   fprintf (stderr, "window_save: end\n");
02419 #endif
02420   return 0;
02421 }
```

Here is the call graph for this function:



5.1.2.18 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 2904 of file calibrator.c.

```
02905 {
02906   unsigned int i, j;
02907   char *buffer;
02908   GFile *file1, *file2;
02909 #if DEBUG
02910   fprintf (stderr, "window_template_experiment: start\n");
02911 #endif
02912   i = (size_t) data;
02913   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02914   file1
02915     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02916   file2 = g_file_new_for_path (input->directory);
02917   buffer = g_file_get_relative_path (file2, file1);
02918   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02919   g_free (buffer);
02920   g_object_unref (file2);
02921   g_object_unref (file1);
02922 #if DEBUG
02923   fprintf (stderr, "window_template_experiment: end\n");
02924 #endif
02925 }
```

5.1.2.19 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 325 of file calibrator.c.

```
00326 {
00327   double x = 0.;
00328   xmlChar *buffer;
00329   buffer = xmlGetProp (node, prop);
00330   if (!buffer)
00331     *error_code = 1;
00332   else
00333     {
00334       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00335         *error_code = 2;
00336       else
00337         *error_code = 0;
00338       xmlFree (buffer);
00339     }
00340   return x;
00341 }
```

5.1.2.20 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 263 of file calibrator.c.

```
00264 {
00265   int i = 0;
00266   xmlChar *buffer;
00267   buffer = xmlGetProp (node, prop);
00268   if (!buffer)
00269     *error_code = 1;
00270   else
00271     {
00272       if (sscanf ((char *) buffer, "%d", &i) != 1)
00273         *error_code = 2;
00274       else
00275         *error_code = 0;
00276       xmlFree (buffer);
00277     }
00278   return i;
00279 }
```

5.1.2.21 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 294 of file calibrator.c.

```
00295 {
00296   unsigned int i = 0;
00297   xmlChar *buffer;
00298   buffer = xmlGetProp (node, prop);
00299   if (!buffer)
00300     *error_code = 1;
00301   else
00302     {
00303       if (sscanf ((char *) buffer, "%u", &i) != 1)
00304         *error_code = 2;
00305       else
00306         *error_code = 0;
00307       xmlFree (buffer);
00308     }
00309   return i;
00310 }
```

5.1.2.22 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 392 of file calibrator.c.

```
00393 {
00394   xmlChar buffer[64];
00395   snprintf ((char *) buffer, 64, "%.14lg", value);
00396   xmlSetProp (node, prop, buffer);
00397 }
```

5.1.2.23 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 354 of file calibrator.c.

```
00355 {
00356   xmlChar buffer[64];
00357   snprintf ((char *) buffer, 64, "%d", value);
00358   xmlSetProp (node, prop, buffer);
00359 }
```

5.1.2.24 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 373 of file calibrator.c.

```
00374 {
00375   xmlChar buffer[64];
00376   snprintf ((char *) buffer, 64, "%u", value);
00377   xmlSetProp (node, prop, buffer);
00378 }
```

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 107 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 112 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 100 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 GMutex mutex[1];
00094 void (*calibrate_step) ();
00096 Input input[1];
00098 Calibrate calibrate[1];
00099
00100 const xmlChar *template[MAX_NINPUTS] = {
00101   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00102   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00103 };
00104
00106
00107 const char *format[NPRECISIONS] = {
00108   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00109   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00110 };
00111
00112 const double precision[NPRECISIONS] = {
00113   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,
00114   1e-13, 1e-14
00115 };
00116
00117 const char *logo[] = {
00118   "32 32 3 1",
00119   "     c None",
00120   ".    c #0000FF",
00121   "+    c #FF0000",
00122   "                                ",
00123   "                                ",
00124   "                                ",
00125   "     .      .      .      .     ",
00126   "     .      .      .      .     ",
00127   "     .      .      .      .     ",
00128   "     .      .      .      .     ",
00129   "     .      .     +++     .     ",
00130   "     .      .    +++++    .     ",
00131   "     .      .    +++++    .     ",
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 const char * logo[] = {
00158 "32 32 3 1",
00159 "     c #FFFFFFFFFFFF",
00160 ".    c #00000000FFFF",
00161 "X    c #FFFF00000000",
00162 "                                ",
00163 "                                ",
00164 "                                ",
00165 "     .      .      .      .     ",
00166 "     .      .      .      .     ",
00167 "     .      .      .      .     ",
00168 "     .      .      .      .     ",
00169 "     .      .     XXX     .     ",
00170 "     .      .    XXXXX    .     ",
00171 "     .      .    XXXXX    .     ",
00172 "     .      .    XXXXX    .     ",
00173 "    XXX     .     XXX    XXX    ",
00174 "   XXXXX    .      .    XXXXX   ",
00175 "   XXXXX    .      .    XXXXX   ",
00176 "   XXXXX    .      .    XXXXX   ",
00177 "    XXX     .      .     XXX    ",
00178 "     .      .      .      .     ",
00179 "     .     XXX     .      .     ",
00180 "     .    XXXXX    .      .     ",
00181 "     .    XXXXX    .      .     ",
00182 "     .    XXXXX    .      .     ",
00183 "     .     XXX     .      .     ",
00184 "     .      .      .      .     ",
00185 "     .      .      .      .     ",
00186 "     .      .      .      .     ",
00187 "     .      .      .      .     ",
00188 "     .      .      .      .     ",
00189 "     .      .      .      .     ",
00190 "     .      .      .      .     ",
00191 "                                ",
00192 "                                ",
00193 "                                "};
00194 */
00195
00196 #if HAVE_GTK
00197 Options options[1];
00199 Running running[1];
00201 Window window[1];
00203 #endif
00204
00215 void
00216 show_message (char *title, char *msg, int type)
00217 {
00218 #if HAVE_GTK
00219   GtkMessageDialog *dlg;
00220
00221   // Creating the dialog
00222   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00223     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00224
00225   // Setting the dialog title
00226   gtk_window_set_title (GTK_WINDOW (dlg), title);
00227
00228   // Showing the dialog and waiting response
00229   gtk_dialog_run (GTK_DIALOG (dlg));
00230
00231   // Closing and freeing memory
00232   gtk_widget_destroy (GTK_WIDGET (dlg));
00233
00234 #else
00235   printf ("%s: %s\n", title, msg);
00236 #endif
00237 }
00238
00245 void
00246 show_error (char *msg)
00247 {
00248   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00249 }
00250
00262 int
00263 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00264 {
```

```
00265   int i = 0;
00266   xmlChar *buffer;
00267   buffer = xmlGetProp (node, prop);
00268   if (!buffer)
00269     *error_code = 1;
00270   else
00271     {
00272       if (sscanf ((char *) buffer, "%d", &i) != 1)
00273         *error_code = 2;
00274       else
00275         *error_code = 0;
00276       xmlFree (buffer);
00277     }
00278   return i;
00279 }
00280
00293 unsigned int
00294 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00295 {
00296   unsigned int i = 0;
00297   xmlChar *buffer;
00298   buffer = xmlGetProp (node, prop);
00299   if (!buffer)
00300     *error_code = 1;
00301   else
00302     {
00303       if (sscanf ((char *) buffer, "%u", &i) != 1)
00304         *error_code = 2;
00305       else
00306         *error_code = 0;
00307       xmlFree (buffer);
00308     }
00309   return i;
00310 }
00311
00324 double
00325 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00326 {
00327   double x = 0.;
00328   xmlChar *buffer;
00329   buffer = xmlGetProp (node, prop);
00330   if (!buffer)
00331     *error_code = 1;
00332   else
00333     {
00334       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00335         *error_code = 2;
00336       else
00337         *error_code = 0;
00338       xmlFree (buffer);
00339     }
00340   return x;
00341 }
00342
00353 void
00354 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00355 {
00356   xmlChar buffer[64];
00357   snprintf ((char *) buffer, 64, "%d", value);
00358   xmlSetProp (node, prop, buffer);
00359 }
00360
00372 void
00373 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00374 {
00375   xmlChar buffer[64];
00376   snprintf ((char *) buffer, 64, "%u", value);
00377   xmlSetProp (node, prop, buffer);
00378 }
00379
00391 void
00392 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00393 {
00394   xmlChar buffer[64];
00395   snprintf ((char *) buffer, 64, "%.14lg", value);
00396   xmlSetProp (node, prop, buffer);
00397 }
00398
00403 void
00404 input_new ()
00405 {
00406   unsigned int i;
00407 #if DEBUG
00408   fprintf (stderr, "input_init: start\n");
00409 #endif
00410   input->nvariables = input->nexperiments = input->ninputs = 0;
00411   input->simulator = input->evaluator = input->directory = input->
```

```
      name = NULL;
00412   input->experiment = input->label = NULL;
00413   input->precision = input->nsweeps = input->nbits = NULL;
00414   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00415     = input->weight = NULL;
00416   for (i = 0; i < MAX_NINPUTS; ++i)
00417     input->template[i] = NULL;
00418 #if DEBUG
00419   fprintf (stderr, "input_init: end\n");
00420 #endif
00421 }
00422
00427 void
00428 input_free ()
00429 {
00430   unsigned int i, j;
00431 #if DEBUG
00432   fprintf (stderr, "input_free: start\n");
00433 #endif
00434   g_free (input->name);
00435   g_free (input->directory);
00436   for (i = 0; i < input->nexperiments; ++i)
00437     {
00438       xmlFree (input->experiment[i]);
00439       for (j = 0; j < input->ninputs; ++j)
00440         xmlFree (input->template[j][i]);
00441     }
00442   g_free (input->experiment);
00443   for (i = 0; i < input->ninputs; ++i)
00444     g_free (input->template[i]);
00445   for (i = 0; i < input->nvariables; ++i)
00446     xmlFree (input->label[i]);
00447   g_free (input->label);
00448   g_free (input->precision);
00449   g_free (input->rangemin);
00450   g_free (input->rangemax);
00451   g_free (input->rangeminabs);
00452   g_free (input->rangemaxabs);
00453   g_free (input->weight);
00454   g_free (input->nsweeps);
00455   g_free (input->nbits);
00456   xmlFree (input->evaluator);
00457   xmlFree (input->simulator);
00458   input->nexperiments = input->ninputs = input->nvariables = 0;
00459 #if DEBUG
00460   fprintf (stderr, "input_free: end\n");
00461 #endif
00462 }
00463
00471 int
00472 input_open (char *filename)
00473 {
00474   char buffer2[64];
00475   xmlDoc *doc;
00476   xmlNode *node, *child;
00477   xmlChar *buffer;
00478   char *msg;
00479   int error_code;
00480   unsigned int i;
00481
00482 #if DEBUG
00483   fprintf (stderr, "input_open: start\n");
00484 #endif
00485
00486   // Resetting input data
00487   input_new ();
00488
00489   // Parsing the input file
00490   doc = xmlParseFile (filename);
00491   if (!doc)
00492     {
00493       msg = gettext ("Unable to parse the input file");
00494       goto exit_on_error;
00495     }
00496
00497   // Getting the root node
00498   node = xmlDocGetRootElement (doc);
00499   if (xmlStrcmp (node->name, XML_CALIBRATE))
00500     {
00501       msg = gettext ("Bad root XML node");
00502       goto exit_on_error;
00503     }
00504
00505   // Opening simulator program name
00506   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00507   if (!input->simulator)
```

```
00508     {
00509       msg = gettext ("Bad simulator program");
00510       goto exit_on_error;
00511     }
00512
00513   // Opening evaluator program name
00514   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00515
00516   // Obtaining pseudo-random numbers generator seed
00517   if (!xmlHasProp (node, XML_SEED))
00518     input->seed = DEFAULT_RANDOM_SEED;
00519   else
00520     {
00521       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00522       if (error_code)
00523         {
00524           msg = gettext ("Bad pseudo-random numbers generator seed");
00525           goto exit_on_error;
00526         }
00527     }
00528
00529   // Opening algorithm
00530   buffer = xmlGetProp (node, XML_ALGORITHM);
00531   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00532     {
00533       input->algorithm = ALGORITHM_MONTE_CARLO;
00534
00535       // Obtaining simulations number
00536       input->nsimulations
00537         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00538       if (error_code)
00539         {
00540           msg = gettext ("Bad simulations number");
00541           goto exit_on_error;
00542         }
00543     }
00544   else if (!xmlStrcmp (buffer, XML_SWEEP))
00545     input->algorithm = ALGORITHM_SWEEP;
00546   else if (!xmlStrcmp (buffer, XML_GENETIC))
00547     {
00548       input->algorithm = ALGORITHM_GENETIC;
00549
00550       // Obtaining population
00551       if (xmlHasProp (node, XML_NPOPULATION))
00552         {
00553           input->nsimulations
00554             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00555           if (error_code || input->nsimulations < 3)
00556             {
00557               msg = gettext ("Invalid population number");
00558               goto exit_on_error;
00559             }
00560         }
00561       else
00562         {
00563           msg = gettext ("No population number");
00564           goto exit_on_error;
00565         }
00566
00567       // Obtaining generations
00568       if (xmlHasProp (node, XML_NGENERATIONS))
00569         {
00570           input->niterations
00571             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00572           if (error_code || !input->niterations)
00573             {
00574               msg = gettext ("Invalid generations number");
00575               goto exit_on_error;
00576             }
00577         }
00578       else
00579         {
00580           msg = gettext ("No generations number");
00581           goto exit_on_error;
00582         }
00583
00584       // Obtaining mutation probability
00585       if (xmlHasProp (node, XML_MUTATION))
00586         {
00587           input->mutation_ratio
00588             = xml_node_get_float (node, XML_MUTATION, &error_code);
00589           if (error_code || input->mutation_ratio < 0.
00590               || input->mutation_ratio >= 1.)
00591             {
00592               msg = gettext ("Invalid mutation probability");
00593               goto exit_on_error;
00594             }
```

```
00595         }
00596       else
00597         {
00598           msg = gettext ("No mutation probability");
00599           goto exit_on_error;
00600         }
00601
00602       // Obtaining reproduction probability
00603       if (xmlHasProp (node, XML_REPRODUCTION))
00604         {
00605           input->reproduction_ratio
00606             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00607           if (error_code || input->reproduction_ratio < 0.
00608               || input->reproduction_ratio >= 1.0)
00609             {
00610               msg = gettext ("Invalid reproduction probability");
00611               goto exit_on_error;
00612             }
00613         }
00614       else
00615         {
00616           msg = gettext ("No reproduction probability");
00617           goto exit_on_error;
00618         }
00619
00620       // Obtaining adaptation probability
00621       if (xmlHasProp (node, XML_ADAPTATION))
00622         {
00623           input->adaptation_ratio
00624             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00625           if (error_code || input->adaptation_ratio < 0.
00626               || input->adaptation_ratio >= 1.)
00627             {
00628               msg = gettext ("Invalid adaptation probability");
00629               goto exit_on_error;
00630             }
00631         }
00632       else
00633         {
00634           msg = gettext ("No adaptation probability");
00635           goto exit_on_error;
00636         }
00637
00638       // Checking survivals
00639       i = input->mutation_ratio * input->nsimulations;
00640       i += input->reproduction_ratio * input->nsimulations;
00641       i += input->adaptation_ratio * input->nsimulations;
00642       if (i > input->nsimulations - 2)
00643         {
00644           msg = gettext
00645             ("No enough survival entities to reproduce the population");
00646           goto exit_on_error;
00647         }
00648     }
00649   else
00650     {
00651       msg = gettext ("Unknown algorithm");
00652       goto exit_on_error;
00653     }
00654
00655   if (input->algorithm == ALGORITHM_MONTE_CARLO
00656       || input->algorithm == ALGORITHM_SWEEP)
00657     {
00658
00659       // Obtaining iterations number
00660       input->niterations
00661         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00662       if (error_code == 1)
00663         input->niterations = 1;
00664       else if (error_code)
00665         {
00666           msg = gettext ("Bad iterations number");
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining best number
00671       if (xmlHasProp (node, XML_NBEST))
00672         {
00673           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00674           if (error_code || !input->nbest)
00675             {
00676               msg = gettext ("Invalid best number");
00677               goto exit_on_error;
00678             }
00679         }
00680       else
```

```
00681         input->nbest = 1;
00682
00683       // Obtaining tolerance
00684       if (xmlHasProp (node, XML_TOLERANCE))
00685         {
00686           input->tolerance
00687             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00688           if (error_code || input->tolerance < 0.)
00689             {
00690               msg = gettext ("Invalid tolerance");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         input->tolerance = 0.;
00696     }
00697
00698   // Reading the experimental data
00699   for (child = node->children; child; child = child->next)
00700     {
00701       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00702         break;
00703 #if DEBUG
00704       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00705 #endif
00706       if (xmlHasProp (child, XML_NAME))
00707         {
00708           input->experiment
00709             = g_realloc (input->experiment,
00710                          (1 + input->nexperiments) * sizeof (char *));
00711           input->experiment[input->nexperiments]
00712             = (char *) xmlGetProp (child, XML_NAME);
00713         }
00714       else
00715         {
00716           snprintf (buffer2, 64, "%s %u: %s",
00717                     gettext ("Experiment"),
00718                     input->nexperiments + 1, gettext ("no data file name"));
00719           msg = buffer2;
00720           goto exit_on_error;
00721         }
00722 #if DEBUG
00723       fprintf (stderr, "input_open: experiment=%s\n",
00724                input->experiment[input->nexperiments]);
00725 #endif
00726       input->weight = g_realloc (input->weight,
00727                                  (1 + input->nexperiments) * sizeof (double));
00728       if (xmlHasProp (child, XML_WEIGHT))
00729         {
00730           input->weight[input->nexperiments]
00731             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00732           if (error_code)
00733             {
00734               snprintf (buffer2, 64, "%s %u: %s",
00735                         gettext ("Experiment"),
00736                         input->nexperiments + 1, gettext ("bad weight"));
00737               msg = buffer2;
00738               goto exit_on_error;
00739             }
00740         }
00741       else
00742         input->weight[input->nexperiments] = 1.;
00743 #if DEBUG
00744       fprintf (stderr, "input_open: weight=%lg\n",
00745                input->weight[input->nexperiments]);
00746 #endif
00747       if (!input->nexperiments)
00748         input->ninputs = 0;
00749 #if DEBUG
00750       fprintf (stderr, "input_open: template[0]\n");
00751 #endif
00752       if (xmlHasProp (child, XML_TEMPLATE1))
00753         {
00754           input->template[0]
00755             = (char **) g_realloc (input->template[0],
00756                                    (1 + input->nexperiments) * sizeof (char *));
00757           input->template[0][input->nexperiments]
00758             = (char *) xmlGetProp (child, template[0]);
00759 #if DEBUG
00760           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00761                    input->nexperiments,
00762                    input->template[0][input->nexperiments]);
00763 #endif
00764           if (!input->nexperiments)
00765             ++input->ninputs;
00766 #if DEBUG
00767           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
```

```
00768 #endif
00769         }
00770       else
00771         {
00772           snprintf (buffer2, 64, "%s %u: %s",
00773                     gettext ("Experiment"),
00774                     input->nexperiments + 1, gettext ("no template"));
00775           msg = buffer2;
00776           goto exit_on_error;
00777         }
00778       for (i = 1; i < MAX_NINPUTS; ++i)
00779         {
00780 #if DEBUG
00781           fprintf (stderr, "input_open: template%u\n", i + 1);
00782 #endif
00783           if (xmlHasProp (child, template[i]))
00784             {
00785               if (input->nexperiments && input->ninputs <= i)
00786                 {
00787                   snprintf (buffer2, 64, "%s %u: %s",
00788                             gettext ("Experiment"),
00789                             input->nexperiments + 1,
00790                             gettext ("bad templates number"));
00791                   msg = buffer2;
00792                   goto exit_on_error;
00793                 }
00794               input->template[i] = (char **)
00795                 g_realloc (input->template[i],
00796                            (1 + input->nexperiments) * sizeof (char *));
00797               input->template[i][input->nexperiments]
00798                 = (char *) xmlGetProp (child, template[i]);
00799 #if DEBUG
00800               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00801                        input->nexperiments, i + 1,
00802                        input->template[i][input->nexperiments]);
00803 #endif
00804               if (!input->nexperiments)
00805                 ++input->ninputs;
00806 #if DEBUG
00807               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00808 #endif
00809             }
00810           else if (input->nexperiments && input->ninputs >= i)
00811             {
00812               snprintf (buffer2, 64, "%s %u: %s%u",
00813                         gettext ("Experiment"),
00814                         input->nexperiments + 1,
00815                         gettext ("no template"), i + 1);
00816               msg = buffer2;
00817               goto exit_on_error;
00818             }
00819           else
00820             break;
00821         }
00822       ++input->nexperiments;
00823 #if DEBUG
00824       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00825 #endif
00826     }
00827   if (!input->nexperiments)
00828     {
00829       msg = gettext ("No calibration experiments");
00830       goto exit_on_error;
00831     }
00832
00833   // Reading the variables data
00834   for (; child; child = child->next)
00835     {
00836       if (xmlStrcmp (child->name, XML_VARIABLE))
00837         {
00838           snprintf (buffer2, 64, "%s %u: %s",
00839                     gettext ("Variable"),
00840                     input->nvariables + 1, gettext ("bad XML node"));
00841           msg = buffer2;
00842           goto exit_on_error;
00843         }
00844       if (xmlHasProp (child, XML_NAME))
00845         {
00846           input->label = g_realloc
00847             (input->label, (1 + input->nvariables) * sizeof (char *));
00848           input->label[input->nvariables]
00849             = (char *) xmlGetProp (child, XML_NAME);
00850         }
00851       else
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
```

```
00855                     input->nvariables + 1, gettext ("no name"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_MINIMUM))
00860         {
00861           input->rangemin = g_realloc
00862             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00863           input->rangeminabs = g_realloc
00864             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00865           input->rangemin[input->nvariables]
00866             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00867           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00868             {
00869               input->rangeminabs[input->nvariables]
00870                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00871             }
00872           else
00873             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00874           if (input->rangemin[input->nvariables]
00875               < input->rangeminabs[input->nvariables])
00876             {
00877               snprintf (buffer2, 64, "%s %u: %s",
00878                         gettext ("Variable"),
00879                         input->nvariables + 1,
00880                         gettext ("minimum range not allowed"));
00881               msg = buffer2;
00882               goto exit_on_error;
00883             }
00884         }
00885       else
00886         {
00887           snprintf (buffer2, 64, "%s %u: %s",
00888                     gettext ("Variable"),
00889                     input->nvariables + 1, gettext ("no minimum range"));
00890           msg = buffer2;
00891           goto exit_on_error;
00892         }
00893       if (xmlHasProp (child, XML_MAXIMUM))
00894         {
00895           input->rangemax = g_realloc
00896             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00897           input->rangemaxabs = g_realloc
00898             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00899           input->rangemax[input->nvariables]
00900             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00901           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00902             input->rangemaxabs[input->nvariables]
00903               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00904           else
00905             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00906           if (input->rangemax[input->nvariables]
00907               > input->rangemaxabs[input->nvariables])
00908             {
00909               snprintf (buffer2, 64, "%s %u: %s",
00910                         gettext ("Variable"),
00911                         input->nvariables + 1,
00912                         gettext ("maximum range not allowed"));
00913               msg = buffer2;
00914               goto exit_on_error;
00915             }
00916         }
00917       else
00918         {
00919           snprintf (buffer2, 64, "%s %u: %s",
00920                     gettext ("Variable"),
00921                     input->nvariables + 1, gettext ("no maximum range"));
00922           msg = buffer2;
00923           goto exit_on_error;
00924         }
00925       if (input->rangemax[input->nvariables]
00926           < input->rangemin[input->nvariables])
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("bad range"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       input->precision = g_realloc
00935         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00936       if (xmlHasProp (child, XML_PRECISION))
00937         input->precision[input->nvariables]
00938           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00939       else
```

```
00940         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00941       if (input->algorithm == ALGORITHM_SWEEP)
00942         {
00943           if (xmlHasProp (child, XML_NSWEEPS))
00944             {
00945               input->nsweeps = (unsigned int *)
00946                 g_realloc (input->nsweeps,
00947                            (1 + input->nvariables) * sizeof (unsigned int));
00948               input->nsweeps[input->nvariables]
00949                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00950             }
00951           else
00952             {
00953               snprintf (buffer2, 64, "%s %u: %s",
00954                         gettext ("Variable"),
00955                         input->nvariables + 1, gettext ("no sweeps number"));
00956               msg = buffer2;
00957               goto exit_on_error;
00958             }
00959 #if DEBUG
00960           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00961                    input->nsweeps[input->nvariables], input->
      nsimulations);
00962 #endif
00963         }
00964       if (input->algorithm == ALGORITHM_GENETIC)
00965         {
00966           // Obtaining bits representing each variable
00967           if (xmlHasProp (child, XML_NBITS))
00968             {
00969               input->nbits = (unsigned int *)
00970                 g_realloc (input->nbits,
00971                            (1 + input->nvariables) * sizeof (unsigned int));
00972               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00973               if (error_code || !i)
00974                 {
00975                   snprintf (buffer2, 64, "%s %u: %s",
00976                             gettext ("Variable"),
00977                             input->nvariables + 1,
00978                             gettext ("invalid bits number"));
00979                   msg = buffer2;
00980                   goto exit_on_error;
00981                 }
00982               input->nbits[input->nvariables] = i;
00983             }
00984           else
00985             {
00986               snprintf (buffer2, 64, "%s %u: %s",
00987                         gettext ("Variable"),
00988                         input->nvariables + 1, gettext ("no bits number"));
00989               msg = buffer2;
00990               goto exit_on_error;
00991             }
00992         }
00993       ++input->nvariables;
00994     }
00995   if (!input->nvariables)
00996     {
00997       msg = gettext ("No calibration variables");
00998       goto exit_on_error;
00999     }
01000
01001   // Getting the working directory
01002   input->directory = g_path_get_dirname (filename);
01003   input->name = g_path_get_basename (filename);
01004
01005   // Closing the XML document
01006   xmlFreeDoc (doc);
01007
01008 #if DEBUG
01009   fprintf (stderr, "input_open: end\n");
01010 #endif
01011   return 1;
01012
01013 exit_on_error:
01014   show_error (msg);
01015   input_free ();
01016 #if DEBUG
01017   fprintf (stderr, "input_open: end\n");
01018 #endif
01019   return 0;
01020 }
01021
01033 void
01034 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01035 {
```

```
01036   unsigned int i;
01037   char buffer[32], value[32], *buffer2, *buffer3, *content;
01038   FILE *file;
01039   gsize length;
01040   GRegex *regex;
01041
01042 #if DEBUG
01043   fprintf (stderr, "calibrate_input: start\n");
01044 #endif
01045
01046   // Checking the file
01047   if (!template)
01048     goto calibrate_input_end;
01049
01050   // Opening template
01051   content = g_mapped_file_get_contents (template);
01052   length = g_mapped_file_get_length (template);
01053 #if DEBUG
01054   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01055            content);
01056 #endif
01057   file = g_fopen (input, "w");
01058
01059   // Parsing template
01060   for (i = 0; i < calibrate->nvariables; ++i)
01061     {
01062 #if DEBUG
01063       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01064 #endif
01065       snprintf (buffer, 32, "@variable%u@", i + 1);
01066       regex = g_regex_new (buffer, 0, 0, NULL);
01067       if (i == 0)
01068         {
01069           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01070                                              calibrate->label[i], 0, NULL);
01071 #if DEBUG
01072           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01073 #endif
01074         }
01075       else
01076         {
01077           length = strlen (buffer3);
01078           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01079                                              calibrate->label[i], 0, NULL);
01080           g_free (buffer3);
01081         }
01082       g_regex_unref (regex);
01083       length = strlen (buffer2);
01084       snprintf (buffer, 32, "@value%u@", i + 1);
01085       regex = g_regex_new (buffer, 0, 0, NULL);
01086       snprintf (value, 32, format[calibrate->precision[i]],
01087                 calibrate->value[simulation * calibrate->nvariables + i]);
01088
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_input: value=%s\n", value);
01091 #endif
01092       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01093                                          0, NULL);
01094       g_free (buffer2);
01095       g_regex_unref (regex);
01096     }
01097
01098   // Saving input file
01099   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01100   g_free (buffer3);
01101   fclose (file);
01102
01103 calibrate_input_end:
01104 #if DEBUG
01105   fprintf (stderr, "calibrate_input: end\n");
01106 #endif
01107   return;
01108 }
01109
01120 double
01121 calibrate_parse (unsigned int simulation, unsigned int experiment)
01122 {
01123   unsigned int i;
01124   double e;
01125   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01126     *buffer3, *buffer4;
01127   FILE *file_result;
01128
01129 #if DEBUG
01130   fprintf (stderr, "calibrate_parse: start\n");
01131   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01132            experiment);
```

```
01133 #endif
01134
01135   // Opening input files
01136   for (i = 0; i < calibrate->ninputs; ++i)
01137     {
01138       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01139 #if DEBUG
01140       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01141 #endif
01142       calibrate_input (simulation, &input[i][0],
01143                        calibrate->file[i][experiment]);
01144     }
01145   for (; i < MAX_NINPUTS; ++i)
01146     strcpy (&input[i][0], "");
01147 #if DEBUG
01148   fprintf (stderr, "calibrate_parse: parsing end\n");
01149 #endif
01150
01151   // Performing the simulation
01152   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01153   buffer2 = g_path_get_dirname (calibrate->simulator);
01154   buffer3 = g_path_get_basename (calibrate->simulator);
01155   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01156   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01157             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01158             input[6], input[7], output);
01159   g_free (buffer4);
01160   g_free (buffer3);
01161   g_free (buffer2);
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01164 #endif
01165   system (buffer);
01166
01167   // Checking the objective value function
01168   if (calibrate->evaluator)
01169     {
01170       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01171       buffer2 = g_path_get_dirname (calibrate->evaluator);
01172       buffer3 = g_path_get_basename (calibrate->evaluator);
01173       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01174       snprintf (buffer, 512, "\"%s\" %s %s %s",
01175                 buffer4, output, calibrate->experiment[experiment], result);
01176       g_free (buffer4);
01177       g_free (buffer3);
01178       g_free (buffer2);
01179 #if DEBUG
01180       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01181 #endif
01182       system (buffer);
01183       file_result = g_fopen (result, "r");
01184       e = atof (fgets (buffer, 512, file_result));
01185       fclose (file_result);
01186     }
01187   else
01188     {
01189       strcpy (result, "");
01190       file_result = g_fopen (output, "r");
01191       e = atof (fgets (buffer, 512, file_result));
01192       fclose (file_result);
01193     }
01194
01195   // Removing files
01196 #if !DEBUG
01197   for (i = 0; i < calibrate->ninputs; ++i)
01198     {
01199       if (calibrate->file[i][0])
01200         {
01201           snprintf (buffer, 512, RM " %s", &input[i][0]);
01202           system (buffer);
01203         }
01204     }
01205   snprintf (buffer, 512, RM " %s %s", output, result);
01206   system (buffer);
01207 #endif
01208
01209 #if DEBUG
01210   fprintf (stderr, "calibrate_parse: end\n");
01211 #endif
01212
01213   // Returning the objective function
01214   return e * calibrate->weight[experiment];
01215 }
01216
01221 void
01222 calibrate_print ()
01223 {
```

```
01224   unsigned int i;
01225   char buffer[512];
01226 #if HAVE_MPI
01227   if (!calibrate->mpi_rank)
01228     {
01229 #endif
01230       printf ("THE BEST IS\n");
01231       fprintf (calibrate->file_result, "THE BEST IS\n");
01232       printf ("error=%.15le\n", calibrate->error_old[0]);
01233       fprintf (calibrate->file_result, "error=%.15le\n",
01234                calibrate->error_old[0]);
01235       for (i = 0; i < calibrate->nvariables; ++i)
01236         {
01237           snprintf (buffer, 512, "%s=%s\n",
01238                     calibrate->label[i], format[calibrate->precision[i]]);
01239           printf (buffer, calibrate->value_old[i]);
01240           fprintf (calibrate->file_result, buffer, calibrate->
      value_old[i]);
01241         }
01242       fflush (calibrate->file_result);
01243 #if HAVE_MPI
01244     }
01245 #endif
01246 }
01247
01256 void
01257 calibrate_save_variables (unsigned int simulation, double error)
01258 {
01259   unsigned int i;
01260   char buffer[64];
01261 #if DEBUG
01262   fprintf (stderr, "calibrate_save_variables: start\n");
01263 #endif
01264   for (i = 0; i < calibrate->nvariables; ++i)
01265     {
01266       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01267       fprintf (calibrate->file_variables, buffer,
01268                calibrate->value[simulation * calibrate->nvariables + i]);
01269     }
01270   fprintf (calibrate->file_variables, "%.14le\n", error);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_save_variables: end\n");
01273 #endif
01274 }
01275
01284 void
01285 calibrate_best_thread (unsigned int simulation, double value)
01286 {
01287   unsigned int i, j;
01288   double e;
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_best_thread: start\n");
01291 #endif
01292   if (calibrate->nsaveds < calibrate->nbest
01293       || value < calibrate->error_best[calibrate->nsaveds - 1])
01294     {
01295       g_mutex_lock (mutex);
01296       if (calibrate->nsaveds < calibrate->nbest)
01297         ++calibrate->nsaveds;
01298       calibrate->error_best[calibrate->nsaveds - 1] = value;
01299       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01300       for (i = calibrate->nsaveds; --i;)
01301         {
01302           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01303             {
01304               j = calibrate->simulation_best[i];
01305               e = calibrate->error_best[i];
01306               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01307               calibrate->error_best[i] = calibrate->error_best[i - 1];
01308               calibrate->simulation_best[i - 1] = j;
01309               calibrate->error_best[i - 1] = e;
01310             }
01311           else
01312             break;
01313         }
01314       g_mutex_unlock (mutex);
01315     }
01316 #if DEBUG
01317   fprintf (stderr, "calibrate_best_thread: end\n");
01318 #endif
01319 }
01320
01329 void
01330 calibrate_best_sequential (unsigned int simulation, double value)
01331 {
01332   unsigned int i, j;
```

```
01333   double e;
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_best_sequential: start\n");
01336 #endif
01337   if (calibrate->nsaveds < calibrate->nbest
01338       || value < calibrate->error_best[calibrate->nsaveds - 1])
01339     {
01340       if (calibrate->nsaveds < calibrate->nbest)
01341         ++calibrate->nsaveds;
01342       calibrate->error_best[calibrate->nsaveds - 1] = value;
01343       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01344       for (i = calibrate->nsaveds; --i;)
01345         {
01346           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01347             {
01348               j = calibrate->simulation_best[i];
01349               e = calibrate->error_best[i];
01350               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01351               calibrate->error_best[i] = calibrate->error_best[i - 1];
01352               calibrate->simulation_best[i - 1] = j;
01353               calibrate->error_best[i - 1] = e;
01354             }
01355           else
01356             break;
01357         }
01358     }
01359 #if DEBUG
01360   fprintf (stderr, "calibrate_best_sequential: end\n");
01361 #endif
01362 }
01363
01371 void *
01372 calibrate_thread (ParallelData * data)
01373 {
01374   unsigned int i, j, thread;
01375   double e;
01376 #if DEBUG
01377   fprintf (stderr, "calibrate_thread: start\n");
01378 #endif
01379   thread = data->thread;
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01382            calibrate->thread[thread], calibrate->thread[thread + 1]);
01383 #endif
01384   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01385     {
01386       e = 0.;
01387       for (j = 0; j < calibrate->nexperiments; ++j)
01388         e += calibrate_parse (i, j);
01389       calibrate_best_thread (i, e);
01390       g_mutex_lock (mutex);
01391       calibrate_save_variables (i, e);
01392       g_mutex_unlock (mutex);
01393 #if DEBUG
01394       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01395 #endif
01396     }
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_thread: end\n");
01399 #endif
01400   g_thread_exit (NULL);
01401   return NULL;
01402 }
01403
01408 void
01409 calibrate_sequential ()
01410 {
01411   unsigned int i, j;
01412   double e;
01413 #if DEBUG
01414   fprintf (stderr, "calibrate_sequential: start\n");
01415   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01416            calibrate->nstart, calibrate->nend);
01417 #endif
01418   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01419     {
01420       e = 0.;
01421       for (j = 0; j < calibrate->nexperiments; ++j)
01422         e += calibrate_parse (i, j);
01423       calibrate_best_sequential (i, e);
01424       calibrate_save_variables (i, e);
01425 #if DEBUG
01426       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01427 #endif
01428     }
01429 #if DEBUG
```

```
01430   fprintf (stderr, "calibrate_sequential: end\n");
01431 #endif
01432 }
01433
01445 void
01446 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01447                  double *error_best)
01448 {
01449   unsigned int i, j, k, s[calibrate->nbest];
01450   double e[calibrate->nbest];
01451 #if DEBUG
01452   fprintf (stderr, "calibrate_merge: start\n");
01453 #endif
01454   i = j = k = 0;
01455   do
01456     {
01457       if (i == calibrate->nsaveds)
01458         {
01459           s[k] = simulation_best[j];
01460           e[k] = error_best[j];
01461           ++j;
01462           ++k;
01463           if (j == nsaveds)
01464             break;
01465         }
01466       else if (j == nsaveds)
01467         {
01468           s[k] = calibrate->simulation_best[i];
01469           e[k] = calibrate->error_best[i];
01470           ++i;
01471           ++k;
01472           if (i == calibrate->nsaveds)
01473             break;
01474         }
01475       else if (calibrate->error_best[i] > error_best[j])
01476         {
01477           s[k] = simulation_best[j];
01478           e[k] = error_best[j];
01479           ++j;
01480           ++k;
01481         }
01482       else
01483         {
01484           s[k] = calibrate->simulation_best[i];
01485           e[k] = calibrate->error_best[i];
01486           ++i;
01487           ++k;
01488         }
01489     }
01490   while (k < calibrate->nbest);
01491   calibrate->nsaveds = k;
01492   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01493   memcpy (calibrate->error_best, e, k * sizeof (double));
01494 #if DEBUG
01495   fprintf (stderr, "calibrate_merge: end\n");
01496 #endif
01497 }
01498
01503 #if HAVE_MPI
01504 void
01505 calibrate_synchronise ()
01506 {
01507   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01508   double error_best[calibrate->nbest];
01509   MPI_Status mpi_stat;
01510 #if DEBUG
01511   fprintf (stderr, "calibrate_synchronise: start\n");
01512 #endif
01513   if (calibrate->mpi_rank == 0)
01514     {
01515       for (i = 1; i < ntasks; ++i)
01516         {
01517           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01518           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01519                     MPI_COMM_WORLD, &mpi_stat);
01520           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01521                     MPI_COMM_WORLD, &mpi_stat);
01522           calibrate_merge (nsaveds, simulation_best, error_best);
01523         }
01524     }
01525   else
01526     {
01527       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01528       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01529                 MPI_COMM_WORLD);
01530       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01531                 MPI_COMM_WORLD);
```

```
01532     }
01533 #if DEBUG
01534   fprintf (stderr, "calibrate_synchronise: end\n");
01535 #endif
01536 }
01537 #endif
01538
01543 void
01544 calibrate_sweep ()
01545 {
01546   unsigned int i, j, k, l;
01547   double e;
01548   GThread *thread[nthreads];
01549   ParallelData data[nthreads];
01550 #if DEBUG
01551   fprintf (stderr, "calibrate_sweep: start\n");
01552 #endif
01553   for (i = 0; i < calibrate->nsimulations; ++i)
01554     {
01555       k = i;
01556       for (j = 0; j < calibrate->nvariables; ++j)
01557         {
01558           l = k % calibrate->nsweeps[j];
01559           k /= calibrate->nsweeps[j];
01560           e = calibrate->rangemin[j];
01561           if (calibrate->nsweeps[j] > 1)
01562             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01563               / (calibrate->nsweeps[j] - 1);
01564           calibrate->value[i * calibrate->nvariables + j] = e;
01565         }
01566     }
01567   calibrate->nsaveds = 0;
01568   if (nthreads <= 1)
01569     calibrate_sequential ();
01570   else
01571     {
01572       for (i = 0; i < nthreads; ++i)
01573         {
01574           data[i].thread = i;
01575           thread[i]
01576             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01577         }
01578       for (i = 0; i < nthreads; ++i)
01579         g_thread_join (thread[i]);
01580     }
01581 #if HAVE_MPI
01582   // Communicating tasks results
01583   calibrate_synchronise ();
01584 #endif
01585 #if DEBUG
01586   fprintf (stderr, "calibrate_sweep: end\n");
01587 #endif
01588 }
01589
01594 void
01595 calibrate_MonteCarlo ()
01596 {
01597   unsigned int i, j;
01598   GThread *thread[nthreads];
01599   ParallelData data[nthreads];
01600 #if DEBUG
01601   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01602 #endif
01603   for (i = 0; i < calibrate->nsimulations; ++i)
01604     for (j = 0; j < calibrate->nvariables; ++j)
01605       calibrate->value[i * calibrate->nvariables + j]
01606         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01607         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01608   calibrate->nsaveds = 0;
01609   if (nthreads <= 1)
01610     calibrate_sequential ();
01611   else
01612     {
01613       for (i = 0; i < nthreads; ++i)
01614         {
01615           data[i].thread = i;
01616           thread[i]
01617             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01618         }
01619       for (i = 0; i < nthreads; ++i)
01620         g_thread_join (thread[i]);
01621     }
01622 #if HAVE_MPI
01623   // Communicating tasks results
01624   calibrate_synchronise ();
01625 #endif
01626 #if DEBUG
```

```
01627   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01628 #endif
01629 }
01630
01638 double
01639 calibrate_genetic_objective (Entity * entity)
01640 {
01641   unsigned int j;
01642   double objective;
01643   char buffer[64];
01644 #if DEBUG
01645   fprintf (stderr, "calibrate_genetic_objective: start\n");
01646 #endif
01647   for (j = 0; j < calibrate->nvariables; ++j)
01648     {
01649       calibrate->value[entity->id * calibrate->nvariables + j]
01650         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01651     }
01652   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01653     objective += calibrate_parse (entity->id, j);
01654   g_mutex_lock (mutex);
01655   for (j = 0; j < calibrate->nvariables; ++j)
01656     {
01657       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01658       fprintf (calibrate->file_variables, buffer,
01659                genetic_get_variable (entity, calibrate->genetic_variable + j));
01660     }
01661   fprintf (calibrate->file_variables, "%.14le\n", objective);
01662   g_mutex_unlock (mutex);
01663 #if DEBUG
01664   fprintf (stderr, "calibrate_genetic_objective: end\n");
01665 #endif
01666   return objective;
01667 }
01668
01673 void
01674 calibrate_genetic ()
01675 {
01676   char *best_genome;
01677   double best_objective, *best_variable;
01678 #if DEBUG
01679   fprintf (stderr, "calibrate_genetic: start\n");
01680   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
01681            nthreads);
01682   fprintf (stderr,
01683            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
01684            calibrate->nvariables, calibrate->nsimulations,
01685            calibrate->niterations);
01686   fprintf (stderr,
01687            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01688            calibrate->mutation_ratio, calibrate->
      reproduction_ratio,
01689            calibrate->adaptation_ratio);
01690 #endif
01691   genetic_algorithm_default (calibrate->nvariables,
01692                              calibrate->genetic_variable,
01693                              calibrate->nsimulations,
01694                              calibrate->niterations,
01695                              calibrate->mutation_ratio,
01696                              calibrate->reproduction_ratio,
01697                              calibrate->adaptation_ratio,
01698                              &calibrate_genetic_objective,
01699                              &best_genome, &best_variable, &best_objective);
01700 #if DEBUG
01701   fprintf (stderr, "calibrate_genetic: the best\n");
01702 #endif
01703   calibrate->error_old = (double *) g_malloc (sizeof (double));
01704   calibrate->value_old
01705     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
01706   calibrate->error_old[0] = best_objective;
01707   memcpy (calibrate->value_old, best_variable,
01708           calibrate->nvariables * sizeof (double));
01709   g_free (best_genome);
01710   g_free (best_variable);
01711   calibrate_print ();
01712 #if DEBUG
01713   fprintf (stderr, "calibrate_genetic: end\n");
01714 #endif
01715 }
01716
01721 void
01722 calibrate_save_old ()
01723 {
01724   unsigned int i, j;
01725 #if DEBUG
01726   fprintf (stderr, "calibrate_save_old: start\n");
01727 #endif
```

```
01728   memcpy (calibrate->error_old, calibrate->error_best,
01729           calibrate->nbest * sizeof (double));
01730   for (i = 0; i < calibrate->nbest; ++i)
01731     {
01732       j = calibrate->simulation_best[i];
01733       memcpy (calibrate->value_old + i * calibrate->nvariables,
01734               calibrate->value + j * calibrate->nvariables,
01735               calibrate->nvariables * sizeof (double));
01736     }
01737 #if DEBUG
01738   for (i = 0; i < calibrate->nvariables; ++i)
01739     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
01740              i, calibrate->value_old[i]);
01741   fprintf (stderr, "calibrate_save_old: end\n");
01742 #endif
01743 }
01744
01750 void
01751 calibrate_merge_old ()
01752 {
01753   unsigned int i, j, k;
01754   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
01755     *enew, *eold;
01756 #if DEBUG
01757   fprintf (stderr, "calibrate_merge_old: start\n");
01758 #endif
01759   enew = calibrate->error_best;
01760   eold = calibrate->error_old;
01761   i = j = k = 0;
01762   do
01763     {
01764       if (*enew < *eold)
01765         {
01766           memcpy (v + k * calibrate->nvariables,
01767                   calibrate->value
01768                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
01769                   calibrate->nvariables * sizeof (double));
01770           e[k] = *enew;
01771           ++k;
01772           ++enew;
01773           ++i;
01774         }
01775       else
01776         {
01777           memcpy (v + k * calibrate->nvariables,
01778                   calibrate->value_old + j * calibrate->nvariables,
01779                   calibrate->nvariables * sizeof (double));
01780           e[k] = *eold;
01781           ++k;
01782           ++eold;
01783           ++j;
01784         }
01785     }
01786   while (k < calibrate->nbest);
01787   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
01788   memcpy (calibrate->error_old, e, k * sizeof (double));
01789 #if DEBUG
01790   fprintf (stderr, "calibrate_merge_old: end\n");
01791 #endif
01792 }
01793
01799 void
01800 calibrate_refine ()
01801 {
01802   unsigned int i, j;
01803   double d;
01804 #if HAVE_MPI
01805   MPI_Status mpi_stat;
01806 #endif
01807 #if DEBUG
01808   fprintf (stderr, "calibrate_refine: start\n");
01809 #endif
01810 #if HAVE_MPI
01811   if (!calibrate->mpi_rank)
01812     {
01813 #endif
01814       for (j = 0; j < calibrate->nvariables; ++j)
01815         {
01816           calibrate->rangemin[j] = calibrate->rangemax[j]
01817             = calibrate->value_old[j];
01818         }
01819       for (i = 0; ++i < calibrate->nbest;)
01820         {
01821           for (j = 0; j < calibrate->nvariables; ++j)
01822             {
```

```
01823               calibrate->rangemin[j]
01824                 = fmin (calibrate->rangemin[j],
01825                         calibrate->value_old[i * calibrate->nvariables + j]);
01826               calibrate->rangemax[j]
01827                 = fmax (calibrate->rangemax[j],
01828                         calibrate->value_old[i * calibrate->nvariables + j]);
01829             }
01830         }
01831       for (j = 0; j < calibrate->nvariables; ++j)
01832         {
01833           d = 0.5 * calibrate->tolerance
01834             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01835           calibrate->rangemin[j] -= d;
01836           calibrate->rangemin[j]
01837             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
01838           calibrate->rangemax[j] += d;
01839           calibrate->rangemax[j]
01840             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
01841           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
01842                   calibrate->rangemin[j], calibrate->rangemax[j]);
01843           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
01844                    calibrate->label[j], calibrate->rangemin[j],
01845                    calibrate->rangemax[j]);
01846         }
01847 #if HAVE_MPI
01848       for (i = 1; i < ntasks; ++i)
01849         {
01850           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
01851                     1, MPI_COMM_WORLD);
01852           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
01853                     1, MPI_COMM_WORLD);
01854         }
01855     }
01856   else
01857     {
01858       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01859                 MPI_COMM_WORLD, &mpi_stat);
01860       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01861                 MPI_COMM_WORLD, &mpi_stat);
01862     }
01863 #endif
01864 #if DEBUG
01865   fprintf (stderr, "calibrate_refine: end\n");
01866 #endif
01867 }
01868
01873 void
01874 calibrate_iterate ()
01875 {
01876   unsigned int i;
01877 #if DEBUG
01878   fprintf (stderr, "calibrate_iterate: start\n");
01879 #endif
01880   calibrate->error_old
01881     = (double *) g_malloc (calibrate->nbest * sizeof (double));
01882   calibrate->value_old = (double *)
01883     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
01884   calibrate_step ();
01885   calibrate_save_old ();
01886   calibrate_refine ();
01887   calibrate_print ();
01888   for (i = 1; i < calibrate->niterations; ++i)
01889     {
01890       calibrate_step ();
01891       calibrate_merge_old ();
01892       calibrate_refine ();
01893       calibrate_print ();
01894     }
01895 #if DEBUG
01896   fprintf (stderr, "calibrate_iterate: end\n");
01897 #endif
01898 }
01899
01904 void
01905 calibrate_free ()
01906 {
01907   unsigned int i, j;
01908 #if DEBUG
01909   fprintf (stderr, "calibrate_free: start\n");
01910 #endif
01911   for (i = 0; i < calibrate->nexperiments; ++i)
01912     {
01913       for (j = 0; j < calibrate->ninputs; ++j)
01914         g_mapped_file_unref (calibrate->file[j][i]);
01915     }
01916   for (i = 0; i < calibrate->ninputs; ++i)
01917     g_free (calibrate->file[i]);
```

```
01918   g_free (calibrate->error_old);
01919   g_free (calibrate->value_old);
01920   g_free (calibrate->value);
01921   g_free (calibrate->genetic_variable);
01922   g_free (calibrate->rangemax);
01923   g_free (calibrate->rangemin);
01924 #if DEBUG
01925   fprintf (stderr, "calibrate_free: end\n");
01926 #endif
01927 }
01928
01933 void
01934 calibrate_new ()
01935 {
01936   unsigned int i, j, *nbits;
01937
01938 #if DEBUG
01939   fprintf (stderr, "calibrate_new: start\n");
01940 #endif
01941
01942   // Initing pseudo-random numbers generator
01943   gsl_rng_set (calibrate->rng, calibrate->seed);
01944
01945   // Replacing the working dir
01946   g_chdir (input->directory);
01947
01948   // Obtaining the simulator file
01949   calibrate->simulator = input->simulator;
01950
01951   // Obtaining the evaluator file
01952   calibrate->evaluator = input->evaluator;
01953
01954   // Obtaining the pseudo-random numbers generator seed
01955   calibrate->seed = input->seed;
01956
01957   // Reading the algorithm
01958   calibrate->algorithm = input->algorithm;
01959   switch (calibrate->algorithm)
01960     {
01961     case ALGORITHM_MONTE_CARLO:
01962       calibrate_step = calibrate_MonteCarlo;
01963       break;
01964     case ALGORITHM_SWEEP:
01965       calibrate_step = calibrate_sweep;
01966       break;
01967     default:
01968       calibrate_step = calibrate_genetic;
01969       calibrate->mutation_ratio = input->mutation_ratio;
01970       calibrate->reproduction_ratio = input->
      reproduction_ratio;
01971       calibrate->adaptation_ratio = input->adaptation_ratio;
01972     }
01973   calibrate->nsimulations = input->nsimulations;
01974   calibrate->niterations = input->niterations;
01975   calibrate->nbest = input->nbest;
01976   calibrate->tolerance = input->tolerance;
01977
01978   calibrate->simulation_best
01979     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
01980   calibrate->error_best
01981     = (double *) alloca (calibrate->nbest * sizeof (double));
01982
01983   // Reading the experimental data
01984 #if DEBUG
01985   fprintf (stderr, "calibrate_new: current directory=%s\n",
01986            g_get_current_dir ());
01987 #endif
01988   calibrate->nexperiments = input->nexperiments;
01989   calibrate->ninputs = input->ninputs;
01990   calibrate->experiment = input->experiment;
01991   calibrate->weight = input->weight;
01992   for (i = 0; i < input->ninputs; ++i)
01993     {
01994       calibrate->template[i] = input->template[i];
01995       calibrate->file[i]
01996         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
01997     }
01998   for (i = 0; i < input->nexperiments; ++i)
01999     {
02000 #if DEBUG
02001       fprintf (stderr, "calibrate_new: i=%u\n", i);
02002       fprintf (stderr, "calibrate_new: experiment=%s\n",
02003                calibrate->experiment[i]);
02004       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
02005 #endif
02006       for (j = 0; j < input->ninputs; ++j)
02007         {
```

```
02008 #if DEBUG
02009           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
02010           fprintf (stderr, "calibrate_new: experiment=%u template%u=%s\n",
02011                    i, j + 1, calibrate->template[j][i]);
02012 #endif
02013           calibrate->file[j][i]
02014             = g_mapped_file_new (input->template[j][i], 0, NULL);
02015         }
02016     }
02017
02018   // Reading the variables data
02019 #if DEBUG
02020   fprintf (stderr, "calibrate_new: reading variables\n");
02021 #endif
02022   calibrate->nvariables = input->nvariables;
02023   calibrate->label = input->label;
02024   j = input->nvariables * sizeof (double);
02025   calibrate->rangemin = (double *) g_malloc (j);
02026   calibrate->rangemax = (double *) g_malloc (j);
02027   memcpy (calibrate->rangemin, input->rangemin, j);
02028   memcpy (calibrate->rangemax, input->rangemax, j);
02029   calibrate->rangeminabs = input->rangeminabs;
02030   calibrate->rangemaxabs = input->rangemaxabs;
02031   calibrate->precision = input->precision;
02032   calibrate->nsweeps = input->nsweeps;
02033   nbits = input->nbits;
02034   if (input->algorithm == ALGORITHM_SWEEP)
02035     calibrate->nsimulations = 1;
02036   else if (input->algorithm == ALGORITHM_GENETIC)
02037     for (i = 0; i < input->nvariables; ++i)
02038       {
02039         if (calibrate->algorithm == ALGORITHM_SWEEP)
02040           {
02041             calibrate->nsimulations *= input->nsweeps[i];
02042 #if DEBUG
02043             fprintf (stderr, "calibrate_new: nsweeps=%u nsimulations=%u\n",
02044                      calibrate->nsweeps[i], calibrate->nsimulations);
02045 #endif
02046           }
02047       }
02048
02049   // Allocating values
02050 #if DEBUG
02051   fprintf (stderr, "calibrate_new: allocating variables\n");
02052   fprintf (stderr, "calibrate_new: nvariables=%u\n", calibrate->nvariables);
02053 #endif
02054   calibrate->genetic_variable = NULL;
02055   if (calibrate->algorithm == ALGORITHM_GENETIC)
02056     {
02057       calibrate->genetic_variable = (GeneticVariable *)
02058         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02059       for (i = 0; i < calibrate->nvariables; ++i)
02060         {
02061 #if DEBUG
02062           fprintf (stderr, "calibrate_new: i=%u min=%lg max=%lg nbits=%u\n",
02063                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02064 #endif
02065           calibrate->genetic_variable[i].minimum = calibrate->
    rangemin[i];
02066           calibrate->genetic_variable[i].maximum = calibrate->
    rangemax[i];
02067           calibrate->genetic_variable[i].nbits = nbits[i];
02068         }
02069     }
02070 #if DEBUG
02071   fprintf (stderr, "calibrate_new: nvariables=%u nsimulations=%u\n",
02072            calibrate->nvariables, calibrate->nsimulations);
02073 #endif
02074   calibrate->value = (double *) g_malloc (calibrate->nsimulations *
02075                                           calibrate->nvariables *
02076                                           sizeof (double));
02077
02078   // Calculating simulations to perform on each task
02079 #if HAVE_MPI
02080 #if DEBUG
02081   fprintf (stderr, "calibrate_new: rank=%u ntasks=%u\n",
02082            calibrate->mpi_rank, ntasks);
02083 #endif
02084   calibrate->nstart = calibrate->mpi_rank * calibrate->
    nsimulations / ntasks;
02085   calibrate->nend = (1 + calibrate->mpi_rank) * calibrate->
    nsimulations
02086     / ntasks;
02087 #else
02088   calibrate->nstart = 0;
02089   calibrate->nend = calibrate->nsimulations;
02090 #endif
```

```
02091 #if DEBUG
02092   fprintf (stderr, "calibrate_new: nstart=%u nend=%u\n", calibrate->nstart,
02093            calibrate->nend);
02094 #endif
02095
02096   // Calculating simulations to perform on each thread
02097   calibrate->thread
02098     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02099   for (i = 0; i <= nthreads; ++i)
02100     {
02101       calibrate->thread[i] = calibrate->nstart
02102         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02103 #if DEBUG
02104       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02105                calibrate->thread[i]);
02106 #endif
02107     }
02108
02109   // Opening result files
02110   calibrate->file_result = g_fopen ("result", "w");
02111   calibrate->file_variables = g_fopen ("variables", "w");
02112
02113   // Performing the algorithm
02114   switch (calibrate->algorithm)
02115     {
02116       // Genetic algorithm
02117     case ALGORITHM_GENETIC:
02118       calibrate_genetic ();
02119       break;
02120
02121       // Iterative algorithm
02122     default:
02123       calibrate_iterate ();
02124     }
02125
02126   // Closing result files
02127   fclose (calibrate->file_variables);
02128   fclose (calibrate->file_result);
02129
02130 #if DEBUG
02131   fprintf (stderr, "calibrate_new: end\n");
02132 #endif
02133 }
02134
02135 #if HAVE_GTK
02136
02143 void
02144 input_save (char *filename)
02145 {
02146   unsigned int i, j;
02147   char *buffer;
02148   xmlDoc *doc;
02149   xmlNode *node, *child;
02150   GFile *file, *file2;
02151
02152   // Getting the input file directory
02153   input->name = g_path_get_basename (filename);
02154   input->directory = g_path_get_dirname (filename);
02155   file = g_file_new_for_path (input->directory);
02156
02157   // Opening the input file
02158   doc = xmlNewDoc ((const xmlChar *) "1.0");
02159
02160   // Setting root XML node
02161   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02162   xmlDocSetRootElement (doc, node);
02163
02164   // Adding properties to the root XML node
02165   file2 = g_file_new_for_path (input->simulator);
02166   buffer = g_file_get_relative_path (file, file2);
02167   g_object_unref (file2);
02168   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02169   g_free (buffer);
02170   if (input->evaluator)
02171     {
02172       file2 = g_file_new_for_path (input->evaluator);
02173       buffer = g_file_get_relative_path (file, file2);
02174       g_object_unref (file2);
02175       if (xmlStrlen ((xmlChar *) buffer))
02176         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02177       g_free (buffer);
02178     }
02179   if (input->seed != DEFAULT_RANDOM_SEED)
02180     xml_node_set_uint (node, XML_SEED, input->seed);
02181
02182   // Setting the algorithm
02183   buffer = (char *) g_malloc (64);
```

```
02184   switch (input->algorithm)
02185     {
02186     case ALGORITHM_MONTE_CARLO:
02187       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02188       snprintf (buffer, 64, "%u", input->nsimulations);
02189       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%u", input->niterations);
02191       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%.3lg", input->tolerance);
02193       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02194       snprintf (buffer, 64, "%u", input->nbest);
02195       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02196       break;
02197     case ALGORITHM_SWEEP:
02198       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02199       snprintf (buffer, 64, "%u", input->niterations);
02200       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%.3lg", input->tolerance);
02202       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02203       snprintf (buffer, 64, "%u", input->nbest);
02204       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02205       break;
02206     default:
02207       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02208       snprintf (buffer, 64, "%u", input->nsimulations);
02209       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%u", input->niterations);
02211       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02213       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02215       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02216       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02217       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02218       break;
02219     }
02220   g_free (buffer);
02221
02222   // Setting the experimental data
02223   for (i = 0; i < input->nexperiments; ++i)
02224     {
02225       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02226       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02227       if (input->weight[i] != 1.)
02228         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02229       for (j = 0; j < input->ninputs; ++j)
02230         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02231     }
02232
02233   // Setting the variables data
02234   for (i = 0; i < input->nvariables; ++i)
02235     {
02236       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02237       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02238       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02239       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02240         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02241       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02242       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02243         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02244       if (input->precision[i] != DEFAULT_PRECISION)
02245         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02246       if (input->algorithm == ALGORITHM_SWEEP)
02247         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02248       else if (input->algorithm == ALGORITHM_GENETIC)
02249         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02250     }
02251
02252   // Saving the XML file
02253   xmlSaveFormatFile (filename, doc, 1);
02254
02255   // Freeing memory
02256   xmlFreeDoc (doc);
02257 }
02258
02263 void
02264 options_new ()
02265 {
02266   options->label_processors
```

```
02267     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02268   options->spin_processors
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02270   gtk_spin_button_set_value (options->spin_processors, (gdouble)
    nthreads);
02271   options->label_seed = (GtkLabel *)
02272     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02273   options->spin_seed = (GtkSpinButton *)
02274     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02275   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02276   options->grid = (GtkGrid *) gtk_grid_new ();
02277   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02278                    0, 0, 1, 1);
02279   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02280                    1, 0, 1, 1);
02281   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 1, 1, 1);
02282   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 1, 1, 1);
02283   gtk_widget_show_all (GTK_WIDGET (options->grid));
02284   options->dialog = (GtkDialog *)
02285     gtk_dialog_new_with_buttons (gettext ("Options"),
02286                                  window->window,
02287                                  GTK_DIALOG_MODAL,
02288                                  gettext ("_OK"), GTK_RESPONSE_OK,
02289                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02290                                  NULL);
02291   gtk_container_add
02292     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02293      GTK_WIDGET (options->grid));
02294   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02295     {
02296       nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02297       input->seed
02298         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02299     }
02300   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02301 }
02302
02307 void
02308 running_new ()
02309 {
02310 #if DEBUG
02311   fprintf (stderr, "running_new: start\n");
02312 #endif
02313   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02314   running->dialog = (GtkDialog *)
02315     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02316                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02317   gtk_container_add
02318     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02319      GTK_WIDGET (running->label));
02320   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02321 #if DEBUG
02322   fprintf (stderr, "running_new: end\n");
02323 #endif
02324 }
02325
02331 int
02332 window_save ()
02333 {
02334   char *buffer;
02335   GtkFileChooserDialog *dlg;
02336
02337 #if DEBUG
02338   fprintf (stderr, "window_save: start\n");
02339 #endif
02340
02341   // Opening the saving dialog
02342   dlg = (GtkFileChooserDialog *)
02343     gtk_file_chooser_dialog_new (gettext ("Save file"),
02344                                  window->window,
02345                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02346                                  gettext ("_Cancel"),
02347                                  GTK_RESPONSE_CANCEL,
02348                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02349   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02350   buffer = g_build_filename (input->directory, input->name, NULL);
02351   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02352   g_free (buffer);
02353
02354   // If OK response then saving
02355   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02356     {
02357
02358       // Adding properties to the root XML node
02359       input->simulator = gtk_file_chooser_get_filename
02360         (GTK_FILE_CHOOSER (window->button_simulator));
02361       if (gtk_toggle_button_get_active
```

```
02362           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02363         input->evaluator = gtk_file_chooser_get_filename
02364           (GTK_FILE_CHOOSER (window->button_evaluator));
02365       else
02366         input->evaluator = NULL;
02367
02368       // Setting the algorithm
02369       switch (window_get_algorithm ())
02370         {
02371         case ALGORITHM_MONTE_CARLO:
02372           input->algorithm = ALGORITHM_MONTE_CARLO;
02373           input->nsimulations
02374             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02375           input->niterations
02376             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02377           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02378           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02379           break;
02380         case ALGORITHM_SWEEP:
02381           input->algorithm = ALGORITHM_SWEEP;
02382           input->niterations
02383             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02384           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02385           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02386           break;
02387         default:
02388           input->algorithm = ALGORITHM_GENETIC;
02389           input->nsimulations
02390             = gtk_spin_button_get_value_as_int (window->spin_population);
02391           input->niterations
02392             = gtk_spin_button_get_value_as_int (window->spin_generations);
02393           input->mutation_ratio
02394             = gtk_spin_button_get_value (window->spin_mutation);
02395           input->reproduction_ratio
02396             = gtk_spin_button_get_value (window->spin_reproduction);
02397           input->adaptation_ratio
02398             = gtk_spin_button_get_value (window->spin_adaptation);
02399           break;
02400         }
02401
02402       // Saving the XML file
02403       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02404       input_save (buffer);
02405
02406       // Closing and freeing memory
02407       g_free (buffer);
02408       gtk_widget_destroy (GTK_WIDGET (dlg));
02409 #if DEBUG
02410       fprintf (stderr, "window_save: end\n");
02411 #endif
02412       return 1;
02413     }
02414
02415   // Closing and freeing memory
02416   gtk_widget_destroy (GTK_WIDGET (dlg));
02417 #if DEBUG
02418   fprintf (stderr, "window_save: end\n");
02419 #endif
02420   return 0;
02421 }
02422
02427 void
02428 window_run ()
02429 {
02430   unsigned int i;
02431   char *msg, *msg2, buffer[64], buffer2[64];
02432 #if DEBUG
02433   fprintf (stderr, "window_run: start\n");
02434 #endif
02435   if (!window_save ())
02436     {
02437 #if DEBUG
02438       fprintf (stderr, "window_run: end\n");
02439 #endif
02440       return;
02441     }
02442   running_new ();
02443   while (gtk_events_pending ())
02444     gtk_main_iteration ();
02445   calibrate_new ();
02446   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02447   snprintf (buffer, 64, "error=%.15le\n", calibrate->error_old[0]);
02448   msg2 = g_strdup (buffer);
```

```
02449   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02450     {
02451       snprintf (buffer, 64, "%s=%s\n",
02452                 calibrate->label[i], format[calibrate->precision[i]]);
02453       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02454       msg = g_strconcat (msg2, buffer2, NULL);
02455       g_free (msg2);
02456     }
02457   show_message (gettext ("Best result"), msg2, INFO_TYPE);
02458   g_free (msg2);
02459   calibrate_free ();
02460 #if DEBUG
02461   fprintf (stderr, "window_run: end\n");
02462 #endif
02463 }
02464
02469 void
02470 window_help ()
02471 {
02472   char *buffer, *buffer2;
02473   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
02474                               gettext ("user-manual.pdf"), NULL);
02475   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02476   g_free (buffer2);
02477   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02478   g_free (buffer);
02479 }
02480
02485 void
02486 window_about ()
02487 {
02488   gchar *authors[] = {
02489     "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02490     "Borja Latorre Garcés (borja.latorre@csic.es)",
02491     NULL
02492   };
02493   gtk_show_about_dialog (window->window,
02494                          "program_name",
02495                          "Calibrator",
02496                          "comments",
02497                          gettext ("A software to make calibrations of "
02498                                   "empirical parameters"),
02499                          "authors", authors,
02500                          "translator-credits",
02501                          "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02502                          "version", "1.0.1",
02503                          "copyright",
02504                          "Copyright 2012-2015 Javier Burguete Tolosa",
02505                          "logo", window->logo,
02506                          "website-label", gettext ("Website"),
02507                          "website",
02508                          "https://github.com/jburguete/calibrator", NULL);
02509 }
02510
02516 int
02517 window_get_algorithm ()
02518 {
02519   unsigned int i;
02520   for (i = 0; i < NALGORITHMS; ++i)
02521     if (gtk_toggle_button_get_active
02522         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02523       break;
02524   return i;
02525 }
02526
02531 void
02532 window_update ()
02533 {
02534   unsigned int i;
02535   gtk_widget_set_sensitive
02536     (GTK_WIDGET (window->button_evaluator),
02537      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02538                                    (window->check_evaluator)));
02539   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02540   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02541   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02542   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02543   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02544   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
02545   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02546   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02547   gtk_widget_hide (GTK_WIDGET (window->label_population));
02548   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02549   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02550   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02551   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02552   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
```

```
02553   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02554   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
02555   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
02556   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
02557   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
02558   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
02559   gtk_widget_hide (GTK_WIDGET (window->label_bits));
02560   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
02561   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
02562   switch (window_get_algorithm ())
02563     {
02564     case ALGORITHM_MONTE_CARLO:
02565       gtk_widget_show (GTK_WIDGET (window->label_simulations));
02566       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
02567       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02568       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02569       if (i > 1)
02570         {
02571           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02572           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02573           gtk_widget_show (GTK_WIDGET (window->label_bests));
02574           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02575         }
02576       break;
02577     case ALGORITHM_SWEEP:
02578       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02579       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02580       if (i > 1)
02581         {
02582           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02583           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02584           gtk_widget_show (GTK_WIDGET (window->label_bests));
02585           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02586         }
02587       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
02588       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
02589       break;
02590     default:
02591       gtk_widget_show (GTK_WIDGET (window->label_population));
02592       gtk_widget_show (GTK_WIDGET (window->spin_population));
02593       gtk_widget_show (GTK_WIDGET (window->label_generations));
02594       gtk_widget_show (GTK_WIDGET (window->spin_generations));
02595       gtk_widget_show (GTK_WIDGET (window->label_mutation));
02596       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
02597       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
02598       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
02599       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
02600       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
02601       gtk_widget_show (GTK_WIDGET (window->label_bits));
02602       gtk_widget_show (GTK_WIDGET (window->spin_bits));
02603     }
02604   gtk_widget_set_sensitive
02605     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
02606   gtk_widget_set_sensitive
02607     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
02608   for (i = 0; i < input->ninputs; ++i)
02609     {
02610       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02611       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02612       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
02613       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
02614       g_signal_handler_block
02615         (window->check_template[i], window->id_template[i]);
02616       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
02617       gtk_toggle_button_set_active
02618         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
02619       g_signal_handler_unblock
02620         (window->button_template[i], window->id_input[i]);
02621       g_signal_handler_unblock
02622         (window->check_template[i], window->id_template[i]);
02623     }
02624   if (i > 0)
02625     {
02626       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
02627       gtk_widget_set_sensitive
02628         (GTK_WIDGET (window->button_template[i - 1]),
02629          gtk_toggle_button_get_active
02630          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
02631     }
02632   if (i < MAX_NINPUTS)
02633     {
02634       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02635       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02636       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
```

```
02637       gtk_widget_set_sensitive
02638         (GTK_WIDGET (window->button_template[i]),
02639          gtk_toggle_button_get_active
02640          GTK_TOGGLE_BUTTON (window->check_template[i]));
02641       g_signal_handler_block
02642         (window->check_template[i], window->id_template[i]);
02643       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02644       gtk_toggle_button_set_active
02645         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02646       g_signal_handler_unblock
02647         (window->button_template[i], window->id_input[i]);
02648       g_signal_handler_unblock
02649         (window->check_template[i], window->id_template[i]);
02650     }
02651   while (++i < MAX_NINPUTS)
02652     {
02653       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02654       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02655     }
02656   gtk_widget_set_sensitive
02657     (GTK_WIDGET (window->spin_minabs),
02658      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02659   gtk_widget_set_sensitive
02660     (GTK_WIDGET (window->spin_maxabs),
02661      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02662 }
02663
02668 void
02669 window_set_algorithm ()
02670 {
02671   int i;
02672 #if DEBUG
02673   fprintf (stderr, "window_set_algorithm: start\n");
02674 #endif
02675   i = window_get_algorithm ();
02676   switch (i)
02677     {
02678     case ALGORITHM_SWEEP:
02679       input->nsweeps = (unsigned int *) g_realloc
02680         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02681       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02682       if (i < 0)
02683         i = 0;
02684       gtk_spin_button_set_value (window->spin_sweeps,
02685                                  (gdouble) input->nsweeps[i]);
02686       break;
02687     case ALGORITHM_GENETIC:
02688       input->nbits = (unsigned int *) g_realloc
02689         (input->nbits, input->nvariables * sizeof (unsigned int));
02690       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02691       if (i < 0)
02692         i = 0;
02693       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02694     }
02695   window_update ();
02696 #if DEBUG
02697   fprintf (stderr, "window_set_algorithm: end\n");
02698 #endif
02699 }
02700
02705 void
02706 window_set_experiment ()
02707 {
02708   unsigned int i, j;
02709   char *buffer1, *buffer2;
02710 #if DEBUG
02711   fprintf (stderr, "window_set_experiment: start\n");
02712 #endif
02713   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02714   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02715   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02716   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02717   g_free (buffer1);
02718   g_signal_handler_block
02719     (window->button_experiment, window->id_experiment_name);
02720   gtk_file_chooser_set_filename
02721     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
02722   g_signal_handler_unblock
02723     (window->button_experiment, window->id_experiment_name);
02724   g_free (buffer2);
02725   for (j = 0; j < input->ninputs; ++j)
02726     {
02727       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02728       buffer2
```

```
02729         = g_build_filename (input->directory, input->template[j][i], NULL);
02730       gtk_file_chooser_set_filename
02731         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02732       g_free (buffer2);
02733       g_signal_handler_unblock
02734         (window->button_template[j], window->id_input[j]);
02735     }
02736 #if DEBUG
02737   fprintf (stderr, "window_set_experiment: end\n");
02738 #endif
02739 }
02740
02745 void
02746 window_remove_experiment ()
02747 {
02748   unsigned int i, j;
02749   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02750   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02751   gtk_combo_box_text_remove (window->combo_experiment, i);
02752   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02753   xmlFree (input->experiment[i]);
02754   --input->nexperiments;
02755   for (j = i; j < input->nexperiments; ++j)
02756     {
02757       input->experiment[j] = input->experiment[j + 1];
02758       input->weight[j] = input->weight[j + 1];
02759     }
02760   j = input->nexperiments - 1;
02761   if (i > j)
02762     i = j;
02763   for (j = 0; j < input->ninputs; ++j)
02764     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02765   g_signal_handler_block
02766     (window->button_experiment, window->id_experiment_name);
02767   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02768   g_signal_handler_unblock
02769     (window->button_experiment, window->id_experiment_name);
02770   for (j = 0; j < input->ninputs; ++j)
02771     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02772   window_update ();
02773 }
02774
02779 void
02780 window_add_experiment ()
02781 {
02782   unsigned int i, j;
02783   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02784   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02785   gtk_combo_box_text_insert_text
02786     (window->combo_experiment, i, input->experiment[i]);
02787   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02788   input->experiment = (char **) g_realloc
02789     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02790   input->weight = (double *) g_realloc
02791     (input->weight, (input->nexperiments + 1) * sizeof (double));
02792   for (j = input->nexperiments - 1; j > i; --j)
02793     {
02794       input->experiment[j + 1] = input->experiment[j];
02795       input->weight[j + 1] = input->weight[j];
02796     }
02797   input->experiment[j + 1]
02798     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02799   input->weight[j + 1] = input->weight[j];
02800   ++input->nexperiments;
02801   for (j = 0; j < input->ninputs; ++j)
02802     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02803   g_signal_handler_block
02804     (window->button_experiment, window->id_experiment_name);
02805   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02806   g_signal_handler_unblock
02807     (window->button_experiment, window->id_experiment_name);
02808   for (j = 0; j < input->ninputs; ++j)
02809     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02810   window_update ();
02811 }
02812
02817 void
02818 window_name_experiment ()
02819 {
```

```
02820   unsigned int i;
02821   char *buffer;
02822   GFile *file1, *file2;
02823 #if DEBUG
02824   fprintf (stderr, "window_name_experiment: start\n");
02825 #endif
02826   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02827   file1
02828     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02829   file2 = g_file_new_for_path (input->directory);
02830   buffer = g_file_get_relative_path (file2, file1);
02831   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02832   gtk_combo_box_text_remove (window->combo_experiment, i);
02833   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02834   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02835   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02836   g_free (buffer);
02837   g_object_unref (file2);
02838   g_object_unref (file1);
02839 #if DEBUG
02840   fprintf (stderr, "window_name_experiment: end\n");
02841 #endif
02842 }
02843
02848 void
02849 window_weight_experiment ()
02850 {
02851   unsigned int i;
02852 #if DEBUG
02853   fprintf (stderr, "window_weight_experiment: start\n");
02854 #endif
02855   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02856   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02857 #if DEBUG
02858   fprintf (stderr, "window_weight_experiment: end\n");
02859 #endif
02860 }
02861
02867 void
02868 window_inputs_experiment ()
02869 {
02870   unsigned int j;
02871 #if DEBUG
02872   fprintf (stderr, "window_inputs_experiment: start\n");
02873 #endif
02874   j = input->ninputs - 1;
02875   if (j
02876       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02877                                         (window->check_template[j])))
02878     --input->ninputs;
02879   if (input->ninputs < MAX_NINPUTS
02880       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02881                                        (window->check_template[j])))
02882     {
02883       ++input->ninputs;
02884       for (j = 0; j < input->ninputs; ++j)
02885         {
02886           input->template[j] = (char **)
02887             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02888         }
02889     }
02890   window_update ();
02891 #if DEBUG
02892   fprintf (stderr, "window_inputs_experiment: end\n");
02893 #endif
02894 }
02895
02903 void
02904 window_template_experiment (void *data)
02905 {
02906   unsigned int i, j;
02907   char *buffer;
02908   GFile *file1, *file2;
02909 #if DEBUG
02910   fprintf (stderr, "window_template_experiment: start\n");
02911 #endif
02912   i = (size_t) data;
02913   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02914   file1
02915     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02916   file2 = g_file_new_for_path (input->directory);
02917   buffer = g_file_get_relative_path (file2, file1);
02918   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02919   g_free (buffer);
02920   g_object_unref (file2);
```

```
02921   g_object_unref (file1);
02922 #if DEBUG
02923   fprintf (stderr, "window_template_experiment: end\n");
02924 #endif
02925 }
02926
02931 void
02932 window_set_variable ()
02933 {
02934   unsigned int i;
02935 #if DEBUG
02936   fprintf (stderr, "window_set_variable: start\n");
02937 #endif
02938   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02939   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02940   gtk_entry_set_text (window->entry_variable, input->label[i]);
02941   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02942   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02943   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
02944   if (input->rangeminabs[i] != -G_MAXDOUBLE)
02945     {
02946       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
02947       gtk_toggle_button_set_active
02948         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
02949     }
02950   else
02951     {
02952       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
02953       gtk_toggle_button_set_active
02954         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
02955     }
02956   if (input->rangemaxabs[i] != G_MAXDOUBLE)
02957     {
02958       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
02959       gtk_toggle_button_set_active
02960         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
02961     }
02962   else
02963     {
02964       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
02965       gtk_toggle_button_set_active
02966         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
02967     }
02968   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
02969 #if DEBUG
02970   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
02971            input->precision[i]);
02972 #endif
02973   switch (window_get_algorithm ())
02974     {
02975     case ALGORITHM_SWEEP:
02976       gtk_spin_button_set_value (window->spin_sweeps,
02977                                  (gdouble) input->nsweeps[i]);
02978 #if DEBUG
02979       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
02980                input->nsweeps[i]);
02981 #endif
02982       break;
02983     case ALGORITHM_GENETIC:
02984       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02985 #if DEBUG
02986       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
02987                input->nbits[i]);
02988 #endif
02989       break;
02990     }
02991   window_update ();
02992 #if DEBUG
02993   fprintf (stderr, "window_set_variable: end\n");
02994 #endif
02995 }
02996
03001 void
03002 window_remove_variable ()
03003 {
03004   unsigned int i, j;
03005   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03006   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03007   gtk_combo_box_text_remove (window->combo_variable, i);
03008   g_signal_handler_unblock (window->combo_variable, window->
```

```
      id_variable);
03009   xmlFree (input->label[i]);
03010   --input->nvariables;
03011   for (j = i; j < input->nvariables; ++j)
03012     {
03013       input->label[j] = input->label[j + 1];
03014       input->rangemin[j] = input->rangemin[j + 1];
03015       input->rangemax[j] = input->rangemax[j + 1];
03016       input->rangeminabs[j] = input->rangeminabs[j + 1];
03017       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03018       input->precision[j] = input->precision[j + 1];
03019       switch (window_get_algorithm ())
03020         {
03021         case ALGORITHM_SWEEP:
03022           input->nsweeps[j] = input->nsweeps[j + 1];
03023           break;
03024         case ALGORITHM_GENETIC:
03025           input->nbits[j] = input->nbits[j + 1];
03026         }
03027     }
03028   j = input->nvariables - 1;
03029   if (i > j)
03030     i = j;
03031   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03032   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03033   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03034   window_update ();
03035 }
03036
03041 void
03042 window_add_variable ()
03043 {
03044   unsigned int i, j;
03045 #if DEBUG
03046   fprintf (stderr, "window_add_variable: start\n");
03047 #endif
03048   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03049   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03050   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03051   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03052   input->label = (char **) g_realloc
03053     (input->label, (input->nvariables + 1) * sizeof (char *));
03054   input->rangemin = (double *) g_realloc
03055     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03056   input->rangemax = (double *) g_realloc
03057     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03058   input->rangeminabs = (double *) g_realloc
03059     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03060   input->rangemaxabs = (double *) g_realloc
03061     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03062   input->precision = (unsigned int *) g_realloc
03063     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03064   for (j = input->nvariables - 1; j > i; --j)
03065     {
03066       input->label[j + 1] = input->label[j];
03067       input->rangemin[j + 1] = input->rangemin[j];
03068       input->rangemax[j + 1] = input->rangemax[j];
03069       input->rangeminabs[j + 1] = input->rangeminabs[j];
03070       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03071       input->precision[j + 1] = input->precision[j];
03072     }
03073   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03074   input->rangemin[j + 1] = input->rangemin[j];
03075   input->rangemax[j + 1] = input->rangemax[j];
03076   input->rangeminabs[j + 1] = input->rangeminabs[j];
03077   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03078   input->precision[j + 1] = input->precision[j];
03079   switch (window_get_algorithm ())
03080     {
03081     case ALGORITHM_SWEEP:
03082       input->nsweeps = (unsigned int *) g_realloc
03083         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03084       for (j = input->nvariables - 1; j > i; --j)
03085         input->nsweeps[j + 1] = input->nsweeps[j];
03086       input->nsweeps[j + 1] = input->nsweeps[j];
03087       break;
03088     case ALGORITHM_GENETIC:
03089       input->nbits = (unsigned int *) g_realloc
03090         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03091       for (j = input->nvariables - 1; j > i; --j)
03092         input->nbits[j + 1] = input->nbits[j];
03093       input->nbits[j + 1] = input->nbits[j];
```

```
03094     }
03095   ++input->nvariables;
03096   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03097   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03098   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03099   window_update ();
03100 #if DEBUG
03101   fprintf (stderr, "window_add_variable: end\n");
03102 #endif
03103 }
03104
03109 void
03110 window_label_variable ()
03111 {
03112   unsigned int i;
03113   const char *buffer;
03114 #if DEBUG
03115   fprintf (stderr, "window_label_variable: start\n");
03116 #endif
03117   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03118   buffer = gtk_entry_get_text (window->entry_variable);
03119   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03120   gtk_combo_box_text_remove (window->combo_variable, i);
03121   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03122   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03123   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03124 #if DEBUG
03125   fprintf (stderr, "window_label_variable: end\n");
03126 #endif
03127 }
03128
03133 void
03134 window_precision_variable ()
03135 {
03136   unsigned int i;
03137 #if DEBUG
03138   fprintf (stderr, "window_precision_variable: start\n");
03139 #endif
03140   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03141   input->precision[i]
03142     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03143   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03144   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03145   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03146   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03147 #if DEBUG
03148   fprintf (stderr, "window_precision_variable: end\n");
03149 #endif
03150 }
03151
03156 void
03157 window_rangemin_variable ()
03158 {
03159   unsigned int i;
03160 #if DEBUG
03161   fprintf (stderr, "window_rangemin_variable: start\n");
03162 #endif
03163   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03164   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03165 #if DEBUG
03166   fprintf (stderr, "window_rangemin_variable: end\n");
03167 #endif
03168 }
03169
03174 void
03175 window_rangemax_variable ()
03176 {
03177   unsigned int i;
03178 #if DEBUG
03179   fprintf (stderr, "window_rangemax_variable: start\n");
03180 #endif
03181   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03182   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03183 #if DEBUG
03184   fprintf (stderr, "window_rangemax_variable: end\n");
03185 #endif
03186 }
03187
03192 void
03193 window_rangeminabs_variable ()
03194 {
03195   unsigned int i;
03196 #if DEBUG
```

```
03197   fprintf (stderr, "window_rangeminabs_variable: start\n");
03198 #endif
03199   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03200   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03201 #if DEBUG
03202   fprintf (stderr, "window_rangeminabs_variable: end\n");
03203 #endif
03204 }
03205
03210 void
03211 window_rangemaxabs_variable ()
03212 {
03213   unsigned int i;
03214 #if DEBUG
03215   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03216 #endif
03217   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03218   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03219 #if DEBUG
03220   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03221 #endif
03222 }
03223
03228 void
03229 window_update_variable ()
03230 {
03231   int i;
03232 #if DEBUG
03233   fprintf (stderr, "window_update_variable: start\n");
03234 #endif
03235   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03236   if (i < 0)
03237     i = 0;
03238   switch (window_get_algorithm ())
03239     {
03240     case ALGORITHM_SWEEP:
03241       input->nsweeps[i]
03242         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03243 #if DEBUG
03244       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03245                input->nsweeps[i]);
03246 #endif
03247       break;
03248     case ALGORITHM_GENETIC:
03249       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03250 #if DEBUG
03251       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
03252                input->nbits[i]);
03253 #endif
03254     }
03255 #if DEBUG
03256   fprintf (stderr, "window_update_variable: end\n");
03257 #endif
03258 }
03259
03267 int
03268 window_read (char *filename)
03269 {
03270   unsigned int i;
03271   char *buffer;
03272 #if DEBUG
03273   fprintf (stderr, "window_read: start\n");
03274 #endif
03275
03276   // Reading new input file
03277   input_free ();
03278   if (!input_open (filename))
03279     return 0;
03280
03281   // Setting GTK+ widgets data
03282   buffer = g_build_filename (input->directory, input->simulator, NULL);
03283   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03284                                  (window->button_simulator), buffer);
03285   g_free (buffer);
03286   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03287                                 (size_t) input->evaluator);
03288   if (input->evaluator)
03289     {
03290       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03291       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03292                                      (window->button_evaluator), buffer);
03293       g_free (buffer);
03294     }
03295   gtk_toggle_button_set_active
03296     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
```

```
      algorithm]), TRUE);
03297   switch (input->algorithm)
03298     {
03299     case ALGORITHM_MONTE_CARLO:
03300       gtk_spin_button_set_value (window->spin_simulations,
03301                                  (gdouble) input->nsimulations);
03302     case ALGORITHM_SWEEP:
03303       gtk_spin_button_set_value (window->spin_iterations,
03304                                  (gdouble) input->niterations);
03305       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03306       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03307       break;
03308     default:
03309       gtk_spin_button_set_value (window->spin_population,
03310                                  (gdouble) input->nsimulations);
03311       gtk_spin_button_set_value (window->spin_generations,
03312                                  (gdouble) input->niterations);
03313       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03314       gtk_spin_button_set_value (window->spin_reproduction,
03315                                  input->reproduction_ratio);
03316       gtk_spin_button_set_value (window->spin_adaptation,
03317                                  input->adaptation_ratio);
03318     }
03319   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03320   g_signal_handler_block (window->button_experiment,
03321                           window->id_experiment_name);
03322   gtk_combo_box_text_remove_all (window->combo_experiment);
03323   for (i = 0; i < input->nexperiments; ++i)
03324     gtk_combo_box_text_append_text (window->combo_experiment,
03325                                     input->experiment[i]);
03326   g_signal_handler_unblock
03327     (window->button_experiment, window->id_experiment_name);
03328   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03329   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03330   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03331   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03332   gtk_combo_box_text_remove_all (window->combo_variable);
03333   for (i = 0; i < input->nvariables; ++i)
03334     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03335   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03336   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03337   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03338   window_set_variable ();
03339   window_update ();
03340
03341 #if DEBUG
03342   fprintf (stderr, "window_read: end\n");
03343 #endif
03344   return 1;
03345 }
03346
03351 void
03352 window_open ()
03353 {
03354   char *buffer, *directory, *name;
03355   GtkFileChooserDialog *dlg;
03356
03357 #if DEBUG
03358   fprintf (stderr, "window_open: start\n");
03359 #endif
03360
03361   // Saving a backup of the current input file
03362   directory = g_strdup (input->directory);
03363   name = g_strdup (input->name);
03364
03365   // Opening dialog
03366   dlg = (GtkFileChooserDialog *)
03367     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03368                                  window->window,
03369                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03370                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03371                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03372   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03373     {
03374
03375       // Traying to open the input file
03376       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
```

```
03377       if (!window_read (buffer))
03378         {
03379 #if DEBUG
03380           fprintf (stderr, "window_open: error reading input file\n");
03381 #endif
03382
03383           // Reading backup file on error
03384           buffer = g_build_filename (directory, name, NULL);
03385           if (!input_open (buffer))
03386             {
03387
03388               // Closing on backup file reading error
03389 #if DEBUG
03390               fprintf (stderr, "window_read: error reading backup file\n");
03391 #endif
03392               g_free (buffer);
03393               g_free (name);
03394               g_free (directory);
03395 #if DEBUG
03396               fprintf (stderr, "window_open: end\n");
03397 #endif
03398               gtk_main_quit ();
03399             }
03400           g_free (buffer);
03401         }
03402       else
03403         break;
03404     }
03405
03406   // Freeing and closing
03407   g_free (name);
03408   g_free (directory);
03409   gtk_widget_destroy (GTK_WIDGET (dlg));
03410 #if DEBUG
03411   fprintf (stderr, "window_open: end\n");
03412 #endif
03413 }
03414
03419 void
03420 window_new ()
03421 {
03422   unsigned int i;
03423   char *buffer, *buffer2, buffer3[64];
03424   GtkViewport *viewport;
03425   char *label_algorithm[NALGORITHMS] = {
03426     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03427   };
03428
03429   // Creating the window
03430   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
03431
03432   // Finish when closing the window
03433   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
03434
03435   // Setting the window title
03436   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03437
03438   // Creating the open button
03439   window->button_open = (GtkToolButton *) gtk_tool_button_new
03440     (gtk_image_new_from_icon_name ("document-open",
03441                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03442      gettext ("Open"));
03443   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03444
03445   // Creating the save button
03446   window->button_save = (GtkToolButton *) gtk_tool_button_new
03447     (gtk_image_new_from_icon_name ("document-save",
03448                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03449      gettext ("Save"));
03450   g_signal_connect (window->button_save, "clicked", (void (*))
    window_save,
03451                     NULL);
03452
03453   // Creating the run button
03454   window->button_run = (GtkToolButton *) gtk_tool_button_new
03455     (gtk_image_new_from_icon_name ("system-run",
03456                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03457      gettext ("Run"));
03458   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03459
03460   // Creating the options button
03461   window->button_options = (GtkToolButton *) gtk_tool_button_new
03462     (gtk_image_new_from_icon_name ("preferences-system",
03463                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03464      gettext ("Options"));
03465   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03466
```

```
03467   // Creating the help button
03468   window->button_help = (GtkToolButton *) gtk_tool_button_new
03469     (gtk_image_new_from_icon_name ("help-browser",
03470                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03471      gettext ("Help"));
03472   g_signal_connect (window->button_help, "clicked", window_help, NULL);
03473
03474   // Creating the about button
03475   window->button_about = (GtkToolButton *) gtk_tool_button_new
03476     (gtk_image_new_from_icon_name ("help-about",
03477                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03478      gettext ("About"));
03479   g_signal_connect (window->button_about, "clicked", window_about, NULL);
03480
03481   // Creating the exit button
03482   window->button_exit = (GtkToolButton *) gtk_tool_button_new
03483     (gtk_image_new_from_icon_name ("application-exit",
03484                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03485      gettext ("Exit"));
03486   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
03487
03488   // Creating the buttons bar
03489   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
03490   gtk_toolbar_insert
03491     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
03492   gtk_toolbar_insert
03493     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
03494   gtk_toolbar_insert
03495     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
03496   gtk_toolbar_insert
03497     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
03498   gtk_toolbar_insert
03499     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
03500   gtk_toolbar_insert
03501     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
03502   gtk_toolbar_insert
03503     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
03504   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
03505
03506   // Creating the simulator program label and entry
03507   window->label_simulator
03508     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
03509   window->button_simulator = (GtkFileChooserButton *)
03510     gtk_file_chooser_button_new (gettext ("Simulator program"),
03511                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03512   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
03513                                gettext ("Simulator program executable file"));
03514
03515   // Creating the evaluator program label and entry
03516   window->check_evaluator = (GtkCheckButton *)
03517     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
03518   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
03519   window->button_evaluator = (GtkFileChooserButton *)
03520     gtk_file_chooser_button_new (gettext ("Evaluator program"),
03521                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03522   gtk_widget_set_tooltip_text
03523     (GTK_WIDGET (window->button_evaluator),
03524      gettext ("Optional evaluator program executable file"));
03525
03526   // Creating the algorithm properties
03527   window->label_simulations = (GtkLabel *) gtk_label_new
03528     (gettext ("Simulations number"));
03529   window->spin_simulations
03530     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03531   window->label_iterations = (GtkLabel *)
03532     gtk_label_new (gettext ("Iterations number"));
03533   window->spin_iterations
03534     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03535   g_signal_connect
03536     (window->spin_iterations, "value-changed", window_update, NULL);
03537   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
03538   window->spin_tolerance
03539     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03540   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
03541   window->spin_bests
03542     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03543   window->label_population
03544     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
03545   window->spin_population
03546     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03547   window->label_generations
03548     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
03549   window->spin_generations
03550     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03551   window->label_mutation
03552     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
```

```
03553   window->spin_mutation
03554     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03555   window->label_reproduction
03556     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
03557   window->spin_reproduction
03558     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03559   window->label_adaptation
03560     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
03561   window->spin_adaptation
03562     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03563
03564   // Creating the array of algorithms
03565   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
03566   window->button_algorithm[0] = (GtkRadioButton *)
03567     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
03568   gtk_grid_attach (window->grid_algorithm,
03569                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
03570   g_signal_connect (window->button_algorithm[0], "clicked",
03571                     window_set_algorithm, NULL);
03572   for (i = 0; ++i < NALGORITHMS;)
03573     {
03574       window->button_algorithm[i] = (GtkRadioButton *)
03575         gtk_radio_button_new_with_mnemonic
03576         (gtk_radio_button_get_group (window->button_algorithm[0]),
03577          label_algorithm[i]);
03578       gtk_grid_attach (window->grid_algorithm,
03579                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
03580       g_signal_connect (window->button_algorithm[i], "clicked",
03581                         window_set_algorithm, NULL);
03582     }
03583   gtk_grid_attach (window->grid_algorithm,
03584                    GTK_WIDGET (window->label_simulations), 0,
03585                    NALGORITHMS, 1, 1);
03586   gtk_grid_attach (window->grid_algorithm,
03587                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
03588   gtk_grid_attach (window->grid_algorithm,
03589                    GTK_WIDGET (window->label_iterations), 0,
03590                    NALGORITHMS + 1, 1, 1);
03591   gtk_grid_attach (window->grid_algorithm,
03592                    GTK_WIDGET (window->spin_iterations), 1,
03593                    NALGORITHMS + 1, 1, 1);
03594   gtk_grid_attach (window->grid_algorithm,
03595                    GTK_WIDGET (window->label_tolerance), 0,
03596                    NALGORITHMS + 2, 1, 1);
03597   gtk_grid_attach (window->grid_algorithm,
03598                    GTK_WIDGET (window->spin_tolerance), 1,
03599                    NALGORITHMS + 2, 1, 1);
03600   gtk_grid_attach (window->grid_algorithm,
03601                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
03602   gtk_grid_attach (window->grid_algorithm,
03603                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
03604   gtk_grid_attach (window->grid_algorithm,
03605                    GTK_WIDGET (window->label_population), 0,
03606                    NALGORITHMS + 4, 1, 1);
03607   gtk_grid_attach (window->grid_algorithm,
03608                    GTK_WIDGET (window->spin_population), 1,
03609                    NALGORITHMS + 4, 1, 1);
03610   gtk_grid_attach (window->grid_algorithm,
03611                    GTK_WIDGET (window->label_generations), 0,
03612                    NALGORITHMS + 5, 1, 1);
03613   gtk_grid_attach (window->grid_algorithm,
03614                    GTK_WIDGET (window->spin_generations), 1,
03615                    NALGORITHMS + 5, 1, 1);
03616   gtk_grid_attach (window->grid_algorithm,
03617                    GTK_WIDGET (window->label_mutation), 0,
03618                    NALGORITHMS + 6, 1, 1);
03619   gtk_grid_attach (window->grid_algorithm,
03620                    GTK_WIDGET (window->spin_mutation), 1,
03621                    NALGORITHMS + 6, 1, 1);
03622   gtk_grid_attach (window->grid_algorithm,
03623                    GTK_WIDGET (window->label_reproduction), 0,
03624                    NALGORITHMS + 7, 1, 1);
03625   gtk_grid_attach (window->grid_algorithm,
03626                    GTK_WIDGET (window->spin_reproduction), 1,
03627                    NALGORITHMS + 7, 1, 1);
03628   gtk_grid_attach (window->grid_algorithm,
03629                    GTK_WIDGET (window->label_adaptation), 0,
03630                    NALGORITHMS + 8, 1, 1);
03631   gtk_grid_attach (window->grid_algorithm,
03632                    GTK_WIDGET (window->spin_adaptation), 1,
03633                    NALGORITHMS + 8, 1, 1);
03634   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
03635   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
03636                      GTK_WIDGET (window->grid_algorithm));
03637
03638   // Creating the variable widgets
03639   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
```

```
03640   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_variable),
03641                                gettext ("Variables selector"));
03642   window->id_variable = g_signal_connect
03643     (window->combo_variable, "changed", window_set_variable, NULL);
03644   window->button_add_variable
03645     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03646                                                    GTK_ICON_SIZE_BUTTON);
03647   g_signal_connect
03648     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
03649   gtk_widget_set_tooltip_text (GTK_WIDGET
03650                                (window->button_add_variable),
03651                                gettext ("Add variable"));
03652   window->button_remove_variable
03653     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03654                                                    GTK_ICON_SIZE_BUTTON);
03655   g_signal_connect
03656     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
03657   gtk_widget_set_tooltip_text (GTK_WIDGET
03658                                (window->button_remove_variable),
03659                                gettext ("Remove variable"));
03660   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
03661   window->entry_variable = (GtkEntry *) gtk_entry_new ();
03662   window->id_variable_label = g_signal_connect
03663     (window->entry_variable, "changed", window_label_variable, NULL);
03664   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
03665   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
03666     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03667   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03668   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
03669   window->scrolled_min
03670     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03671   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
03672                      GTK_WIDGET (viewport));
03673   g_signal_connect (window->spin_min, "value-changed",
03674                     window_rangemin_variable, NULL);
03675   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
03676   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
03677     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03678   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03679   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
03680   window->scrolled_max
03681     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03682   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
03683                      GTK_WIDGET (viewport));
03684   g_signal_connect (window->spin_max, "value-changed",
03685                     window_rangemax_variable, NULL);
03686   window->check_minabs = (GtkCheckButton *)
03687     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
03688   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
03689   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03690     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03691   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03692   gtk_container_add (GTK_CONTAINER (viewport),
03693                      GTK_WIDGET (window->spin_minabs));
03694   window->scrolled_minabs
03695     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03696   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
03697                      GTK_WIDGET (viewport));
03698   g_signal_connect (window->spin_minabs, "value-changed",
03699                     window_rangeminabs_variable, NULL);
03700   window->check_maxabs = (GtkCheckButton *)
03701     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
03702   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
03703   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03704     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03705   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03706   gtk_container_add (GTK_CONTAINER (viewport),
03707                      GTK_WIDGET (window->spin_maxabs));
03708   window->scrolled_maxabs
03709     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03710   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
03711                      GTK_WIDGET (viewport));
03712   g_signal_connect (window->spin_maxabs, "value-changed",
03713                     window_rangemaxabs_variable, NULL);
03714   window->label_precision
03715     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
03716   window->spin_precision = (GtkSpinButton *)
03717     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
03718   g_signal_connect (window->spin_precision, "value-changed",
03719                     window_precision_variable, NULL);
03720   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
03721   window->spin_sweeps
03722     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03723   g_signal_connect
03724     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
```

```
03725   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
03726   window->spin_bits
03727     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03728   g_signal_connect
03729     (window->spin_bits, "value-changed", window_update_variable, NULL);
03730   window->grid_variable = (GtkGrid *) gtk_grid_new ();
03731   gtk_grid_attach (window->grid_variable,
03732                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
03733   gtk_grid_attach (window->grid_variable,
03734                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
03735   gtk_grid_attach (window->grid_variable,
03736                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
03737   gtk_grid_attach (window->grid_variable,
03738                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
03739   gtk_grid_attach (window->grid_variable,
03740                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
03741   gtk_grid_attach (window->grid_variable,
03742                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
03743   gtk_grid_attach (window->grid_variable,
03744                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
03745   gtk_grid_attach (window->grid_variable,
03746                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
03747   gtk_grid_attach (window->grid_variable,
03748                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
03749   gtk_grid_attach (window->grid_variable,
03750                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
03751   gtk_grid_attach (window->grid_variable,
03752                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
03753   gtk_grid_attach (window->grid_variable,
03754                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
03755   gtk_grid_attach (window->grid_variable,
03756                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
03757   gtk_grid_attach (window->grid_variable,
03758                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
03759   gtk_grid_attach (window->grid_variable,
03760                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
03761   gtk_grid_attach (window->grid_variable,
03762                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
03763   gtk_grid_attach (window->grid_variable,
03764                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
03765   gtk_grid_attach (window->grid_variable,
03766                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
03767   gtk_grid_attach (window->grid_variable,
03768                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
03769   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
03770   gtk_container_add (GTK_CONTAINER (window->frame_variable),
03771                      GTK_WIDGET (window->grid_variable));
03772
03773   // Creating the experiment widgets
03774   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03775   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03776                                gettext ("Experiment selector"));
03777   window->id_experiment = g_signal_connect
03778     (window->combo_experiment, "changed", window_set_experiment, NULL)
    ;
03779   window->button_add_experiment
03780     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03781                                                    GTK_ICON_SIZE_BUTTON);
03782   g_signal_connect
03783     (window->button_add_experiment, "clicked",
    window_add_experiment, NULL);
03784   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03785                                gettext ("Add experiment"));
03786   window->button_remove_experiment
03787     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03788                                                    GTK_ICON_SIZE_BUTTON);
03789   g_signal_connect (window->button_remove_experiment, "clicked",
03790                     window_remove_experiment, NULL);
03791   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03792                                gettext ("Remove experiment"));
03793   window->label_experiment
03794     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
03795   window->button_experiment = (GtkFileChooserButton *)
03796     gtk_file_chooser_button_new (gettext ("Experimental data file"),
03797                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03798   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03799                                gettext ("Experimental data file"));
03800   window->id_experiment_name
03801     = g_signal_connect (window->button_experiment, "selection-changed",
03802                         window_name_experiment, NULL);
03803   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
03804   window->spin_weight
03805     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03806   gtk_widget_set_tooltip_text
03807     (GTK_WIDGET (window->spin_weight),
03808      gettext ("Weight factor to build the objective function"));
03809   g_signal_connect
```

```
03810     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
03811   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03812   gtk_grid_attach (window->grid_experiment,
03813                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03814   gtk_grid_attach (window->grid_experiment,
03815                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03816   gtk_grid_attach (window->grid_experiment,
03817                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03818   gtk_grid_attach (window->grid_experiment,
03819                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03820   gtk_grid_attach (window->grid_experiment,
03821                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03822   gtk_grid_attach (window->grid_experiment,
03823                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03824   gtk_grid_attach (window->grid_experiment,
03825                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03826   for (i = 0; i < MAX_NINPUTS; ++i)
03827     {
03828       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03829       window->check_template[i] = (GtkCheckButton *)
03830         gtk_check_button_new_with_label (buffer3);
03831       window->id_template[i]
03832         = g_signal_connect (window->check_template[i], "toggled",
03833                             window_inputs_experiment, NULL);
03834       gtk_grid_attach (window->grid_experiment,
03835                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03836       window->button_template[i] = (GtkFileChooserButton *)
03837         gtk_file_chooser_button_new (gettext ("Input template"),
03838                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03839       gtk_widget_set_tooltip_text
03840         (GTK_WIDGET (window->button_template[i]),
03841          gettext ("Experimental input template file"));
03842       window->id_input[i]
03843         = g_signal_connect_swapped (window->button_template[i],
03844                                     "selection-changed",
03845                                     (void (*)) window_template_experiment,
03846                                     (void *) (size_t) i);
03847       gtk_grid_attach (window->grid_experiment,
03848                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03849     }
03850   window->frame_experiment
03851     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03852   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
03853                      GTK_WIDGET (window->grid_experiment));
03854
03855   // Creating the grid and attaching the widgets to the grid
03856   window->grid = (GtkGrid *) gtk_grid_new ();
03857   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 6, 1);
03858   gtk_grid_attach (window->grid,
03859                    GTK_WIDGET (window->label_simulator), 0, 1, 1, 1);
03860   gtk_grid_attach (window->grid,
03861                    GTK_WIDGET (window->button_simulator), 1, 1, 1, 1);
03862   gtk_grid_attach (window->grid,
03863                    GTK_WIDGET (window->check_evaluator), 2, 1, 1, 1);
03864   gtk_grid_attach (window->grid,
03865                    GTK_WIDGET (window->button_evaluator), 3, 1, 1, 1);
03866   gtk_grid_attach (window->grid,
03867                    GTK_WIDGET (window->frame_algorithm), 0, 2, 2, 1);
03868   gtk_grid_attach (window->grid,
03869                    GTK_WIDGET (window->frame_variable), 2, 2, 2, 1);
03870   gtk_grid_attach (window->grid,
03871                    GTK_WIDGET (window->frame_experiment), 4, 2, 2, 1);
03872   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
03873
03874   // Setting the window logo
03875   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03876   gtk_window_set_icon (window->window, window->logo);
03877
03878   // Showing the window
03879   gtk_widget_show_all (GTK_WIDGET (window->window));
03880
03881   // In Windows the default scrolled size is wrong
03882 #ifdef G_OS_WIN32
03883   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03884   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03885   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03886   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03887 #endif
03888
03889   // Reading initial example
03890   input_new ();
03891   buffer2 = g_get_current_dir ();
03892   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03893   g_free (buffer2);
03894   window_read (buffer);
```

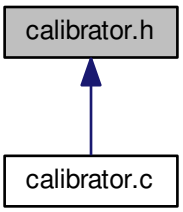
```
03895   g_free (buffer);
03896 }
03897
03898 #endif
03899
03905 int
03906 cores_number ()
03907 {
03908 #ifdef G_OS_WIN32
03909   SYSTEM_INFO sysinfo;
03910   GetSystemInfo (&sysinfo);
03911   return sysinfo.dwNumberOfProcessors;
03912 #else
03913   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03914 #endif
03915 }
03916
03926 int
03927 main (int argn, char **argc)
03928 {
03929   // Starting pseudo-random numbers generator
03930   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03931   calibrate->seed = DEFAULT_RANDOM_SEED;
03932
03933   // Allowing spaces in the XML data file
03934   xmlKeepBlanksDefault (0);
03935
03936   // Starting MPI
03937 #if HAVE_MPI
03938   MPI_Init (&argn, &argc);
03939   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03940   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03941   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03942 #else
03943   ntasks = 1;
03944 #endif
03945
03946 #if HAVE_GTK
03947
03948   // Getting threads number
03949   nthreads = cores_number ();
03950
03951   // Setting local language and international floating point numbers notation
03952   setlocale (LC_ALL, "");
03953   setlocale (LC_NUMERIC, "C");
03954   window->application_directory = g_get_current_dir ();
03955   bindtextdomain (PROGRAM_INTERFACE,
03956                   g_build_filename (window->application_directory,
03957                                     LOCALE_DIR, NULL));
03958   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03959   textdomain (PROGRAM_INTERFACE);
03960
03961   // Initing GTK+
03962   gtk_disable_setlocale ();
03963   gtk_init (&argn, &argc);
03964
03965   // Opening the main window
03966   window_new ();
03967   gtk_main ();
03968
03969   // Freeing memory
03970   gtk_widget_destroy (GTK_WIDGET (window->window));
03971   g_free (window->application_directory);
03972
03973 #else
03974
03975   // Checking syntax
03976   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03977     {
03978       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03979       return 1;
03980     }
03981
03982   // Getting threads number
03983   if (argn == 2)
03984     nthreads = cores_number ();
03985   else
03986     nthreads = atoi (argc[2]);
03987   printf ("nthreads=%u\n", nthreads);
03988
03989   // Making calibration
03990   input_new ();
03991   if (input_open (argc[argn - 1]))
03992     calibrate_new ();
03993
03994   // Freeing memory
03995   calibrate_free ();
```

```
03996
03997 #endif
03998
03999   // Closing MPI
04000 #if HAVE_MPI
04001   MPI_Finalize ();
04002 #endif
04003
04004   // Freeing memory
04005   gsl_rng_free (calibrate->rng);
04006
04007   // Closing
04008   return 0;
04009 }
```

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.

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_thread (unsigned int simulation, double value)

  Function to save the best simulations of a thread.
- void calibrate_best_sequential (unsigned int simulation, double value)

  Function to save the best simulations.
- void * calibrate_thread (ParallelData *data)

  Function to calibrate on a thread.
- void calibrate_sequential ()

  Function to calibrate sequentially.
- 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.
- 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_iterate ()

  Function to iterate the algorithm.

- void calibrate_new ()

  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 };
```

5.3.3 Function Documentation

5.3.3.1 void calibrate_best_sequential (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1330 of file calibrator.c.

```
01331 {
01332   unsigned int i, j;
01333   double e;
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_best_sequential: start\n");
01336 #endif
```

```
01337   if (calibrate->nsaveds < calibrate->nbest
01338       || value < calibrate->error_best[calibrate->nsaveds - 1])
01339     {
01340       if (calibrate->nsaveds < calibrate->nbest)
01341         ++calibrate->nsaveds;
01342       calibrate->error_best[calibrate->nsaveds - 1] = value;
01343       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01344       for (i = calibrate->nsaveds; --i;)
01345         {
01346           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01347             {
01348               j = calibrate->simulation_best[i];
01349               e = calibrate->error_best[i];
01350               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01351               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01352               calibrate->simulation_best[i - 1] = j;
01353               calibrate->error_best[i - 1] = e;
01354             }
01355           else
01356             break;
01357         }
01358     }
01359 #if DEBUG
01360   fprintf (stderr, "calibrate_best_sequential: end\n");
01361 #endif
01362 }
```

5.3.3.2 void calibrate_best_thread (unsigned int *simulation*, double *value*)

Function to save the best simulations of a thread.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1285 of file calibrator.c.

```
01286 {
01287   unsigned int i, j;
01288   double e;
01289 #if DEBUG
01290   fprintf (stderr, "calibrate_best_thread: start\n");
01291 #endif
01292   if (calibrate->nsaveds < calibrate->nbest
01293       || value < calibrate->error_best[calibrate->nsaveds - 1])
01294     {
01295       g_mutex_lock (mutex);
01296       if (calibrate->nsaveds < calibrate->nbest)
01297         ++calibrate->nsaveds;
01298       calibrate->error_best[calibrate->nsaveds - 1] = value;
01299       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01300       for (i = calibrate->nsaveds; --i;)
01301         {
01302           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01303             {
01304               j = calibrate->simulation_best[i];
01305               e = calibrate->error_best[i];
01306               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01307               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01308               calibrate->simulation_best[i - 1] = j;
01309               calibrate->error_best[i - 1] = e;
01310             }
01311           else
01312             break;
01313         }
01314       g_mutex_unlock (mutex);
01315     }
01316 #if DEBUG
01317   fprintf (stderr, "calibrate_best_thread: end\n");
01318 #endif
01319 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

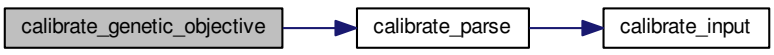
Returns

objective function value.

Definition at line 1639 of file calibrator.c.

```
01640 {
01641   unsigned int j;
01642   double objective;
01643   char buffer[64];
01644 #if DEBUG
01645   fprintf (stderr, "calibrate_genetic_objective: start\n");
01646 #endif
01647   for (j = 0; j < calibrate->nvariables; ++j)
01648     {
01649       calibrate->value[entity->id * calibrate->nvariables + j]
01650         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01651     }
01652   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01653     objective += calibrate_parse (entity->id, j);
01654   g_mutex_lock (mutex);
01655   for (j = 0; j < calibrate->nvariables; ++j)
01656     {
01657       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01658       fprintf (calibrate->file_variables, buffer,
01659                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01660     }
01661   fprintf (calibrate->file_variables, "%.14le\n", objective);
01662   g_mutex_unlock (mutex);
01663 #if DEBUG
01664   fprintf (stderr, "calibrate_genetic_objective: end\n");
01665 #endif
01666   return objective;
01667 }
```

Here is the call graph for this function:



5.3.3.4 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 1034 of file calibrator.c.

```
01035 {
01036   unsigned int i;
01037   char buffer[32], value[32], *buffer2, *buffer3, *content;
01038   FILE *file;
01039   gsize length;
01040   GRegex *regex;
01041
01042 #if DEBUG
01043   fprintf (stderr, "calibrate_input: start\n");
01044 #endif
01045
01046   // Checking the file
01047   if (!template)
01048     goto calibrate_input_end;
01049
01050   // Opening template
01051   content = g_mapped_file_get_contents (template);
01052   length = g_mapped_file_get_length (template);
01053 #if DEBUG
01054   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01055            content);
01056 #endif
01057   file = g_fopen (input, "w");
01058
01059   // Parsing template
01060   for (i = 0; i < calibrate->nvariables; ++i)
01061     {
01062 #if DEBUG
01063       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01064 #endif
01065       snprintf (buffer, 32, "@variable%u@", i + 1);
01066       regex = g_regex_new (buffer, 0, 0, NULL);
01067       if (i == 0)
01068         {
01069           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01070                                              calibrate->label[i], 0, NULL);
01071 #if DEBUG
01072           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01073 #endif
01074         }
01075       else
01076         {
01077           length = strlen (buffer3);
01078           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01079                                              calibrate->label[i], 0, NULL);
01080           g_free (buffer3);
01081         }
01082       g_regex_unref (regex);
01083       length = strlen (buffer2);
01084       snprintf (buffer, 32, "@value%u@", i + 1);
01085       regex = g_regex_new (buffer, 0, 0, NULL);
01086       snprintf (value, 32, format[calibrate->precision[i]],
01087                 calibrate->value[simulation * calibrate->
    nvariables + i]);
01088
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_input: value=%s\n", value);
01091 #endif
01092       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01093                                          0, NULL);
01094       g_free (buffer2);
01095       g_regex_unref (regex);
01096     }
01097
01098   // Saving input file
01099   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01100   g_free (buffer3);
01101   fclose (file);
01102
01103 calibrate_input_end:
01104 #if DEBUG
01105   fprintf (stderr, "calibrate_input: end\n");
01106 #endif
01107   return;
01108 }
```

5.3.3.5 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 1446 of file calibrator.c.

```
01448 {
01449   unsigned int i, j, k, s[calibrate->nbest];
01450   double e[calibrate->nbest];
01451 #if DEBUG
01452   fprintf (stderr, "calibrate_merge: start\n");
01453 #endif
01454   i = j = k = 0;
01455   do
01456     {
01457       if (i == calibrate->nsaveds)
01458         {
01459           s[k] = simulation_best[j];
01460           e[k] = error_best[j];
01461           ++j;
01462           ++k;
01463           if (j == nsaveds)
01464             break;
01465         }
01466       else if (j == nsaveds)
01467         {
01468           s[k] = calibrate->simulation_best[i];
01469           e[k] = calibrate->error_best[i];
01470           ++i;
01471           ++k;
01472           if (i == calibrate->nsaveds)
01473             break;
01474         }
01475       else if (calibrate->error_best[i] > error_best[j])
01476         {
01477           s[k] = simulation_best[j];
01478           e[k] = error_best[j];
01479           ++j;
01480           ++k;
01481         }
01482       else
01483         {
01484           s[k] = calibrate->simulation_best[i];
01485           e[k] = calibrate->error_best[i];
01486           ++i;
01487           ++k;
01488         }
01489     }
01490   while (k < calibrate->nbest);
01491   calibrate->nsaveds = k;
01492   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01493   memcpy (calibrate->error_best, e, k * sizeof (double));
01494 #if DEBUG
01495   fprintf (stderr, "calibrate_merge: end\n");
01496 #endif
01497 }
```

5.3.3.6 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 1121 of file calibrator.c.

```
01122 {
```

```
01123   unsigned int i;
01124   double e;
01125   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01126     *buffer3, *buffer4;
01127   FILE *file_result;
01128
01129 #if DEBUG
01130   fprintf (stderr, "calibrate_parse: start\n");
01131   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01132            experiment);
01133 #endif
01134
01135   // Opening input files
01136   for (i = 0; i < calibrate->ninputs; ++i)
01137     {
01138       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01139 #if DEBUG
01140       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01141 #endif
01142       calibrate_input (simulation, &input[i][0],
01143                        calibrate->file[i][experiment]);
01144     }
01145   for (; i < MAX_NINPUTS; ++i)
01146     strcpy (&input[i][0], "");
01147 #if DEBUG
01148   fprintf (stderr, "calibrate_parse: parsing end\n");
01149 #endif
01150
01151   // Performing the simulation
01152   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01153   buffer2 = g_path_get_dirname (calibrate->simulator);
01154   buffer3 = g_path_get_basename (calibrate->simulator);
01155   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01156   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01157             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01158             input[6], input[7], output);
01159   g_free (buffer4);
01160   g_free (buffer3);
01161   g_free (buffer2);
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01164 #endif
01165   system (buffer);
01166
01167   // Checking the objective value function
01168   if (calibrate->evaluator)
01169     {
01170       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01171       buffer2 = g_path_get_dirname (calibrate->evaluator);
01172       buffer3 = g_path_get_basename (calibrate->evaluator);
01173       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01174       snprintf (buffer, 512, "\"%s\" %s %s %s",
01175                 buffer4, output, calibrate->experiment[experiment], result);
01176       g_free (buffer4);
01177       g_free (buffer3);
01178       g_free (buffer2);
01179 #if DEBUG
01180       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01181 #endif
01182       system (buffer);
01183       file_result = g_fopen (result, "r");
01184       e = atof (fgets (buffer, 512, file_result));
01185       fclose (file_result);
01186     }
01187   else
01188     {
01189       strcpy (result, "");
01190       file_result = g_fopen (output, "r");
01191       e = atof (fgets (buffer, 512, file_result));
01192       fclose (file_result);
01193     }
01194
01195   // Removing files
01196 #if !DEBUG
01197   for (i = 0; i < calibrate->ninputs; ++i)
01198     {
01199       if (calibrate->file[i][0])
01200         {
01201           snprintf (buffer, 512, RM " %s", &input[i][0]);
01202           system (buffer);
01203         }
01204     }
01205   snprintf (buffer, 512, RM " %s %s", output, result);
01206   system (buffer);
01207 #endif
01208
01209 #if DEBUG
```

```
01210   fprintf (stderr, "calibrate_parse: end\n");
01211 #endif
01212
01213   // Returning the objective function
01214   return e * calibrate->weight[experiment];
01215 }
```

Here is the call graph for this function:

5.3.3.7 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 1257 of file calibrator.c.

```
01258 {
01259   unsigned int i;
01260   char buffer[64];
01261 #if DEBUG
01262   fprintf (stderr, "calibrate_save_variables: start\n");
01263 #endif
01264   for (i = 0; i < calibrate->nvariables; ++i)
01265     {
01266       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01267       fprintf (calibrate->file_variables, buffer,
01268                calibrate->value[simulation * calibrate->
      nvariables + i]);
01269     }
01270   fprintf (calibrate->file_variables, "%.14le\n", error);
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_save_variables: end\n");
01273 #endif
01274 }
```

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

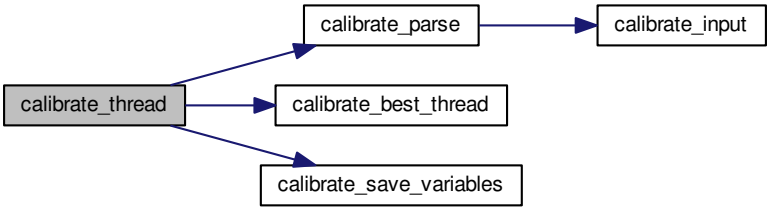
Returns

NULL

Definition at line 1372 of file calibrator.c.

```
01373 {
01374   unsigned int i, j, thread;
01375   double e;
01376 #if DEBUG
01377   fprintf (stderr, "calibrate_thread: start\n");
01378 #endif
01379   thread = data->thread;
01380 #if DEBUG
01381   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01382            calibrate->thread[thread], calibrate->thread[thread + 1]);
01383 #endif
01384   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01385     {
01386       e = 0.;
01387       for (j = 0; j < calibrate->nexperiments; ++j)
01388         e += calibrate_parse (i, j);
01389       calibrate_best_thread (i, e);
01390       g_mutex_lock (mutex);
01391       calibrate_save_variables (i, e);
01392       g_mutex_unlock (mutex);
01393 #if DEBUG
01394       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01395 #endif
01396     }
01397 #if DEBUG
01398   fprintf (stderr, "calibrate_thread: end\n");
01399 #endif
01400   g_thread_exit (NULL);
01401   return NULL;
01402 }
```

Here is the call graph for this function:



5.3.3.9 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 472 of file calibrator.c.

```
00473 {
00474   char buffer2[64];
00475   xmlDoc *doc;
00476   xmlNode *node, *child;
00477   xmlChar *buffer;
00478   char *msg;
00479   int error_code;
00480   unsigned int i;
00481
```

```
00482 #if DEBUG
00483   fprintf (stderr, "input_open: start\n");
00484 #endif
00485
00486   // Resetting input data
00487   input_new ();
00488
00489   // Parsing the input file
00490   doc = xmlParseFile (filename);
00491   if (!doc)
00492     {
00493       msg = gettext ("Unable to parse the input file");
00494       goto exit_on_error;
00495     }
00496
00497   // Getting the root node
00498   node = xmlDocGetRootElement (doc);
00499   if (xmlStrcmp (node->name, XML_CALIBRATE))
00500     {
00501       msg = gettext ("Bad root XML node");
00502       goto exit_on_error;
00503     }
00504
00505   // Opening simulator program name
00506   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00507   if (!input->simulator)
00508     {
00509       msg = gettext ("Bad simulator program");
00510       goto exit_on_error;
00511     }
00512
00513   // Opening evaluator program name
00514   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00515
00516   // Obtaining pseudo-random numbers generator seed
00517   if (!xmlHasProp (node, XML_SEED))
00518     input->seed = DEFAULT_RANDOM_SEED;
00519   else
00520     {
00521       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00522       if (error_code)
00523         {
00524           msg = gettext ("Bad pseudo-random numbers generator seed");
00525           goto exit_on_error;
00526         }
00527     }
00528
00529   // Opening algorithm
00530   buffer = xmlGetProp (node, XML_ALGORITHM);
00531   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00532     {
00533       input->algorithm = ALGORITHM_MONTE_CARLO;
00534
00535       // Obtaining simulations number
00536       input->nsimulations
00537         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00538       if (error_code)
00539         {
00540           msg = gettext ("Bad simulations number");
00541           goto exit_on_error;
00542         }
00543     }
00544   else if (!xmlStrcmp (buffer, XML_SWEEP))
00545     input->algorithm = ALGORITHM_SWEEP;
00546   else if (!xmlStrcmp (buffer, XML_GENETIC))
00547     {
00548       input->algorithm = ALGORITHM_GENETIC;
00549
00550       // Obtaining population
00551       if (xmlHasProp (node, XML_NPOPULATION))
00552         {
00553           input->nsimulations
00554             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00555           if (error_code || input->nsimulations < 3)
00556             {
00557               msg = gettext ("Invalid population number");
00558               goto exit_on_error;
00559             }
00560         }
00561       else
00562         {
00563           msg = gettext ("No population number");
00564           goto exit_on_error;
00565         }
00566
00567       // Obtaining generations
00568       if (xmlHasProp (node, XML_NGENERATIONS))
```

```
00569         {
00570           input->niterations
00571             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00572           if (error_code || !input->niterations)
00573             {
00574               msg = gettext ("Invalid generations number");
00575               goto exit_on_error;
00576             }
00577         }
00578       else
00579         {
00580           msg = gettext ("No generations number");
00581           goto exit_on_error;
00582         }
00583
00584       // Obtaining mutation probability
00585       if (xmlHasProp (node, XML_MUTATION))
00586         {
00587           input->mutation_ratio
00588             = xml_node_get_float (node, XML_MUTATION, &error_code);
00589           if (error_code || input->mutation_ratio < 0.
00590               || input->mutation_ratio >= 1.)
00591             {
00592               msg = gettext ("Invalid mutation probability");
00593               goto exit_on_error;
00594             }
00595         }
00596       else
00597         {
00598           msg = gettext ("No mutation probability");
00599           goto exit_on_error;
00600         }
00601
00602       // Obtaining reproduction probability
00603       if (xmlHasProp (node, XML_REPRODUCTION))
00604         {
00605           input->reproduction_ratio
00606             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00607           if (error_code || input->reproduction_ratio < 0.
00608               || input->reproduction_ratio >= 1.0)
00609             {
00610               msg = gettext ("Invalid reproduction probability");
00611               goto exit_on_error;
00612             }
00613         }
00614       else
00615         {
00616           msg = gettext ("No reproduction probability");
00617           goto exit_on_error;
00618         }
00619
00620       // Obtaining adaptation probability
00621       if (xmlHasProp (node, XML_ADAPTATION))
00622         {
00623           input->adaptation_ratio
00624             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00625           if (error_code || input->adaptation_ratio < 0.
00626               || input->adaptation_ratio >= 1.)
00627             {
00628               msg = gettext ("Invalid adaptation probability");
00629               goto exit_on_error;
00630             }
00631         }
00632       else
00633         {
00634           msg = gettext ("No adaptation probability");
00635           goto exit_on_error;
00636         }
00637
00638       // Checking survivals
00639       i = input->mutation_ratio * input->nsimulations;
00640       i += input->reproduction_ratio * input->
      nsimulations;
00641       i += input->adaptation_ratio * input->
      nsimulations;
00642       if (i > input->nsimulations - 2)
00643         {
00644           msg = gettext
00645             ("No enough survival entities to reproduce the population");
00646           goto exit_on_error;
00647         }
00648     }
00649   else
00650     {
00651       msg = gettext ("Unknown algorithm");
00652       goto exit_on_error;
00653     }
```

```
00654
00655   if (input->algorithm == ALGORITHM_MONTE_CARLO
00656       || input->algorithm == ALGORITHM_SWEEP)
00657     {
00658
00659       // Obtaining iterations number
00660       input->niterations
00661         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00662       if (error_code == 1)
00663         input->niterations = 1;
00664       else if (error_code)
00665         {
00666           msg = gettext ("Bad iterations number");
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining best number
00671       if (xmlHasProp (node, XML_NBEST))
00672         {
00673           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00674           if (error_code || !input->nbest)
00675             {
00676               msg = gettext ("Invalid best number");
00677               goto exit_on_error;
00678             }
00679         }
00680       else
00681         input->nbest = 1;
00682
00683       // Obtaining tolerance
00684       if (xmlHasProp (node, XML_TOLERANCE))
00685         {
00686           input->tolerance
00687             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00688           if (error_code || input->tolerance < 0.)
00689             {
00690               msg = gettext ("Invalid tolerance");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         input->tolerance = 0.;
00696     }
00697
00698   // Reading the experimental data
00699   for (child = node->children; child; child = child->next)
00700     {
00701       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00702         break;
00703 #if DEBUG
00704       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00705 #endif
00706       if (xmlHasProp (child, XML_NAME))
00707         {
00708           input->experiment
00709             = g_realloc (input->experiment,
00710                          (1 + input->nexperiments) * sizeof (char *));
00711           input->experiment[input->nexperiments]
00712             = (char *) xmlGetProp (child, XML_NAME);
00713         }
00714       else
00715         {
00716           snprintf (buffer2, 64, "%s %u: %s",
00717                     gettext ("Experiment"),
00718                     input->nexperiments + 1, gettext ("no data file name"));
00719           msg = buffer2;
00720           goto exit_on_error;
00721         }
00722 #if DEBUG
00723       fprintf (stderr, "input_open: experiment=%s\n",
00724                input->experiment[input->nexperiments]);
00725 #endif
00726       input->weight = g_realloc (input->weight,
00727                                  (1 + input->nexperiments) * sizeof (double));
00728       if (xmlHasProp (child, XML_WEIGHT))
00729         {
00730           input->weight[input->nexperiments]
00731             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00732           if (error_code)
00733             {
00734               snprintf (buffer2, 64, "%s %u: %s",
00735                         gettext ("Experiment"),
00736                         input->nexperiments + 1, gettext ("bad weight"));
00737               msg = buffer2;
00738               goto exit_on_error;
00739             }
```

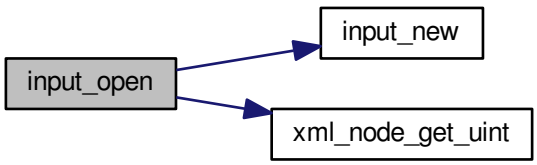
```
00740         }
00741       else
00742         input->weight[input->nexperiments] = 1.;
00743 #if DEBUG
00744       fprintf (stderr, "input_open: weight=%lg\n",
00745                input->weight[input->nexperiments]);
00746 #endif
00747       if (!input->nexperiments)
00748         input->ninputs = 0;
00749 #if DEBUG
00750       fprintf (stderr, "input_open: template[0]\n");
00751 #endif
00752       if (xmlHasProp (child, XML_TEMPLATE1))
00753         {
00754           input->template[0]
00755             = (char **) g_realloc (input->template[0],
00756                                    (1 + input->nexperiments) * sizeof (char *));
00757           input->template[0][input->nexperiments]
00758             = (char *) xmlGetProp (child, template[0]);
00759 #if DEBUG
00760           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00761                    input->nexperiments,
00762                    input->template[0][input->nexperiments]);
00763 #endif
00764           if (!input->nexperiments)
00765             ++input->ninputs;
00766 #if DEBUG
00767           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00768 #endif
00769         }
00770       else
00771         {
00772           snprintf (buffer2, 64, "%s %u: %s",
00773                     gettext ("Experiment"),
00774                     input->nexperiments + 1, gettext ("no template"));
00775           msg = buffer2;
00776           goto exit_on_error;
00777         }
00778       for (i = 1; i < MAX_NINPUTS; ++i)
00779         {
00780 #if DEBUG
00781           fprintf (stderr, "input_open: template%u\n", i + 1);
00782 #endif
00783           if (xmlHasProp (child, template[i]))
00784             {
00785               if (input->nexperiments && input->ninputs <= i)
00786                 {
00787                   snprintf (buffer2, 64, "%s %u: %s",
00788                             gettext ("Experiment"),
00789                             input->nexperiments + 1,
00790                             gettext ("bad templates number"));
00791                   msg = buffer2;
00792                   goto exit_on_error;
00793                 }
00794               input->template[i] = (char **)
00795                 g_realloc (input->template[i],
00796                            (1 + input->nexperiments) * sizeof (char *));
00797               input->template[i][input->nexperiments]
00798                 = (char *) xmlGetProp (child, template[i]);
00799 #if DEBUG
00800               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00801                        input->nexperiments, i + 1,
00802                        input->template[i][input->nexperiments]);
00803 #endif
00804               if (!input->nexperiments)
00805                 ++input->ninputs;
00806 #if DEBUG
00807               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00808 #endif
00809             }
00810           else if (input->nexperiments && input->ninputs >= i)
00811             {
00812               snprintf (buffer2, 64, "%s %u: %s%u",
00813                         gettext ("Experiment"),
00814                         input->nexperiments + 1,
00815                         gettext ("no template"), i + 1);
00816               msg = buffer2;
00817               goto exit_on_error;
00818             }
00819           else
00820             break;
00821         }
00822       ++input->nexperiments;
00823 #if DEBUG
00824       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00825 #endif
00826     }
```

```
00827   if (!input->nexperiments)
00828     {
00829       msg = gettext ("No calibration experiments");
00830       goto exit_on_error;
00831     }
00832
00833   // Reading the variables data
00834   for (; child; child = child->next)
00835     {
00836       if (xmlStrcmp (child->name, XML_VARIABLE))
00837         {
00838           snprintf (buffer2, 64, "%s %u: %s",
00839                     gettext ("Variable"),
00840                     input->nvariables + 1, gettext ("bad XML node"));
00841           msg = buffer2;
00842           goto exit_on_error;
00843         }
00844       if (xmlHasProp (child, XML_NAME))
00845         {
00846           input->label = g_realloc
00847             (input->label, (1 + input->nvariables) * sizeof (char *));
00848           input->label[input->nvariables]
00849             = (char *) xmlGetProp (child, XML_NAME);
00850         }
00851       else
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("no name"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_MINIMUM))
00860         {
00861           input->rangemin = g_realloc
00862             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00863           input->rangeminabs = g_realloc
00864             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00865           input->rangemin[input->nvariables]
00866             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00867           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00868             {
00869               input->rangeminabs[input->nvariables]
00870                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00871             }
00872           else
00873             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00874           if (input->rangemin[input->nvariables]
00875               < input->rangeminabs[input->nvariables])
00876             {
00877               snprintf (buffer2, 64, "%s %u: %s",
00878                         gettext ("Variable"),
00879                         input->nvariables + 1,
00880                         gettext ("minimum range not allowed"));
00881               msg = buffer2;
00882               goto exit_on_error;
00883             }
00884         }
00885       else
00886         {
00887           snprintf (buffer2, 64, "%s %u: %s",
00888                     gettext ("Variable"),
00889                     input->nvariables + 1, gettext ("no minimum range"));
00890           msg = buffer2;
00891           goto exit_on_error;
00892         }
00893       if (xmlHasProp (child, XML_MAXIMUM))
00894         {
00895           input->rangemax = g_realloc
00896             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00897           input->rangemaxabs = g_realloc
00898             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00899           input->rangemax[input->nvariables]
00900             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00901           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00902             input->rangemaxabs[input->nvariables]
00903               = xml_node_get_float (child,
    XML_ABSOLUTE_MAXIMUM, &error_code);
00904           else
00905             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00906           if (input->rangemax[input->nvariables]
00907               > input->rangemaxabs[input->nvariables])
00908             {
00909               snprintf (buffer2, 64, "%s %u: %s",
00910                         gettext ("Variable"),
00911                         input->nvariables + 1,
```

```
00912                         gettext ("maximum range not allowed"));
00913               msg = buffer2;
00914               goto exit_on_error;
00915             }
00916         }
00917       else
00918         {
00919           snprintf (buffer2, 64, "%s %u: %s",
00920                     gettext ("Variable"),
00921                     input->nvariables + 1, gettext ("no maximum range"));
00922           msg = buffer2;
00923           goto exit_on_error;
00924         }
00925       if (input->rangemax[input->nvariables]
00926           < input->rangemin[input->nvariables])
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("bad range"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       input->precision = g_realloc
00935         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00936       if (xmlHasProp (child, XML_PRECISION))
00937         input->precision[input->nvariables]
00938           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00939       else
00940         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00941       if (input->algorithm == ALGORITHM_SWEEP)
00942         {
00943           if (xmlHasProp (child, XML_NSWEEPS))
00944             {
00945               input->nsweeps = (unsigned int *)
00946                 g_realloc (input->nsweeps,
00947                            (1 + input->nvariables) * sizeof (unsigned int));
00948               input->nsweeps[input->nvariables]
00949                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00950             }
00951           else
00952             {
00953               snprintf (buffer2, 64, "%s %u: %s",
00954                         gettext ("Variable"),
00955                         input->nvariables + 1, gettext ("no sweeps number"));
00956               msg = buffer2;
00957               goto exit_on_error;
00958             }
00959 #if DEBUG
00960           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00961                    input->nsweeps[input->nvariables],
    input->nsimulations);
00962 #endif
00963         }
00964       if (input->algorithm == ALGORITHM_GENETIC)
00965         {
00966           // Obtaining bits representing each variable
00967           if (xmlHasProp (child, XML_NBITS))
00968             {
00969               input->nbits = (unsigned int *)
00970                 g_realloc (input->nbits,
00971                            (1 + input->nvariables) * sizeof (unsigned int));
00972               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00973               if (error_code || !i)
00974                 {
00975                   snprintf (buffer2, 64, "%s %u: %s",
00976                             gettext ("Variable"),
00977                             input->nvariables + 1,
00978                             gettext ("invalid bits number"));
00979                   msg = buffer2;
00980                   goto exit_on_error;
00981                 }
00982               input->nbits[input->nvariables] = i;
00983             }
00984           else
00985             {
00986               snprintf (buffer2, 64, "%s %u: %s",
00987                         gettext ("Variable"),
00988                         input->nvariables + 1, gettext ("no bits number"));
00989               msg = buffer2;
00990               goto exit_on_error;
00991             }
00992         }
00993       ++input->nvariables;
00994     }
00995   if (!input->nvariables)
00996     {
```

```
00997       msg = gettext ("No calibration variables");
00998       goto exit_on_error;
00999     }
01000
01001   // Getting the working directory
01002   input->directory = g_path_get_dirname (filename);
01003   input->name = g_path_get_basename (filename);
01004
01005   // Closing the XML document
01006   xmlFreeDoc (doc);
01007
01008 #if DEBUG
01009   fprintf (stderr, "input_open: end\n");
01010 #endif
01011   return 1;
01012
01013 exit_on_error:
01014   show_error (msg);
01015   input_free ();
01016 #if DEBUG
01017   fprintf (stderr, "input_open: end\n");
01018 #endif
01019   return 0;
01020 }
```

Here is the call graph for this function:



5.3.3.10 void show_error (char * *msg*)

Function to show a dialog with an error message.

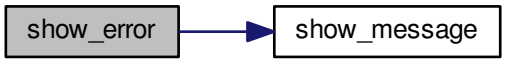
Parameters

msg	Error message.

Definition at line 246 of file calibrator.c.

```
00247 {
00248   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00249 }
```

Here is the call graph for this function:



5.3.3.11 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 216 of file calibrator.c.

```
00217 {
00218 #if HAVE_GTK
00219   GtkMessageDialog *dlg;
00220
00221   // Creating the dialog
00222   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00223     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00224
00225   // Setting the dialog title
00226   gtk_window_set_title (GTK_WINDOW (dlg), title);
00227
00228   // Showing the dialog and waiting response
00229   gtk_dialog_run (GTK_DIALOG (dlg));
00230
00231   // Closing and freeing memory
00232   gtk_widget_destroy (GTK_WIDGET (dlg));
00233
00234 #else
00235   printf ("%s: %s\n", title, msg);
00236 #endif
00237 }
```

5.3.3.12 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 325 of file calibrator.c.

```
00326 {
00327   double x = 0.;
00328   xmlChar *buffer;
00329   buffer = xmlGetProp (node, prop);
00330   if (!buffer)
00331     *error_code = 1;
00332   else
00333     {
00334       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00335         *error_code = 2;
00336       else
00337         *error_code = 0;
00338       xmlFree (buffer);
00339     }
00340   return x;
00341 }
```

5.3.3.13 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 263 of file calibrator.c.

```
00264 {
00265   int i = 0;
00266   xmlChar *buffer;
00267   buffer = xmlGetProp (node, prop);
00268   if (!buffer)
00269     *error_code = 1;
00270   else
00271     {
00272       if (sscanf ((char *) buffer, "%d", &i) != 1)
00273         *error_code = 2;
00274       else
00275         *error_code = 0;
00276       xmlFree (buffer);
00277     }
00278   return i;
00279 }
```

5.3.3.14 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 294 of file calibrator.c.

```
00295 {
00296   unsigned int i = 0;
00297   xmlChar *buffer;
00298   buffer = xmlGetProp (node, prop);
00299   if (!buffer)
00300     *error_code = 1;
00301   else
00302     {
00303       if (sscanf ((char *) buffer, "%u", &i) != 1)
00304         *error_code = 2;
00305       else
00306         *error_code = 0;
00307       xmlFree (buffer);
00308     }
00309   return i;
00310 }
```

5.3.3.15 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 392 of file calibrator.c.

```
00393 {
00394   xmlChar buffer[64];
00395   snprintf ((char *) buffer, 64, "%.14lg", value);
00396   xmlSetProp (node, prop, buffer);
00397 }
```

5.3.3.16 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 354 of file calibrator.c.

```
00355 {
00356   xmlChar buffer[64];
00357   snprintf ((char *) buffer, 64, "%d", value);
00358   xmlSetProp (node, prop, buffer);
00359 }
```

5.3.3.17 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 373 of file calibrator.c.

```
00374 {
00375   xmlChar buffer[64];
00376   snprintf ((char *) buffer, 64, "%u", value);
00377   xmlSetProp (node, prop, buffer);
00378 }
```

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
```

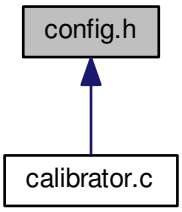
```
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' 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 typedef struct
00055 {
00056   char *simulator;
00057   char *evaluator;
00059   char **experiment;
00060   char **template[MAX_NINPUTS];
00061   char **label;
00062   char *directory;
00063   char *name;
00064   double *rangemin;
00065   double *rangemax;
00066   double *rangeminabs;
00067   double *rangemaxabs;
00068   double *weight;
00069   double tolerance;
00070   double mutation_ratio;
00071   double reproduction_ratio;
00072   double adaptation_ratio;
00073   unsigned long int seed;
00075   unsigned int nvariables;
00076   unsigned int nexperiments;
00077   unsigned int ninputs;
00078   unsigned int nsimulations;
00079   unsigned int algorithm;
00080   unsigned int *precision;
00081   unsigned int *nsweeps;
00082   unsigned int *nbits;
00084   unsigned int niterations;
00085   unsigned int nbest;
00086 } Input;
00087
00092 typedef struct
00093 {
00094   char *simulator;
00095   char *evaluator;
00097   char **experiment;
00098   char **template[MAX_NINPUTS];
00099   char **label;
00100   unsigned int nvariables;
00101   unsigned int nexperiments;
00102   unsigned int ninputs;
00103   unsigned int nsimulations;
00104   unsigned int algorithm;
00105   unsigned int *precision;
00106   unsigned int *nsweeps;
00107   unsigned int nstart;
00108   unsigned int nend;
00109   unsigned int *thread;
00111   unsigned int niterations;
00112   unsigned int nbest;
00113   unsigned int nsaveds;
00114   unsigned int *simulation_best;
00115   unsigned long int seed;
00117   double *value;
00118   double *rangemin;
00119   double *rangemax;
00120   double *rangeminabs;
00121   double *rangemaxabs;
00122   double *error_best;
00123   double *weight;
00124   double *value_old;
```

```
00126   double *error_old;
00128   double tolerance;
00129   double mutation_ratio;
00130   double reproduction_ratio;
00131   double adaptation_ratio;
00132   FILE *file_result;
00133   FILE *file_variables;
00134   gsl_rng *rng;
00135   GMappedFile **file[MAX_NINPUTS];
00136   GeneticVariable *genetic_variable;
00138 #if HAVE_MPI
00139   int mpi_rank;
00140 #endif
00141 } Calibrate;
00142
00147 typedef struct
00148 {
00149   unsigned int thread;
00150 } ParallelData;
00151
00152 // Public functions
00153 void show_message (char *title, char *msg, int type);
00154 void show_error (char *msg);
00155 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00156 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00157                                 int *error_code);
00158 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00159                            int *error_code);
00160 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00161 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00162                         unsigned int value);
00163 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00164 void input_new ();
00165 void input_free ();
00166 int input_open (char *filename);
00167 void calibrate_input (unsigned int simulation, char *input,
00168                       GMappedFile * template);
00169 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00170 void calibrate_print ();
00171 void calibrate_save_variables (unsigned int simulation, double error);
00172 void calibrate_best_thread (unsigned int simulation, double value);
00173 void calibrate_best_sequential (unsigned int simulation, double value);
00174 void *calibrate_thread (ParallelData * data);
00175 void calibrate_sequential ();
00176 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00177                       double *error_best);
00178 #if HAVE_MPI
00179 void calibrate_synchronise ();
00180 #endif
00181 void calibrate_sweep ();
00182 void calibrate_MonteCarlo ();
00183 double calibrate_genetic_objective (Entity * entity);
00184 void calibrate_genetic ();
00185 void calibrate_save_old ();
00186 void calibrate_merge_old ();
00187 void calibrate_refine ();
00188 void calibrate_iterate ();
00189 void calibrate_new ();
00190
00191 #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 algorithms.
- #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 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_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_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_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_NSWEEPS (const xmlChar*)"nsweeps"

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

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

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

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

  seed 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_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

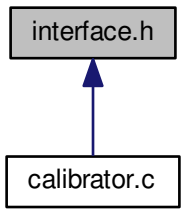
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary 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 NPRECISIONS 15
00046
00047 // Default choices
00048
00049 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00050 #define DEFAULT_RANDOM_SEED 7007
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "calibrator"
00056
00057 // XML labels
00058
00059 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00060 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00062 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00064 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00066 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00068 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00070 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00072 #define XML_GENETIC (const xmlChar*)"genetic"
00074 #define XML_MINIMUM (const xmlChar*)"minimum"
00075 #define XML_MAXIMUM (const xmlChar*)"maximum"
```

```
00076 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00077 #define XML_MUTATION (const xmlChar*)"mutation"
00079 #define XML_NAME (const xmlChar*)"name"
00080 #define XML_NBEST (const xmlChar*)"nbest"
00081 #define XML_NBITS (const xmlChar*)"nbits"
00082 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00083 #define XML_NITERATIONS (const xmlChar*)"niterations"
00085 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00087 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00089 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00091 #define XML_PRECISION (const xmlChar*)"precision"
00092 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00094 #define XML_SIMULATOR (const xmlChar*)"simulator"
00096 #define XML_SEED (const xmlChar*)"seed"
00098 #define XML_SWEEP (const xmlChar*)"sweep"
00099 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00100 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00102 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00104 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00106 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00108 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00110 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00112 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00114 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00116 #define XML_VARIABLE (const xmlChar*)"variable"
00118 #define XML_WEIGHT (const xmlChar*)"weight"
00119
00120 #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_save ()

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

  Function to run a calibration.
- void window_help ()

  Function to show a help dialog.
- int window_get_algorithm ()

  Function to get the algorithm number.
- 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 3906 of file calibrator.c.

```
03907 {
03908 #ifdef G_OS_WIN32
03909   SYSTEM_INFO sysinfo;
03910   GetSystemInfo (&sysinfo);
03911   return sysinfo.dwNumberOfProcessors;
03912 #else
03913   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03914 #endif
03915 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

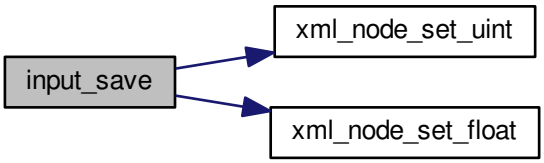
filename	Input file name.

Definition at line 2144 of file calibrator.c.

```
02145 {
02146   unsigned int i, j;
02147   char *buffer;
02148   xmlDoc *doc;
02149   xmlNode *node, *child;
02150   GFile *file, *file2;
02151
02152   // Getting the input file directory
02153   input->name = g_path_get_basename (filename);
02154   input->directory = g_path_get_dirname (filename);
02155   file = g_file_new_for_path (input->directory);
02156
02157   // Opening the input file
02158   doc = xmlNewDoc ((const xmlChar *) "1.0");
02159
02160   // Setting root XML node
02161   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02162   xmlDocSetRootElement (doc, node);
02163
02164   // Adding properties to the root XML node
02165   file2 = g_file_new_for_path (input->simulator);
02166   buffer = g_file_get_relative_path (file, file2);
02167   g_object_unref (file2);
02168   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02169   g_free (buffer);
02170   if (input->evaluator)
02171     {
02172       file2 = g_file_new_for_path (input->evaluator);
02173       buffer = g_file_get_relative_path (file, file2);
02174       g_object_unref (file2);
02175       if (xmlStrlen ((xmlChar *) buffer))
02176         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02177       g_free (buffer);
02178     }
02179   if (input->seed != DEFAULT_RANDOM_SEED)
02180     xml_node_set_uint (node, XML_SEED, input->seed);
02181
02182   // Setting the algorithm
02183   buffer = (char *) g_malloc (64);
02184   switch (input->algorithm)
02185     {
02186     case ALGORITHM_MONTE_CARLO:
02187       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02188       snprintf (buffer, 64, "%u", input->nsimulations);
02189       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%u", input->niterations);
02191       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%.3lg", input->tolerance);
02193       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02194       snprintf (buffer, 64, "%u", input->nbest);
02195       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02196       break;
02197     case ALGORITHM_SWEEP:
02198       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02199       snprintf (buffer, 64, "%u", input->niterations);
02200       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%.3lg", input->tolerance);
02202       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02203       snprintf (buffer, 64, "%u", input->nbest);
02204       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02205       break;
02206     default:
02207       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02208       snprintf (buffer, 64, "%u", input->nsimulations);
02209       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%u", input->niterations);
02211       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02213       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02215       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02216       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02217       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02218       break;
02219     }
02220   g_free (buffer);
02221
02222   // Setting the experimental data
02223   for (i = 0; i < input->nexperiments; ++i)
02224     {
```

```
02225       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02226       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02227       if (input->weight[i] != 1.)
02228         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02229       for (j = 0; j < input->ninputs; ++j)
02230         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02231     }
02232
02233   // Setting the variables data
02234   for (i = 0; i < input->nvariables; ++i)
02235     {
02236       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02237       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02238       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02239       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02240         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02241       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02242       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02243         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02244       if (input->precision[i] != DEFAULT_PRECISION)
02245         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02246       if (input->algorithm == ALGORITHM_SWEEP)
02247         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02248       else if (input->algorithm == ALGORITHM_GENETIC)
02249         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02250     }
02251
02252   // Saving the XML file
02253   xmlSaveFormatFile (filename, doc, 1);
02254
02255   // Freeing memory
02256   xmlFreeDoc (doc);
02257 }
```

Here is the call graph for this function:



5.7.2.3 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2517 of file calibrator.c.

```
02518 {
02519   unsigned int i;
02520   for (i = 0; i < NALGORITHMS; ++i)
```

```
02521     if (gtk_toggle_button_get_active
02522         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02523       break;
02524   return i;
02525 }
```

5.7.2.4 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

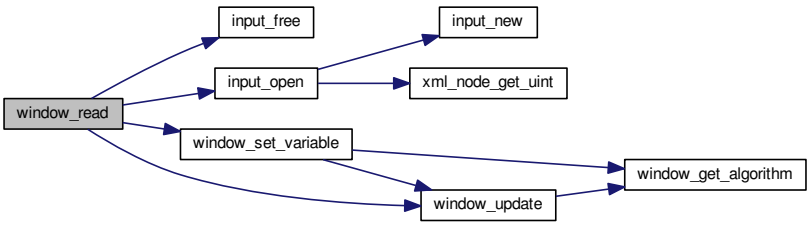
1 on succes, 0 on error.

Definition at line 3268 of file calibrator.c.

```
03269 {
03270   unsigned int i;
03271   char *buffer;
03272 #if DEBUG
03273   fprintf (stderr, "window_read: start\n");
03274 #endif
03275
03276   // Reading new input file
03277   input_free ();
03278   if (!input_open (filename))
03279     return 0;
03280
03281   // Setting GTK+ widgets data
03282   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03283   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03284                                  (window->button_simulator), buffer);
03285   g_free (buffer);
03286   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03287                                 (size_t) input->evaluator);
03288   if (input->evaluator)
03289     {
03290       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03291       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03292                                      (window->button_evaluator), buffer);
03293       g_free (buffer);
03294     }
03295   gtk_toggle_button_set_active
03296     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03297   switch (input->algorithm)
03298     {
03299     case ALGORITHM_MONTE_CARLO:
03300       gtk_spin_button_set_value (window->spin_simulations,
03301                                  (gdouble) input->nsimulations);
03302     case ALGORITHM_SWEEP:
03303       gtk_spin_button_set_value (window->spin_iterations,
03304                                  (gdouble) input->niterations);
03305       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03306       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03307       break;
03308     default:
03309       gtk_spin_button_set_value (window->spin_population,
03310                                  (gdouble) input->nsimulations);
03311       gtk_spin_button_set_value (window->spin_generations,
03312                                  (gdouble) input->niterations);
03313       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03314       gtk_spin_button_set_value (window->spin_reproduction,
03315                                  input->reproduction_ratio);
03316       gtk_spin_button_set_value (window->spin_adaptation,
03317                                  input->adaptation_ratio);
03318     }
03319   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
```

```
03320   g_signal_handler_block (window->button_experiment,
03321                           window->id_experiment_name);
03322   gtk_combo_box_text_remove_all (window->combo_experiment);
03323   for (i = 0; i < input->nexperiments; ++i)
03324     gtk_combo_box_text_append_text (window->combo_experiment,
03325                                     input->experiment[i]);
03326   g_signal_handler_unblock
03327     (window->button_experiment, window->
      id_experiment_name);
03328   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03329   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03330   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03331   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03332   gtk_combo_box_text_remove_all (window->combo_variable);
03333   for (i = 0; i < input->nvariables; ++i)
03334     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03335   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03336   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03337   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03338   window_set_variable ();
03339   window_update ();
03340
03341 #if DEBUG
03342   fprintf (stderr, "window_read: end\n");
03343 #endif
03344   return 1;
03345 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

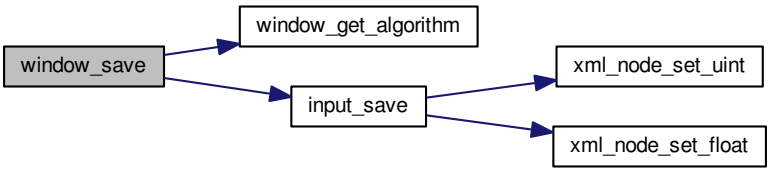
1 on OK, 0 on Cancel.

Definition at line 2332 of file calibrator.c.

```
02333 {
02334   char *buffer;
02335   GtkFileChooserDialog *dlg;
02336
02337 #if DEBUG
02338   fprintf (stderr, "window_save: start\n");
02339 #endif
02340
02341   // Opening the saving dialog
02342   dlg = (GtkFileChooserDialog *)
02343     gtk_file_chooser_dialog_new (gettext ("Save file"),
02344                                  window->window,
```

```
02345                                 GTK_FILE_CHOOSER_ACTION_SAVE,
02346                                 gettext ("_Cancel"),
02347                                 GTK_RESPONSE_CANCEL,
02348                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02349   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02350   buffer = g_build_filename (input->directory, input->name, NULL);
02351   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02352   g_free (buffer);
02353
02354   // If OK response then saving
02355   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02356     {
02357
02358       // Adding properties to the root XML node
02359       input->simulator = gtk_file_chooser_get_filename
02360         (GTK_FILE_CHOOSER (window->button_simulator));
02361       if (gtk_toggle_button_get_active
02362           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02363         input->evaluator = gtk_file_chooser_get_filename
02364           (GTK_FILE_CHOOSER (window->button_evaluator));
02365       else
02366         input->evaluator = NULL;
02367
02368       // Setting the algorithm
02369       switch (window_get_algorithm ())
02370         {
02371         case ALGORITHM_MONTE_CARLO:
02372           input->algorithm = ALGORITHM_MONTE_CARLO;
02373           input->nsimulations
02374             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02375           input->niterations
02376             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02377           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02378           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02379           break;
02380         case ALGORITHM_SWEEP:
02381           input->algorithm = ALGORITHM_SWEEP;
02382           input->niterations
02383             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02384           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02385           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02386           break;
02387         default:
02388           input->algorithm = ALGORITHM_GENETIC;
02389           input->nsimulations
02390             = gtk_spin_button_get_value_as_int (window->spin_population);
02391           input->niterations
02392             = gtk_spin_button_get_value_as_int (window->spin_generations);
02393           input->mutation_ratio
02394             = gtk_spin_button_get_value (window->spin_mutation);
02395           input->reproduction_ratio
02396             = gtk_spin_button_get_value (window->spin_reproduction);
02397           input->adaptation_ratio
02398             = gtk_spin_button_get_value (window->spin_adaptation);
02399           break;
02400         }
02401
02402       // Saving the XML file
02403       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02404       input_save (buffer);
02405
02406       // Closing and freeing memory
02407       g_free (buffer);
02408       gtk_widget_destroy (GTK_WIDGET (dlg));
02409 #if DEBUG
02410       fprintf (stderr, "window_save: end\n");
02411 #endif
02412       return 1;
02413     }
02414
02415   // Closing and freeing memory
02416   gtk_widget_destroy (GTK_WIDGET (dlg));
02417 #if DEBUG
02418   fprintf (stderr, "window_save: end\n");
02419 #endif
02420   return 0;
02421 }
```

Here is the call graph for this function:



5.7.2.6 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 2904 of file calibrator.c.

```
02905 {
02906   unsigned int i, j;
02907   char *buffer;
02908   GFile *file1, *file2;
02909 #if DEBUG
02910   fprintf (stderr, "window_template_experiment: start\n");
02911 #endif
02912   i = (size_t) data;
02913   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02914   file1
02915     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02916   file2 = g_file_new_for_path (input->directory);
02917   buffer = g_file_get_relative_path (file2, file1);
02918   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02919   g_free (buffer);
02920   g_object_unref (file2);
02921   g_object_unref (file1);
02922 #if DEBUG
02923   fprintf (stderr, "window_template_experiment: end\n");
02924 #endif
02925 }
```

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_processors;
00079   GtkSpinButton *spin_processors;
00080   GtkLabel *label_seed;
00082   GtkSpinButton *spin_seed;
00084 } Options;
00085
00090 typedef struct
00091 {
00092   GtkDialog *dialog;
00093   GtkLabel *label;
00094 } Running;
00095
00100 typedef struct
00101 {
00102   GtkWindow *window;
00103   GtkGrid *grid;
00104   GtkToolbar *bar_buttons;
00105   GtkToolButton *button_open;
00106   GtkToolButton *button_save;
00107   GtkToolButton *button_run;
00108   GtkToolButton *button_options;
00109   GtkToolButton *button_help;
00110   GtkToolButton *button_about;
00111   GtkToolButton *button_exit;
00112   GtkLabel *label_simulator;
00113   GtkFileChooserButton *button_simulator;
00115   GtkCheckButton *check_evaluator;
00116   GtkFileChooserButton *button_evaluator;
00118   GtkFrame *frame_algorithm;
00119   GtkGrid *grid_algorithm;
00120   GtkRadioButton *button_algorithm[NALGORITHMS];
00122   GtkLabel *label_simulations;
00123   GtkSpinButton *spin_simulations;
00125   GtkLabel *label_iterations;
00126   GtkSpinButton *spin_iterations;
00128   GtkLabel *label_tolerance;
00129   GtkSpinButton *spin_tolerance;
00130   GtkLabel *label_bests;
00131   GtkSpinButton *spin_bests;
00132   GtkLabel *label_population;
00133   GtkSpinButton *spin_population;
00135   GtkLabel *label_generations;
00136   GtkSpinButton *spin_generations;
00138   GtkLabel *label_mutation;
00139   GtkSpinButton *spin_mutation;
00140   GtkLabel *label_reproduction;
00141   GtkSpinButton *spin_reproduction;
00143   GtkLabel *label_adaptation;
00144   GtkSpinButton *spin_adaptation;
00146   GtkFrame *frame_variable;
00147   GtkGrid *grid_variable;
00148   GtkComboBoxText *combo_variable;
```

```
00150   GtkButton *button_add_variable;
00151   GtkButton *button_remove_variable;
00152   GtkLabel *label_variable;
00153   GtkEntry *entry_variable;
00154   GtkLabel *label_min;
00155   GtkSpinButton *spin_min;
00156   GtkScrolledWindow *scrolled_min;
00157   GtkLabel *label_max;
00158   GtkSpinButton *spin_max;
00159   GtkScrolledWindow *scrolled_max;
00160   GtkCheckButton *check_minabs;
00161   GtkSpinButton *spin_minabs;
00162   GtkScrolledWindow *scrolled_minabs;
00163   GtkCheckButton *check_maxabs;
00164   GtkSpinButton *spin_maxabs;
00165   GtkScrolledWindow *scrolled_maxabs;
00166   GtkLabel *label_precision;
00167   GtkSpinButton *spin_precision;
00168   GtkLabel *label_sweeps;
00169   GtkSpinButton *spin_sweeps;
00170   GtkLabel *label_bits;
00171   GtkSpinButton *spin_bits;
00172   GtkFrame *frame_experiment;
00173   GtkGrid *grid_experiment;
00174   GtkComboBoxText *combo_experiment;
00175   GtkButton *button_add_experiment;
00176   GtkButton *button_remove_experiment;
00177   GtkLabel *label_experiment;
00178   GtkFileChooserButton *button_experiment;
00180   GtkLabel *label_weight;
00181   GtkSpinButton *spin_weight;
00182   GtkCheckButton *check_template[MAX_NINPUTS];
00184   GtkFileChooserButton *button_template[MAX_NINPUTS];
00186   GdkPixbuf *logo;
00187   Experiment *experiment;
00188   Variable *variable;
00189   char *application_directory;
00190   gulong id_experiment;
00191   gulong id_experiment_name;
00192   gulong id_variable;
00193   gulong id_variable_label;
00194   gulong id_template[MAX_NINPUTS];
00196   gulong id_input[MAX_NINPUTS];
00198   unsigned int nexperiments;
00199   unsigned int nvariables;
00200 } Window;
00201
00202 // Public functions
00203 void input_save (char *filename);
00204 void options_new ();
00205 void running_new ();
00206 int window_save ();
00207 void window_run ();
00208 void window_help ();
00209 int window_get_algorithm ();
00210 void window_update ();
00211 void window_set_algorithm ();
00212 void window_set_experiment ();
00213 void window_remove_experiment ();
00214 void window_add_experiment ();
00215 void window_name_experiment ();
00216 void window_weight_experiment ();
00217 void window_inputs_experiment ();
00218 void window_template_experiment (void *data);
00219 void window_set_variable ();
00220 void window_remove_variable ();
00221 void window_add_variable ();
00222 void window_label_variable ();
00223 void window_precision_variable ();
00224 void window_rangemin_variable ();
00225 void window_rangemax_variable ();
00226 void window_rangeminabs_variable ();
00227 void window_rangemaxabs_variable ();
00228 void window_update_variable ();
00229 int window_read (char *filename);
00230 void window_open ();
00231 void window_new ();
00232 int cores_number ();
00233
00234 #endif
```

Index

ALGORITHM_GENETIC
    calibrator.h, 99
ALGORITHM_MONTE_CARLO
    calibrator.h, 99
ALGORITHM_SWEEP
    calibrator.h, 99

Calibrate, 13
calibrator.h
    ALGORITHM_GENETIC, 99
    ALGORITHM_MONTE_CARLO, 99
    ALGORITHM_SWEEP, 99

Experiment, 15

Input, 15

Options, 17

Running, 18

Variable, 18

Window, 19