Calibrator

1.1.26

Generated by Doxygen 1.8.8

Wed Oct 28 2015 16:37:55

Contents

Chapter 1

CALIBRATOR (1.1.26 version)

A software to perform calibrations or optimizations of empirical parameters.

AUTHORS

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

REQUIRED

- mpicc, 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)
- gthreads (to use multicores in shared memory machines)
- glib (extended utilities of C to work with data, lists, mapped files, regular expressions, ...)
- genetic (genetic algorithm)
- openmpi or mpich (optional: to run in parallelized tasks)
- doxygen (optional: standard comments format to generate documentation)
- latex (optional: to build the PDF manuals)

FILES

- configure.ac: configure generator.
- Makefile.in: Makefile generator.
- config.h.in: config header generator.
- calibrator.c: source code.

- Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- README.md: this file.
- tests/testX/∗: several tests to check the program working.

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 8.2 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2.4

FreeBSD 10.2

NetBSD 6.1.5

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/1.1.26
   $ ln -s ../../genetic/0.6.1 genetic

4. Build doing on a terminal:

   $ ./build

OpenBSD 5.7

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 8.2 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 8.2 section.

MAKING REFERENCE MANUAL INSTRUCTIONS (file latex/refman.pdf)

On UNIX type systems you need `texlive` installed. On Windows systems you need `MiKTeX`. Then do in a terminal:

$ cd calibrator/1.1.26

$ doxygen

$ cd latex

$ make

USER INSTRUCTIONS

- Command line in sequential mode:

  $ ./calibrator [-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] ./calibrator [-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

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" format="c_string_format" sweeps="sweep
    ...
    <variable name="variable_M" minimum="min_value" maximum="max_value" format="c_string_format" sweeps="sweep
</calibrate>
```

- ∗"weight"∗ associated to every experiment multiplies the objective value obtained for every experiment in the final objective function value.

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:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

`#include <calibrator.h>`

Data Fields

- 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.
- 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 128 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 * name
  File name.
- char * template [MAX_NINPUTS]
  Array of input template names.
- double weight
  Weight to calculate the objective function value.

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 49 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 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
- 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 60 of file calibrator.h.

4.3.2 Field Documentation

4.3.2.1 Input::nbits

Parameters

Array	of bits numbers of the genetic algorithm.

Definition at line 120 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

- GtkLabel * label_processors
  Processors number GtkLabel.
- GtkSpinButton * spin_processors
  Processors number GtkSpinButton.
- GtkGrid * grid
  main GtkGrid.
- GtkDialog * dialog
  main GtkDialog.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 96 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 228 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

- GtkLabel * label
  GtkLabel.
- GtkDialog * dialog
  main GtkDialog.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 118 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 67 of file interface.h.

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

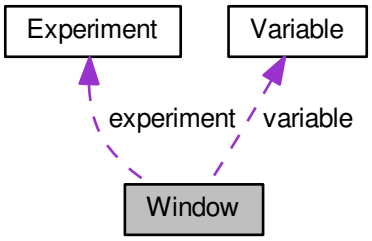
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

- 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.
- GtkButton * button_add_variable
  GtkButton to add a variable.
- GtkButton * button_remove_variable
  GtkButton to remove a variable.
- GtkButton * button_add_experiment

GtkButton to add a experiment.

- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkRadioButton * button_algorithm [NALGORITHMS]

  Array of GtkButtons to set the algorithm.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

  Array of GtkCheckButtons to set the input templates.
- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkLabel * label_simulations

  GtkLabel to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkLabel * label_weight

  Weight GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.

- GtkComboBoxText * combo_variable

  Variable GtkComboBoxEntry.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkFileChooserButton * button_experiment

  GtkFileChooserButton to set the experimental data file.
- GtkFileChooserButton * button_template [MAX_NINPUTS]

  Array of GtkFileChooserButtons to set the input templates.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkToolbar * bar_buttons

  GtkToolbar to store the main buttons.
- GtkGrid * grid

  Main GtkGrid.
- GtkGrid * grid_algorithm

GtkGrid to set the algorithm.

- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GdkPixbuf * logo

  Logo GdkPixbuf.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkWindow * window

  Main GtkWindow.
- GtkApplication * application

  Main GtkApplication.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- gulong id_experiment

  Identifier (gulong) of the combo_experiment signal.
- gulong id_experiment_name

  Identifier (gulong) of the button_experiment signal.
- gulong id_variable

  Identifier (gulong) of the combo_variable signal.
- gulong id_variable_label

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

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

  Array of identifiers (gulong) 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 134 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 <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.
- int input_open (char *filename)

  Function to open the input file.
- void input_free ()

  Function to free the memory of the input file data.
- 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.
- char * current_directory

  Application directory.
- 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 1267 of file calibrator.c.

```
01268 {
01269   unsigned int i, j;
01270   double e;
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_best_sequential: start\n");
01273 #endif
01274   if (calibrate->nsaveds < calibrate->nbest
01275       || value < calibrate->error_best[calibrate->nsaveds - 1])
01276     {
01277       if (calibrate->nsaveds < calibrate->nbest)
01278         ++calibrate->nsaveds;
01279       calibrate->error_best[calibrate->nsaveds - 1] = value;
01280       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01281       for (i = calibrate->nsaveds; --i;)
01282         {
01283           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01284             {
01285               j = calibrate->simulation_best[i];
01286               e = calibrate->error_best[i];
01287               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01288               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
```

```
01289               calibrate->simulation_best[i - 1] = j;
01290               calibrate->error_best[i - 1] = e;
01291             }
01292           else
01293             break;
01294         }
01295     }
01296 #if DEBUG
01297   fprintf (stderr, "calibrate_best_sequential: end\n");
01298 #endif
01299 }
```

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 1222 of file calibrator.c.

```
01223 {
01224   unsigned int i, j;
01225   double e;
01226 #if DEBUG
01227   fprintf (stderr, "calibrate_best_thread: start\n");
01228 #endif
01229   if (calibrate->nsaveds < calibrate->nbest
01230       || value < calibrate->error_best[calibrate->nsaveds - 1])
01231     {
01232       g_mutex_lock (mutex);
01233       if (calibrate->nsaveds < calibrate->nbest)
01234         ++calibrate->nsaveds;
01235       calibrate->error_best[calibrate->nsaveds - 1] = value;
01236       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01237       for (i = calibrate->nsaveds; --i;)
01238         {
01239           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01240             {
01241               j = calibrate->simulation_best[i];
01242               e = calibrate->error_best[i];
01243               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01244               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01245               calibrate->simulation_best[i - 1] = j;
01246               calibrate->error_best[i - 1] = e;
01247             }
01248           else
01249             break;
01250         }
01251       g_mutex_unlock (mutex);
01252     }
01253 #if DEBUG
01254   fprintf (stderr, "calibrate_best_thread: end\n");
01255 #endif
01256 }
```

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

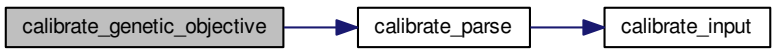
Returns

objective function value.

Definition at line 1576 of file calibrator.c.

```
01577 {
01578   unsigned int j;
01579   double objective;
01580   char buffer[64];
01581 #if DEBUG
01582   fprintf (stderr, "calibrate_genetic_objective: start\n");
01583 #endif
01584   for (j = 0; j < calibrate->nvariables; ++j)
01585     {
01586       calibrate->value[entity->id * calibrate->nvariables + j]
01587         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01588     }
01589   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01590     objective += calibrate_parse (entity->id, j);
01591   g_mutex_lock (mutex);
01592   for (j = 0; j < calibrate->nvariables; ++j)
01593     {
01594       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01595       fprintf (calibrate->file_variables, buffer,
01596                genetic_get_variable (entity, calibrate->
    genetic_variable + j));
01597     }
01598   fprintf (calibrate->file_variables, "%.14le\n", objective);
01599   g_mutex_unlock (mutex);
01600 #if DEBUG
01601   fprintf (stderr, "calibrate_genetic_objective: end\n");
01602 #endif
01603   return objective;
01604 }
```

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 971 of file calibrator.c.

```
00972 {
00973   unsigned int i;
00974   char buffer[32], value[32], *buffer2, *buffer3, *content;
00975   FILE *file;
00976   gsize length;
00977   GRegex *regex;
00978
00979 #if DEBUG
00980   fprintf (stderr, "calibrate_input: start\n");
00981 #endif
00982
00983   // Checking the file
00984   if (!template)
```

```
00985     goto calibrate_input_end;
00986
00987   // Opening template
00988   content = g_mapped_file_get_contents (template);
00989   length = g_mapped_file_get_length (template);
00990 #if DEBUG
00991   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
00992            content);
00993 #endif
00994   file = fopen (input, "w");
00995
00996   // Parsing template
00997   for (i = 0; i < calibrate->nvariables; ++i)
00998     {
00999 #if DEBUG
01000       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01001 #endif
01002       snprintf (buffer, 32, "@variable%u@", i + 1);
01003       regex = g_regex_new (buffer, 0, 0, NULL);
01004       if (i == 0)
01005         {
01006           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01007                                              calibrate->label[i], 0, NULL);
01008 #if DEBUG
01009           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01010 #endif
01011         }
01012       else
01013         {
01014           length = strlen (buffer3);
01015           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01016                                              calibrate->label[i], 0, NULL);
01017           g_free (buffer3);
01018         }
01019       g_regex_unref (regex);
01020       length = strlen (buffer2);
01021       snprintf (buffer, 32, "@value%u@", i + 1);
01022       regex = g_regex_new (buffer, 0, 0, NULL);
01023       snprintf (value, 32, format[calibrate->precision[i]],
01024                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01025
01026 #if DEBUG
01027       fprintf (stderr, "calibrate_input: value=%s\n", value);
01028 #endif
01029       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01030                                          0, NULL);
01031       g_free (buffer2);
01032       g_regex_unref (regex);
01033     }
01034
01035   // Saving input file
01036   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01037   g_free (buffer3);
01038   fclose (file);
01039
01040 calibrate_input_end:
01041 #if DEBUG
01042   fprintf (stderr, "calibrate_input: end\n");
01043 #endif
01044   return;
01045 }
```

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 1383 of file calibrator.c.

```
01385 {
01386   unsigned int i, j, k, s[calibrate->nbest];
01387   double e[calibrate->nbest];
01388 #if DEBUG
01389   fprintf (stderr, "calibrate_merge: start\n");
```

```
01390 #endif
01391   i = j = k = 0;
01392   do
01393     {
01394       if (i == calibrate->nsaveds)
01395         {
01396           s[k] = simulation_best[j];
01397           e[k] = error_best[j];
01398           ++j;
01399           ++k;
01400           if (j == nsaveds)
01401             break;
01402         }
01403       else if (j == nsaveds)
01404         {
01405           s[k] = calibrate->simulation_best[i];
01406           e[k] = calibrate->error_best[i];
01407           ++i;
01408           ++k;
01409           if (i == calibrate->nsaveds)
01410             break;
01411         }
01412       else if (calibrate->error_best[i] > error_best[j])
01413         {
01414           s[k] = simulation_best[j];
01415           e[k] = error_best[j];
01416           ++j;
01417           ++k;
01418         }
01419       else
01420         {
01421           s[k] = calibrate->simulation_best[i];
01422           e[k] = calibrate->error_best[i];
01423           ++i;
01424           ++k;
01425         }
01426     }
01427   while (k < calibrate->nbest);
01428   calibrate->nsaveds = k;
01429   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01430   memcpy (calibrate->error_best, e, k * sizeof (double));
01431 #if DEBUG
01432   fprintf (stderr, "calibrate_merge: end\n");
01433 #endif
01434 }
```

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 1058 of file calibrator.c.

```
01059 {
01060   unsigned int i;
01061   double e;
01062   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01063     *buffer3, *buffer4;
01064   FILE *file_result;
01065
01066 #if DEBUG
01067   fprintf (stderr, "calibrate_parse: start\n");
01068   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01069            experiment);
01070 #endif
01071
01072   // Opening input files
01073   for (i = 0; i < calibrate->ninputs; ++i)
01074     {
01075       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
```

```
01076 #if DEBUG
01077       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01078 #endif
01079       calibrate_input (simulation, &input[i][0],
01080                        calibrate->file[i][experiment]);
01081     }
01082   for (; i < MAX_NINPUTS; ++i)
01083     strcpy (&input[i][0], "");
01084 #if DEBUG
01085   fprintf (stderr, "calibrate_parse: parsing end\n");
01086 #endif
01087
01088   // Performing the simulation
01089   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01090   buffer2 = g_path_get_dirname (calibrate->simulator);
01091   buffer3 = g_path_get_basename (calibrate->simulator);
01092   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01093   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01094             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01095             input[6], input[7], output);
01096   g_free (buffer4);
01097   g_free (buffer3);
01098   g_free (buffer2);
01099 #if DEBUG
01100   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01101 #endif
01102   system (buffer);
01103
01104   // Checking the objective value function
01105   if (calibrate->evaluator)
01106     {
01107       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01108       buffer2 = g_path_get_dirname (calibrate->evaluator);
01109       buffer3 = g_path_get_basename (calibrate->evaluator);
01110       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01111       snprintf (buffer, 512, "\"%s\" %s %s %s",
01112                 buffer4, output, calibrate->experiment[experiment], result);
01113       g_free (buffer4);
01114       g_free (buffer3);
01115       g_free (buffer2);
01116 #if DEBUG
01117       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01118 #endif
01119       system (buffer);
01120       file_result = fopen (result, "r");
01121       e = atof (fgets (buffer, 512, file_result));
01122       fclose (file_result);
01123     }
01124   else
01125     {
01126       strcpy (result, "");
01127       file_result = fopen (output, "r");
01128       e = atof (fgets (buffer, 512, file_result));
01129       fclose (file_result);
01130     }
01131
01132   // Removing files
01133 #if !DEBUG
01134   for (i = 0; i < calibrate->ninputs; ++i)
01135     {
01136       if (calibrate->file[i][0])
01137         {
01138           snprintf (buffer, 512, RM " %s", &input[i][0]);
01139           system (buffer);
01140         }
01141     }
01142   snprintf (buffer, 512, RM " %s %s", output, result);
01143   system (buffer);
01144 #endif
01145
01146 #if DEBUG
01147   fprintf (stderr, "calibrate_parse: end\n");
01148 #endif
01149
01150   // Returning the objective function
01151   return e * calibrate->weight[experiment];
01152 }
```

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 1194 of file calibrator.c.

```
01195 {
01196   unsigned int i;
01197   char buffer[64];
01198 #if DEBUG
01199   fprintf (stderr, "calibrate_save_variables: start\n");
01200 #endif
01201   for (i = 0; i < calibrate->nvariables; ++i)
01202     {
01203       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01204       fprintf (calibrate->file_variables, buffer,
01205                calibrate->value[simulation * calibrate->
      nvariables + i]);
01206     }
01207   fprintf (calibrate->file_variables, "%.14le\n", error);
01208 #if DEBUG
01209   fprintf (stderr, "calibrate_save_variables: end\n");
01210 #endif
01211 }
```

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

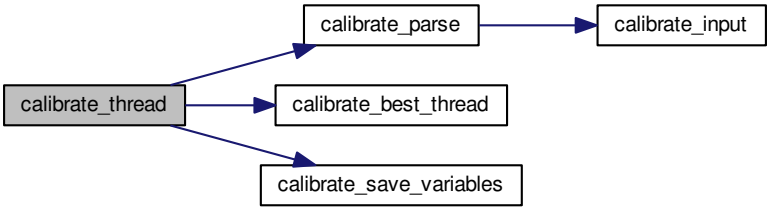
NULL

Definition at line 1309 of file calibrator.c.

```
01310 {
01311   unsigned int i, j, thread;
01312   double e;
01313 #if DEBUG
01314   fprintf (stderr, "calibrate_thread: start\n");
01315 #endif
01316   thread = data->thread;
01317 #if DEBUG
01318   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01319            calibrate->thread[thread], calibrate->thread[thread + 1]);
```

```
01320 #endif
01321   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01322     {
01323       e = 0.;
01324       for (j = 0; j < calibrate->nexperiments; ++j)
01325         e += calibrate_parse (i, j);
01326       calibrate_best_thread (i, e);
01327       g_mutex_lock (mutex);
01328       calibrate_save_variables (i, e);
01329       g_mutex_unlock (mutex);
01330 #if DEBUG
01331       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01332 #endif
01333     }
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_thread: end\n");
01336 #endif
01337   g_thread_exit (NULL);
01338   return NULL;
01339 }
```

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 3731 of file calibrator.c.

```
03732 {
03733 #ifdef G_OS_WIN32
03734   SYSTEM_INFO sysinfo;
03735   GetSystemInfo (&sysinfo);
03736   return sysinfo.dwNumberOfProcessors;
03737 #else
03738   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03739 #endif
03740 }
```

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 458 of file calibrator.c.

```
00459 {
00460   int error_code;
00461   unsigned int i;
00462   char buffer2[64];
00463   xmlChar *buffer;
00464   xmlDoc *doc;
00465   xmlNode *node, *child;
00466 
00467 #if DEBUG
00468   fprintf (stderr, "input_new: start\n");
00469 #endif
00470 
00471   // Resetting input data
00472   input_new ();
00473 
00474   // Parsing the input file
00475   doc = xmlParseFile (filename);
00476   if (!doc)
00477     {
00478       show_error (gettext ("Unable to parse the input file"));
00479       return 0;
00480     }
00481 
00482   // Getting the root node
00483   node = xmlDocGetRootElement (doc);
00484   if (xmlStrcmp (node->name, XML_CALIBRATE))
00485     {
00486       show_error (gettext ("Bad root XML node"));
00487       return 0;
00488     }
00489 
00490   // Opening simulator program name
00491   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00492   if (!input->simulator)
00493     {
00494       show_error (gettext ("Bad simulator program"));
00495       return 0;
00496     }
00497 
00498   // Opening evaluator program name
00499   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00500 
00501   // Opening algorithm
00502   buffer = xmlGetProp (node, XML_ALGORITHM);
00503   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00504     {
00505       input->algorithm = ALGORITHM_MONTE_CARLO;
00506 
00507       // Obtaining simulations number
00508       input->nsimulations
00509         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00510       if (error_code)
00511         {
00512           show_error (gettext ("Bad simulations number"));
00513           return 0;
00514         }
00515     }
00516   else if (!xmlStrcmp (buffer, XML_SWEEP))
00517     input->algorithm = ALGORITHM_SWEEP;
00518   else if (!xmlStrcmp (buffer, XML_GENETIC))
00519     {
00520       input->algorithm = ALGORITHM_GENETIC;
00521 
00522       // Obtaining population
00523       if (xmlHasProp (node, XML_NPOPULATION))
00524         {
00525           input->nsimulations
00526             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00527           if (error_code || input->nsimulations < 3)
00528             {
00529               show_error (gettext ("Invalid population number"));
00530               return 0;
00531             }
```

```
00532         }
00533       else
00534         {
00535           show_error (gettext ("No population number"));
00536           return 0;
00537         }
00538
00539       // Obtaining generations
00540       if (xmlHasProp (node, XML_NGENERATIONS))
00541         {
00542           input->niterations
00543             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00544           if (error_code || !input->niterations)
00545             {
00546               show_error (gettext ("Invalid generation number"));
00547               return 0;
00548             }
00549         }
00550       else
00551         {
00552           show_error (gettext ("No generation number"));
00553           return 0;
00554         }
00555
00556       // Obtaining mutation probability
00557       if (xmlHasProp (node, XML_MUTATION))
00558         {
00559           input->mutation_ratio
00560             = xml_node_get_float (node, XML_MUTATION, &error_code);
00561           if (error_code || input->mutation_ratio < 0.
00562               || input->mutation_ratio >= 1.)
00563             {
00564               show_error (gettext ("Invalid mutation probability"));
00565               return 0;
00566             }
00567         }
00568       else
00569         {
00570           show_error (gettext ("No mutation probability"));
00571           return 0;
00572         }
00573
00574       // Obtaining reproduction probability
00575       if (xmlHasProp (node, XML_REPRODUCTION))
00576         {
00577           input->reproduction_ratio
00578             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00579           if (error_code || input->reproduction_ratio < 0.
00580               || input->reproduction_ratio >= 1.0)
00581             {
00582               show_error (gettext ("Invalid reproduction probability"));
00583               return 0;
00584             }
00585         }
00586       else
00587         {
00588           show_error (gettext ("No reproduction probability"));
00589           return 0;
00590         }
00591
00592       // Obtaining adaptation probability
00593       if (xmlHasProp (node, XML_ADAPTATION))
00594         {
00595           input->adaptation_ratio
00596             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00597           if (error_code || input->adaptation_ratio < 0.
00598               || input->adaptation_ratio >= 1.)
00599             {
00600               show_error (gettext ("Invalid adaptation probability"));
00601               return 0;
00602             }
00603         }
00604       else
00605         {
00606           show_error (gettext ("No adaptation probability"));
00607           return 0;
00608         }
00609
00610       // Checking survivals
00611       i = input->mutation_ratio * input->nsimulations;
00612       i += input->reproduction_ratio * input->
     nsimulations;
00613       i += input->adaptation_ratio * input->
     nsimulations;
00614       if (i > input->nsimulations - 2)
00615         {
00616           show_error
```

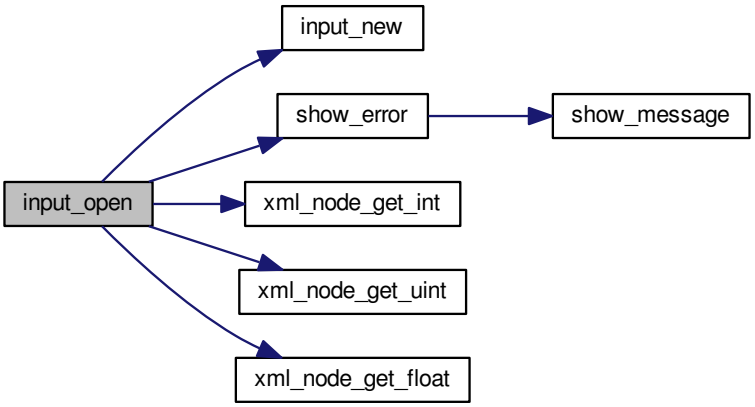
```
00617             (gettext
00618              ("No enough survival entities to reproduce the population"));
00619           return 0;
00620         }
00621     }
00622   else
00623     {
00624       show_error (gettext ("Unknown algorithm"));
00625       return 0;
00626     }
00627
00628   if (input->algorithm == ALGORITHM_MONTE_CARLO
00629       || input->algorithm == ALGORITHM_SWEEP)
00630     {
00631
00632       // Obtaining iterations number
00633       input->niterations
00634         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00635       if (error_code == 1)
00636         input->niterations = 1;
00637       else if (error_code)
00638         {
00639           show_error (gettext ("Bad iterations number"));
00640           return 0;
00641         }
00642
00643       // Obtaining best number
00644       if (xmlHasProp (node, XML_NBEST))
00645         {
00646           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00647           if (error_code || !input->nbest)
00648             {
00649               show_error (gettext ("Invalid best number"));
00650               return 0;
00651             }
00652         }
00653       else
00654         input->nbest = 1;
00655
00656       // Obtaining tolerance
00657       if (xmlHasProp (node, XML_TOLERANCE))
00658         {
00659           input->tolerance
00660             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00661           if (error_code || input->tolerance < 0.)
00662             {
00663               show_error (gettext ("Invalid tolerance"));
00664               return 0;
00665             }
00666         }
00667       else
00668         input->tolerance = 0.;
00669     }
00670
00671   // Reading the experimental data
00672   for (child = node->children; child; child = child->next)
00673     {
00674       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00675         break;
00676 #if DEBUG
00677       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00678 #endif
00679       if (xmlHasProp (child, XML_NAME))
00680         {
00681           input->experiment
00682             = g_realloc (input->experiment,
00683                          (1 + input->nexperiments) * sizeof (char *));
00684           input->experiment[input->nexperiments]
00685             = (char *) xmlGetProp (child, XML_NAME);
00686         }
00687       else
00688         {
00689           show_error (gettext ("No experiment file name"));
00690           return 0;
00691         }
00692 #if DEBUG
00693       fprintf (stderr, "input_new: experiment=%s\n",
00694                input->experiment[input->nexperiments]);
00695 #endif
00696       input->weight = g_realloc (input->weight,
00697                                  (1 + input->nexperiments) * sizeof (double));
00698       if (xmlHasProp (child, XML_WEIGHT))
00699         input->weight[input->nexperiments]
00700           = xml_node_get_float (child, XML_WEIGHT, &error_code);
00701       else
00702         input->weight[input->nexperiments] = 1.;
```

```
00703 #if DEBUG
00704       fprintf (stderr, "input_new: weight=%lg\n",
00705                input->weight[input->nexperiments]);
00706 #endif
00707       if (!input->nexperiments)
00708         input->ninputs = 0;
00709 #if DEBUG
00710       fprintf (stderr, "input_new: template[0]\n");
00711 #endif
00712       if (xmlHasProp (child, XML_TEMPLATE1))
00713         {
00714           input->template[0]
00715             = (char **) g_realloc (input->template[0],
00716                                    (1 + input->nexperiments) * sizeof (char *));
00717           input->template[0][input->nexperiments]
00718             = (char *) xmlGetProp (child, template[0]);
00719 #if DEBUG
00720           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00721                    input->nexperiments,
00722                    input->template[0][input->nexperiments]);
00723 #endif
00724           if (!input->nexperiments)
00725             ++input->ninputs;
00726 #if DEBUG
00727           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00728 #endif
00729         }
00730       else
00731         {
00732           show_error (gettext ("No experiment template"));
00733           return 0;
00734         }
00735       for (i = 1; i < MAX_NINPUTS; ++i)
00736         {
00737 #if DEBUG
00738           fprintf (stderr, "input_new: template%u\n", i + 1);
00739 #endif
00740           if (xmlHasProp (child, template[i]))
00741             {
00742               if (input->nexperiments && input->ninputs < 2)
00743                 {
00744                   snprintf (buffer2, 64,
00745                             gettext ("Experiment %u: bad templates number"),
00746                             input->nexperiments + 1);
00747                   show_error (buffer2);
00748                   return 0;
00749                 }
00750               input->template[i] = (char **)
00751                 g_realloc (input->template[i],
00752                            (1 + input->nexperiments) * sizeof (char *));
00753               input->template[i][input->nexperiments]
00754                 = (char *) xmlGetProp (child, template[i]);
00755 #if DEBUG
00756               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00757                        input->nexperiments, i + 1,
00758                        input->template[i][input->nexperiments]);
00759 #endif
00760               if (!input->nexperiments)
00761                 ++input->ninputs;
00762 #if DEBUG
00763               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00764 #endif
00765             }
00766           else if (input->nexperiments && input->ninputs > 1)
00767             {
00768               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00769                         input->nexperiments + 1, i + 1);
00770               show_error (buffer2);
00771               return 0;
00772             }
00773           else
00774             break;
00775         }
00776       ++input->nexperiments;
00777 #if DEBUG
00778       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00779 #endif
00780     }
00781   if (!input->nexperiments)
00782     {
00783       show_error (gettext ("No calibration experiments"));
00784       return 0;
00785     }
00786
00787   // Reading the variables data
00788   for (; child; child = child->next)
00789     {
```

```
00790       if (xmlStrcmp (child->name, XML_VARIABLE))
00791         {
00792           show_error (gettext ("Bad XML node"));
00793           return 0;
00794         }
00795       if (xmlHasProp (child, XML_NAME))
00796         {
00797           input->label = g_realloc
00798             (input->label, (1 + input->nvariables) * sizeof (char *));
00799           input->label[input->nvariables]
00800             = (char *) xmlGetProp (child, XML_NAME);
00801         }
00802       else
00803         {
00804           show_error (gettext ("No variable name"));
00805           return 0;
00806         }
00807       if (xmlHasProp (child, XML_MINIMUM))
00808         {
00809           input->rangemin = g_realloc
00810             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00811           input->rangeminabs = g_realloc
00812             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00813           input->rangemin[input->nvariables]
00814             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00815           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00816             {
00817               input->rangeminabs[input->nvariables]
00818                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00819             }
00820           else
00821             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00822         }
00823       else
00824         {
00825           show_error (gettext ("No minimum range"));
00826           return 0;
00827         }
00828       if (xmlHasProp (child, XML_MAXIMUM))
00829         {
00830           input->rangemax = g_realloc
00831             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00832           input->rangemaxabs = g_realloc
00833             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00834           input->rangemax[input->nvariables]
00835             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00836           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00837             input->rangemaxabs[input->nvariables]
00838               = xml_node_get_float (child,
    XML_ABSOLUTE_MAXIMUM, &error_code);
00839           else
00840             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00841         }
00842       else
00843         {
00844           show_error (gettext ("No maximum range"));
00845           return 0;
00846         }
00847       input->precision = g_realloc
00848         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00849       if (xmlHasProp (child, XML_PRECISION))
00850         input->precision[input->nvariables]
00851           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00852       else
00853         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00854       if (input->algorithm == ALGORITHM_SWEEP)
00855         {
00856           if (xmlHasProp (child, XML_NSWEEPS))
00857             {
00858               input->nsweeps = (unsigned int *)
00859                 g_realloc (input->nsweeps,
00860                            (1 + input->nvariables) * sizeof (unsigned int));
00861               input->nsweeps[input->nvariables]
00862                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00863             }
00864           else
00865             {
00866               show_error (gettext ("No sweeps number"));
00867               return 0;
00868             }
00869 #if DEBUG
00870           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00871                    input->nsweeps[input->nvariables],
    input->nsimulations);
00872 #endif
```

```
00873         }
00874       if (input->algorithm == ALGORITHM_GENETIC)
00875         {
00876           // Obtaining bits representing each variable
00877           if (xmlHasProp (child, XML_NBITS))
00878             {
00879               input->nbits = (unsigned int *)
00880                 g_realloc (input->nbits,
00881                            (1 + input->nvariables) * sizeof (unsigned int));
00882               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00883               if (error_code || !i)
00884                 {
00885                   show_error (gettext ("Invalid bit number"));
00886                   return 0;
00887                 }
00888               input->nbits[input->nvariables] = i;
00889             }
00890           else
00891             {
00892               show_error (gettext ("No bits number"));
00893               return 0;
00894             }
00895         }
00896       ++input->nvariables;
00897     }
00898   if (!input->nvariables)
00899     {
00900       show_error (gettext ("No calibration variables"));
00901       return 0;
00902     }
00903
00904   // Getting the working directory
00905   input->directory = g_path_get_dirname (filename);
00906   input->name = g_path_get_basename (filename);
00907
00908   // Closing the XML document
00909   xmlFreeDoc (doc);
00910
00911 #if DEBUG
00912   fprintf (stderr, "input_new: end\n");
00913 #endif
00914
00915   return 1;
00916 }
```

Here is the call graph for this function:



5.1.2.11 void input_save (char * *filename*)

Function to save the input file.

Parameters

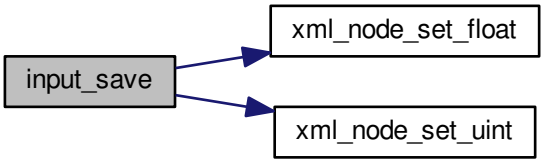
filename	Input file name.

Definition at line 2070 of file calibrator.c.

```
02071 {
02072   unsigned int i, j;
02073   char *buffer;
02074   xmlDoc *doc;
02075   xmlNode *node, *child;
02076   GFile *file, *file2;
02077
02078   // Getting the input file directory
02079   input->name = g_path_get_basename (filename);
02080   input->directory = g_path_get_dirname (filename);
02081   file = g_file_new_for_path (input->directory);
02082
02083   // Opening the input file
02084   doc = xmlNewDoc ((const xmlChar *) "1.0");
02085
02086   // Setting root XML node
02087   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02088   xmlDocSetRootElement (doc, node);
02089
02090   // Adding properties to the root XML node
02091   file2 = g_file_new_for_path (input->simulator);
02092   buffer = g_file_get_relative_path (file, file2);
02093   g_object_unref (file2);
02094   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02095   g_free (buffer);
02096   if (input->evaluator)
02097     {
02098       file2 = g_file_new_for_path (input->evaluator);
02099       buffer = g_file_get_relative_path (file, file2);
02100       g_object_unref (file2);
02101       if (xmlStrlen ((xmlChar *) buffer))
02102         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02103       g_free (buffer);
02104     }
02105
02106   // Setting the algorithm
02107   buffer = (char *) g_malloc (64);
02108   switch (input->algorithm)
02109     {
02110     case ALGORITHM_MONTE_CARLO:
02111       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02112       snprintf (buffer, 64, "%u", input->nsimulations);
02113       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02114       snprintf (buffer, 64, "%u", input->niterations);
02115       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02116       snprintf (buffer, 64, "%.3lg", input->tolerance);
02117       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02118       snprintf (buffer, 64, "%u", input->nbest);
02119       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02120       break;
02121     case ALGORITHM_SWEEP:
02122       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02123       snprintf (buffer, 64, "%u", input->niterations);
02124       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%.3lg", input->tolerance);
02126       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02127       snprintf (buffer, 64, "%u", input->nbest);
02128       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02129       break;
02130     default:
02131       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02132       snprintf (buffer, 64, "%u", input->nsimulations);
02133       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->niterations);
02135       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02136       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02137       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02138       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02139       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02140       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02141       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02142       break;
02143     }
02144   g_free (buffer);
02145
02146   // Setting the experimental data
02147   for (i = 0; i < input->nexperiments; ++i)
02148     {
02149       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02150       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
```

```
02151       if (input->weight[i] != 1.)
02152         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02153       for (j = 0; j < input->ninputs; ++j)
02154         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02155     }
02156
02157   // Setting the variables data
02158   for (i = 0; i < input->nvariables; ++i)
02159     {
02160       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02161       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02162       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02163       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02164         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02165       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02166       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02167         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02168       if (input->precision[i] != DEFAULT_PRECISION)
02169         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02170       if (input->algorithm == ALGORITHM_SWEEP)
02171         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02172       else if (input->algorithm == ALGORITHM_GENETIC)
02173         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02174     }
02175
02176   // Saving the XML file
02177   xmlSaveFormatFile (filename, doc, 1);
02178
02179   // Freeing memory
02180   xmlFreeDoc (doc);
02181 }
```

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.

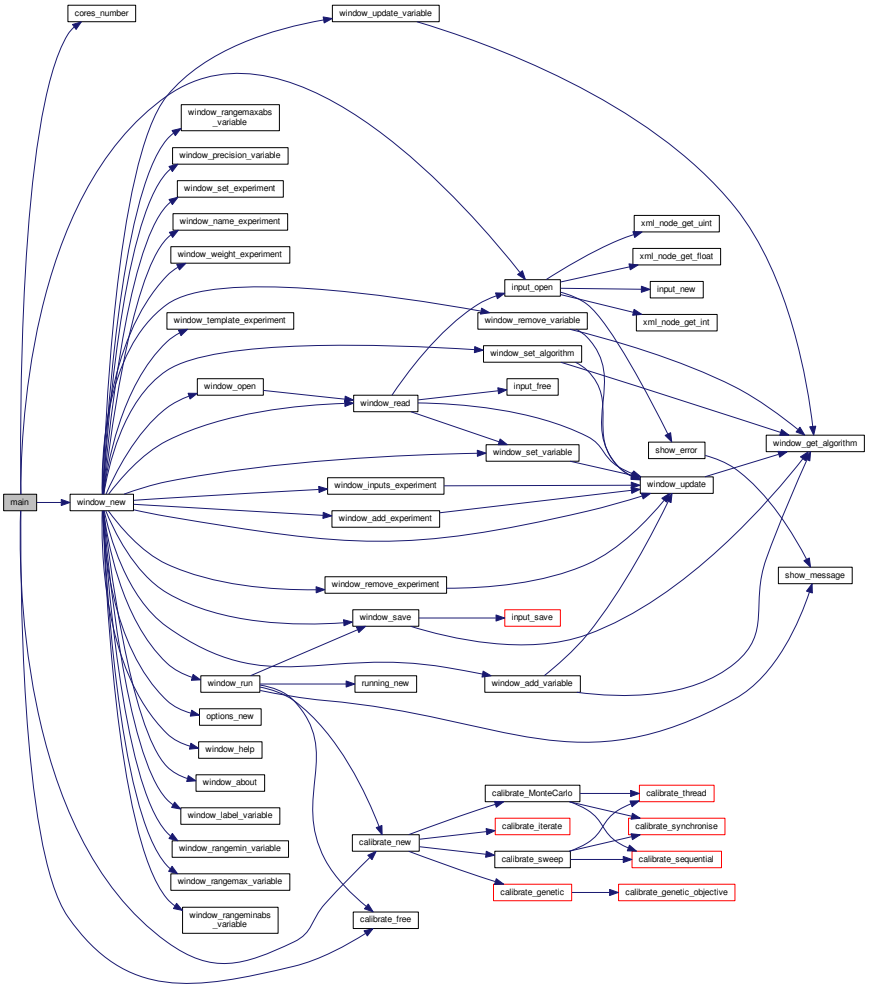
Returns

0 on success, >0 on error.

Definition at line 3752 of file calibrator.c.

```
03753 {
03754 #if HAVE_GTK
03755   int status;
03756 #endif
03757 #if HAVE_MPI
03758   // Starting MPI
03759   MPI_Init (&argn, &argc);
03760   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03761   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03762   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03763 #else
03764   ntasks = 1;
03765 #endif
03766   // Starting pseudo-random numbers generator
03767   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03768   gsl_rng_set (calibrate->rng, DEFAULT_RANDOM_SEED);
03769   // Allowing spaces in the XML data file
03770   xmlKeepBlanksDefault (0);
03771
03772 #if HAVE_GTK
03773
03774   nthreads = cores_number ();
03775   setlocale (LC_ALL, "");
03776   setlocale (LC_NUMERIC, "C");
03777   current_directory = g_get_current_dir ();
03778   bindtextdomain
03779     (PROGRAM_INTERFACE, g_build_filename (current_directory,
     LOCALE_DIR, NULL));
03780   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03781   textdomain (PROGRAM_INTERFACE);
03782   gtk_disable_setlocale ();
03783   window->application = gtk_application_new ("git.jburguete.calibrator",
03784                                              G_APPLICATION_FLAGS_NONE);
03785   g_signal_connect (window->application, "activate", window_new, NULL);
03786   status = g_application_run (G_APPLICATION (window->application), argn, argc);
03787   g_object_unref (window->application);
03788
03789 #else
03790
03791   // Checking syntax
03792   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03793     {
03794       printf ("The syntax is:\ncalibrator [-nthreads x] data_file\n");
03795 #if HAVE_MPI
03796       // Closing MPI
03797       MPI_Finalize ();
03798 #endif
03799       return 1;
03800     }
03801   // Getting threads number
03802   if (argn == 2)
03803     nthreads = cores_number ();
03804   else
03805     nthreads = atoi (argc[2]);
03806   printf ("nthreads=%u\n", nthreads);
03807   // Making calibration
03808   if (input_open (argc[argn - 1]))
03809     calibrate_new ();
03810   // Freeing memory
03811   calibrate_free ();
03812
03813 #endif
03814
03815   // Freeing memory
03816   gsl_rng_free (calibrate->rng);
03817 #if HAVE_MPI
03818   // Closing MPI
03819   MPI_Finalize ();
03820 #endif
03821
03822 #if HAVE_GTK
03823   g_free (current_directory);
03824   return status;
03825 #else
03826   return 0;
03827 #endif
03828 }
```

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.

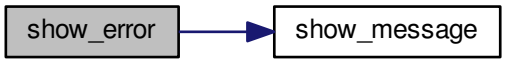
Parameters

msg	Error message.

Definition at line 273 of file calibrator.c.

```
00274 {
00275   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00276 }
```

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 243 of file calibrator.c.

```
00244 {
00245 #if HAVE_GTK
00246   GtkMessageDialog *dlg;
00247
00248   // Creating the dialog
00249   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00250     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00251
00252   // Setting the dialog title
00253   gtk_window_set_title (GTK_WINDOW (dlg), title);
00254
00255   // Showing the dialog and waiting response
00256   gtk_dialog_run (GTK_DIALOG (dlg));
00257
00258   // Closing and freeing memory
00259   gtk_widget_destroy (GTK_WIDGET (dlg));
00260
00261 #else
00262   printf ("%s: %s\n", title, msg);
00263 #endif
00264 }
```

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2421 of file calibrator.c.

```
02422 {
02423   unsigned int i;
02424   for (i = 0; i < NALGORITHMS; ++i)
02425     if (gtk_toggle_button_get_active
02426         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02427       break;
02428   return i;
02429 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

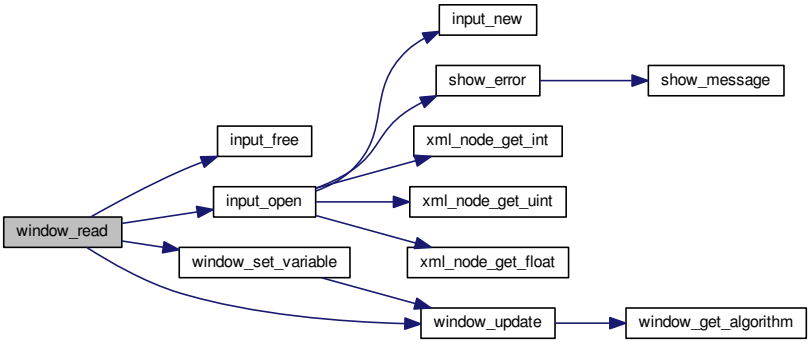
1 on succes, 0 on error.

Definition at line 3137 of file calibrator.c.

```
03138 {
03139   unsigned int i;
03140   char *buffer;
03141 #if DEBUG
03142   fprintf (stderr, "window_read: start\n");
03143 #endif
03144   input_free ();
03145   if (!input_open (filename))
03146     {
03147 #if DEBUG
03148       fprintf (stderr, "window_read: end\n");
03149 #endif
03150       return 0;
03151     }
03152   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03153   puts (buffer);
03154   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03155                                  (window->button_simulator), buffer);
03156   g_free (buffer);
03157   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03158                                 (size_t) input->evaluator);
03159   if (input->evaluator)
03160     {
03161       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03162       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03163                                      (window->button_evaluator), buffer);
03164       g_free (buffer);
03165     }
03166   gtk_toggle_button_set_active
03167     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03168   switch (input->algorithm)
03169     {
03170     case ALGORITHM_MONTE_CARLO:
03171       gtk_spin_button_set_value (window->spin_simulations,
03172                                  (gdouble) input->nsimulations);
03173     case ALGORITHM_SWEEP:
03174       gtk_spin_button_set_value (window->spin_iterations,
03175                                  (gdouble) input->niterations);
03176       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03177       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03178       break;
03179     default:
03180       gtk_spin_button_set_value (window->spin_population,
03181                                  (gdouble) input->nsimulations);
03182       gtk_spin_button_set_value (window->spin_generations,
03183                                  (gdouble) input->niterations);
03184       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03185       gtk_spin_button_set_value (window->spin_reproduction,
03186                                  input->reproduction_ratio);
03187       gtk_spin_button_set_value (window->spin_adaptation,
03188                                  input->adaptation_ratio);
03189     }
03190   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03191   g_signal_handler_block (window->button_experiment,
03192                           window->id_experiment_name);
03193   gtk_combo_box_text_remove_all (window->combo_experiment);
03194   for (i = 0; i < input->nexperiments; ++i)
03195     gtk_combo_box_text_append_text (window->combo_experiment,
03196                                     input->experiment[i]);
03197   g_signal_handler_unblock
03198     (window->button_experiment, window->
      id_experiment_name);
03199   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03200   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03201   g_signal_handler_block (window->combo_variable, window->
```

```
      id_variable);
03202   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03203   gtk_combo_box_text_remove_all (window->combo_variable);
03204   for (i = 0; i < input->nvariables; ++i)
03205     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03206   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03207   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03208   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03209   window_set_variable ();
03210   window_update ();
03211 #if DEBUG
03212   fprintf (stderr, "window_read: end\n");
03213 #endif
03214   return 1;
03215 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

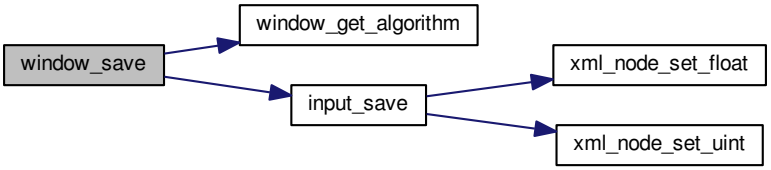
1 on OK, 0 on Cancel.

Definition at line 2240 of file calibrator.c.

```
02241 {
02242   char *buffer;
02243   GtkFileChooserDialog *dlg;
02244
02245 #if DEBUG
02246   fprintf (stderr, "window_save: start\n");
02247 #endif
02248
02249   // Opening the saving dialog
02250   dlg = (GtkFileChooserDialog *)
02251     gtk_file_chooser_dialog_new (gettext ("Save file"),
02252                                  window->window,
02253                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02254                                  gettext ("_Cancel"),
02255                                  GTK_RESPONSE_CANCEL,
02256                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02257   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02258
02259   // If OK response then saving
02260   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02261     {
```

```
02262
02263       // Adding properties to the root XML node
02264       input->simulator = gtk_file_chooser_get_filename
02265         (GTK_FILE_CHOOSER (window->button_simulator));
02266       if (gtk_toggle_button_get_active
02267           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02268         input->evaluator = gtk_file_chooser_get_filename
02269           (GTK_FILE_CHOOSER (window->button_evaluator));
02270       else
02271         input->evaluator = NULL;
02272
02273       // Setting the algorithm
02274       switch (window_get_algorithm ())
02275         {
02276         case ALGORITHM_MONTE_CARLO:
02277           input->algorithm = ALGORITHM_MONTE_CARLO;
02278           input->nsimulations
02279             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02280           input->niterations
02281             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02282           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02283           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02284           break;
02285         case ALGORITHM_SWEEP:
02286           input->algorithm = ALGORITHM_SWEEP;
02287           input->niterations
02288             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02289           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02290           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02291           break;
02292         default:
02293           input->algorithm = ALGORITHM_GENETIC;
02294           input->nsimulations
02295             = gtk_spin_button_get_value_as_int (window->spin_population);
02296           input->niterations
02297             = gtk_spin_button_get_value_as_int (window->spin_generations);
02298           input->mutation_ratio
02299             = gtk_spin_button_get_value (window->spin_mutation);
02300           input->reproduction_ratio
02301             = gtk_spin_button_get_value (window->spin_reproduction);
02302           input->adaptation_ratio
02303             = gtk_spin_button_get_value (window->spin_adaptation);
02304           break;
02305         }
02306
02307       // Saving the XML file
02308       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02309       input_save (buffer);
02310
02311       // Closing and freeing memory
02312       g_free (buffer);
02313       gtk_widget_destroy (GTK_WIDGET (dlg));
02314 #if DEBUG
02315       fprintf (stderr, "window_save: end\n");
02316 #endif
02317       return 1;
02318     }
02319
02320   // Closing and freeing memory
02321   gtk_widget_destroy (GTK_WIDGET (dlg));
02322 #if DEBUG
02323   fprintf (stderr, "window_save: end\n");
02324 #endif
02325   return 0;
02326 }
```

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 2795 of file calibrator.c.

```
02796 {
02797   unsigned int i, j;
02798   char *buffer;
02799   GFile *file1, *file2;
02800 #if DEBUG
02801   fprintf (stderr, "window_template_experiment: start\n");
02802 #endif
02803   i = (size_t) data;
02804   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02805   file1
02806     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02807   file2 = g_file_new_for_path (input->directory);
02808   buffer = g_file_get_relative_path (file2, file1);
02809   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02810   g_free (buffer);
02811   g_object_unref (file2);
02812   g_object_unref (file1);
02813 #if DEBUG
02814   fprintf (stderr, "window_template_experiment: end\n");
02815 #endif
02816 }
```

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 352 of file calibrator.c.

```
00353 {
```

```
00354   double x = 0.;
00355   xmlChar *buffer;
00356   buffer = xmlGetProp (node, prop);
00357   if (!buffer)
00358     *error_code = 1;
00359   else
00360     {
00361       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00362         *error_code = 2;
00363       else
00364         *error_code = 0;
00365       xmlFree (buffer);
00366     }
00367   return x;
00368 }
```

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 290 of file calibrator.c.

```
00291 {
00292   int i = 0;
00293   xmlChar *buffer;
00294   buffer = xmlGetProp (node, prop);
00295   if (!buffer)
00296     *error_code = 1;
00297   else
00298     {
00299       if (sscanf ((char *) buffer, "%d", &i) != 1)
00300         *error_code = 2;
00301       else
00302         *error_code = 0;
00303       xmlFree (buffer);
00304     }
00305   return i;
00306 }
```

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 321 of file calibrator.c.

```
00322 {
00323   unsigned int i = 0;
00324   xmlChar *buffer;
00325   buffer = xmlGetProp (node, prop);
00326   if (!buffer)
00327     *error_code = 1;
00328   else
00329     {
00330       if (sscanf ((char *) buffer, "%u", &i) != 1)
00331         *error_code = 2;
00332       else
00333         *error_code = 0;
00334       xmlFree (buffer);
00335     }
00336   return i;
00337 }
```

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 419 of file calibrator.c.

```
00420 {
00421   xmlChar buffer[64];
00422   snprintf ((char *) buffer, 64, "%.14lg", value);
00423   xmlSetProp (node, prop, buffer);
00424 }
```

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 381 of file calibrator.c.

```
00382 {
00383   xmlChar buffer[64];
00384   snprintf ((char *) buffer, 64, "%d", value);
00385   xmlSetProp (node, prop, buffer);
00386 }
```

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 400 of file calibrator.c.

```
00401 {
00402   xmlChar buffer[64];
00403   snprintf ((char *) buffer, 64, "%u", value);
00404   xmlSetProp (node, prop, buffer);
00405 }
```

5.1.3 Variable Documentation

5.1.3.1 format

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 129 of file calibrator.c.

5.1.3.2 precision

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 134 of file calibrator.c.

5.1.3.3 template

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 124 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 #ifdef G_OS_WIN32
00049 #include <windows.h>
00050 #elif (!__BSD_VISIBLE)
00051 #include <alloca.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #include "genetic/genetic.h"
00057 #include "calibrator.h"
00058 #if HAVE_GTK
00059 #include <gio/gio.h>
00060 #include <gtk/gtk.h>
00061 #include "interface.h"
00062 #endif
00063
00076 #define DEBUG 0
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00116 int ntasks;
00117 unsigned int nthreads;
00118 char *current_directory;
00119 GMutex mutex[1];
00120 void (*calibrate_step) ();
00121 Input input[1];
00122 Calibrate calibrate[1];
00123
00124 const xmlChar *template[MAX_NINPUTS] = {
00125   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00126   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00127 };
00128
00129 const char *format[NPRECISIONS] = {
00130   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00131   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00132 };
00133
00134 const double precision[NPRECISIONS] = {
```

```
00135   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,
00136   1e-13, 1e-14
00137 };
00138
00139 const char *logo[] = {
00140   "32 32 3 1",
00141   "      c None",
00142   ".     c #0000FF",
00143   "+     c #FF0000",
00144   "                                ",
00145   "                                ",
00146   "                                ",
00147   "      .       .       .       . ",
00148   "      .       .       .       . ",
00149   "      .       .       .       . ",
00150   "      .       .       .       . ",
00151   "      .       .      +++      . ",
00152   "      .       .     +++++     . ",
00153   "      .       .     +++++     . ",
00154   "      .       .     +++++     . ",
00155   "     +++      .      +++     +++ ",
00156   "    +++++     .       .     +++++ ",
00157   "    +++++     .       .     +++++ ",
00158   "    +++++     .       .     +++++ ",
00159   "     +++      .       .      +++ ",
00160   "      .       .       .       . ",
00161   "      .      +++      .       . ",
00162   "      .     +++++     .       . ",
00163   "      .     +++++     .       . ",
00164   "      .     +++++     .       . ",
00165   "      .      +++      .       . ",
00166   "      .       .       .       . ",
00167   "      .       .       .       . ",
00168   "      .       .       .       . ",
00169   "      .       .       .       . ",
00170   "      .       .       .       . ",
00171   "      .       .       .       . ",
00172   "      .       .       .       . ",
00173   "                                ",
00174   "                                ",
00175   "                                "
00176 };
00177
00178 /*
00179 const char * logo[] = {
00180 "32 32 3 1",
00181 "   c #FFFFFFFFFFFF",
00182 ".  c #00000000FFFF",
00183 "X  c #FFFF00000000",
00184 "                                ",
00185 "                                ",
00186 "                                ",
00187 "      .       .       .       . ",
00188 "      .       .       .       . ",
00189 "      .       .       .       . ",
00190 "      .       .       .       . ",
00191 "      .       .      XXX      . ",
00192 "      .       .     XXXXX     . ",
00193 "      .       .     XXXXX     . ",
00194 "      .       .     XXXXX     . ",
00195 "     XXX      .      XXX     XXX ",
00196 "    XXXXX     .       .     XXXXX ",
00197 "    XXXXX     .       .     XXXXX ",
00198 "    XXXXX     .       .     XXXXX ",
00199 "     XXX      .       .      XXX ",
00200 "      .       .       .       . ",
00201 "      .      XXX      .       . ",
00202 "      .     XXXXX     .       . ",
00203 "      .     XXXXX     .       . ",
00204 "      .     XXXXX     .       . ",
00205 "      .      XXX      .       . ",
00206 "      .       .       .       . ",
00207 "      .       .       .       . ",
00208 "      .       .       .       . ",
00209 "      .       .       .       . ",
00210 "      .       .       .       . ",
00211 "      .       .       .       . ",
00212 "      .       .       .       . ",
00213 "                                ",
00214 "                                ",
00215 "                                "};
00216 */
00217
00218 #if HAVE_GTK
00219
00227 Options options[1];
00228 Running running[1];
```

```
00229 Window window[1];
00230 #endif
00231
00242 void
00243 show_message (char *title, char *msg, int type)
00244 {
00245 #if HAVE_GTK
00246   GtkMessageDialog *dlg;
00247
00248   // Creating the dialog
00249   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00250     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00251
00252   // Setting the dialog title
00253   gtk_window_set_title (GTK_WINDOW (dlg), title);
00254
00255   // Showing the dialog and waiting response
00256   gtk_dialog_run (GTK_DIALOG (dlg));
00257
00258   // Closing and freeing memory
00259   gtk_widget_destroy (GTK_WIDGET (dlg));
00260
00261 #else
00262   printf ("%s: %s\n", title, msg);
00263 #endif
00264 }
00265
00272 void
00273 show_error (char *msg)
00274 {
00275   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00276 }
00277
00289 int
00290 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00291 {
00292   int i = 0;
00293   xmlChar *buffer;
00294   buffer = xmlGetProp (node, prop);
00295   if (!buffer)
00296     *error_code = 1;
00297   else
00298     {
00299       if (sscanf ((char *) buffer, "%d", &i) != 1)
00300         *error_code = 2;
00301       else
00302         *error_code = 0;
00303       xmlFree (buffer);
00304     }
00305   return i;
00306 }
00307
00320 unsigned int
00321 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00322 {
00323   unsigned int i = 0;
00324   xmlChar *buffer;
00325   buffer = xmlGetProp (node, prop);
00326   if (!buffer)
00327     *error_code = 1;
00328   else
00329     {
00330       if (sscanf ((char *) buffer, "%u", &i) != 1)
00331         *error_code = 2;
00332       else
00333         *error_code = 0;
00334       xmlFree (buffer);
00335     }
00336   return i;
00337 }
00338
00351 double
00352 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00353 {
00354   double x = 0.;
00355   xmlChar *buffer;
00356   buffer = xmlGetProp (node, prop);
00357   if (!buffer)
00358     *error_code = 1;
00359   else
00360     {
00361       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00362         *error_code = 2;
00363       else
00364         *error_code = 0;
00365       xmlFree (buffer);
00366     }
```

```
00367   return x;
00368 }
00369
00380 void
00381 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00382 {
00383   xmlChar buffer[64];
00384   snprintf ((char *) buffer, 64, "%d", value);
00385   xmlSetProp (node, prop, buffer);
00386 }
00387
00399 void
00400 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00401 {
00402   xmlChar buffer[64];
00403   snprintf ((char *) buffer, 64, "%u", value);
00404   xmlSetProp (node, prop, buffer);
00405 }
00406
00418 void
00419 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00420 {
00421   xmlChar buffer[64];
00422   snprintf ((char *) buffer, 64, "%.14lg", value);
00423   xmlSetProp (node, prop, buffer);
00424 }
00425
00430 void
00431 input_new ()
00432 {
00433   unsigned int i;
00434 #if DEBUG
00435   fprintf (stderr, "input_init: start\n");
00436 #endif
00437   input->nvariables = input->nexperiments = input->ninputs = 0;
00438   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00439   input->experiment = input->label = NULL;
00440   input->precision = input->nsweeps = input->nbits = NULL;
00441   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00442     = input->weight = NULL;
00443   for (i = 0; i < MAX_NINPUTS; ++i)
00444     input->template[i] = NULL;
00445 #if DEBUG
00446   fprintf (stderr, "input_init: end\n");
00447 #endif
00448 }
00449
00457 int
00458 input_open (char *filename)
00459 {
00460   int error_code;
00461   unsigned int i;
00462   char buffer2[64];
00463   xmlChar *buffer;
00464   xmlDoc *doc;
00465   xmlNode *node, *child;
00466
00467 #if DEBUG
00468   fprintf (stderr, "input_new: start\n");
00469 #endif
00470
00471   // Resetting input data
00472   input_new ();
00473
00474   // Parsing the input file
00475   doc = xmlParseFile (filename);
00476   if (!doc)
00477     {
00478       show_error (gettext ("Unable to parse the input file"));
00479       return 0;
00480     }
00481
00482   // Getting the root node
00483   node = xmlDocGetRootElement (doc);
00484   if (xmlStrcmp (node->name, XML_CALIBRATE))
00485     {
00486       show_error (gettext ("Bad root XML node"));
00487       return 0;
00488     }
00489
00490   // Opening simulator program name
00491   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00492   if (!input->simulator)
00493     {
00494       show_error (gettext ("Bad simulator program"));
```

```
00495       return 0;
00496     }
00497
00498   // Opening evaluator program name
00499   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00500
00501   // Opening algorithm
00502   buffer = xmlGetProp (node, XML_ALGORITHM);
00503   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00504     {
00505       input->algorithm = ALGORITHM_MONTE_CARLO;
00506
00507       // Obtaining simulations number
00508       input->nsimulations
00509         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00510       if (error_code)
00511         {
00512           show_error (gettext ("Bad simulations number"));
00513           return 0;
00514         }
00515     }
00516   else if (!xmlStrcmp (buffer, XML_SWEEP))
00517     input->algorithm = ALGORITHM_SWEEP;
00518   else if (!xmlStrcmp (buffer, XML_GENETIC))
00519     {
00520       input->algorithm = ALGORITHM_GENETIC;
00521
00522       // Obtaining population
00523       if (xmlHasProp (node, XML_NPOPULATION))
00524         {
00525           input->nsimulations
00526             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00527           if (error_code || input->nsimulations < 3)
00528             {
00529               show_error (gettext ("Invalid population number"));
00530               return 0;
00531             }
00532         }
00533       else
00534         {
00535           show_error (gettext ("No population number"));
00536           return 0;
00537         }
00538
00539       // Obtaining generations
00540       if (xmlHasProp (node, XML_NGENERATIONS))
00541         {
00542           input->niterations
00543             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00544           if (error_code || !input->niterations)
00545             {
00546               show_error (gettext ("Invalid generation number"));
00547               return 0;
00548             }
00549         }
00550       else
00551         {
00552           show_error (gettext ("No generation number"));
00553           return 0;
00554         }
00555
00556       // Obtaining mutation probability
00557       if (xmlHasProp (node, XML_MUTATION))
00558         {
00559           input->mutation_ratio
00560             = xml_node_get_float (node, XML_MUTATION, &error_code);
00561           if (error_code || input->mutation_ratio < 0.
00562               || input->mutation_ratio >= 1.)
00563             {
00564               show_error (gettext ("Invalid mutation probability"));
00565               return 0;
00566             }
00567         }
00568       else
00569         {
00570           show_error (gettext ("No mutation probability"));
00571           return 0;
00572         }
00573
00574       // Obtaining reproduction probability
00575       if (xmlHasProp (node, XML_REPRODUCTION))
00576         {
00577           input->reproduction_ratio
00578             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00579           if (error_code || input->reproduction_ratio < 0.
00580               || input->reproduction_ratio >= 1.0)
00581             {
```

```
00582               show_error (gettext ("Invalid reproduction probability"));
00583               return 0;
00584             }
00585         }
00586       else
00587         {
00588           show_error (gettext ("No reproduction probability"));
00589           return 0;
00590         }
00591
00592       // Obtaining adaptation probability
00593       if (xmlHasProp (node, XML_ADAPTATION))
00594         {
00595           input->adaptation_ratio
00596             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00597           if (error_code || input->adaptation_ratio < 0.
00598               || input->adaptation_ratio >= 1.)
00599             {
00600               show_error (gettext ("Invalid adaptation probability"));
00601               return 0;
00602             }
00603         }
00604       else
00605         {
00606           show_error (gettext ("No adaptation probability"));
00607           return 0;
00608         }
00609
00610       // Checking survivals
00611       i = input->mutation_ratio * input->nsimulations;
00612       i += input->reproduction_ratio * input->nsimulations;
00613       i += input->adaptation_ratio * input->nsimulations;
00614       if (i > input->nsimulations - 2)
00615         {
00616           show_error
00617             (gettext
00618              ("No enough survival entities to reproduce the population"));
00619           return 0;
00620         }
00621     }
00622   else
00623     {
00624       show_error (gettext ("Unknown algorithm"));
00625       return 0;
00626     }
00627
00628   if (input->algorithm == ALGORITHM_MONTE_CARLO
00629       || input->algorithm == ALGORITHM_SWEEP)
00630     {
00631
00632       // Obtaining iterations number
00633       input->niterations
00634         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00635       if (error_code == 1)
00636         input->niterations = 1;
00637       else if (error_code)
00638         {
00639           show_error (gettext ("Bad iterations number"));
00640           return 0;
00641         }
00642
00643       // Obtaining best number
00644       if (xmlHasProp (node, XML_NBEST))
00645         {
00646           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00647           if (error_code || !input->nbest)
00648             {
00649               show_error (gettext ("Invalid best number"));
00650               return 0;
00651             }
00652         }
00653       else
00654         input->nbest = 1;
00655
00656       // Obtaining tolerance
00657       if (xmlHasProp (node, XML_TOLERANCE))
00658         {
00659           input->tolerance
00660             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00661           if (error_code || input->tolerance < 0.)
00662             {
00663               show_error (gettext ("Invalid tolerance"));
00664               return 0;
00665             }
00666         }
00667       else
```

```
00668         input->tolerance = 0.;
00669     }
00670
00671   // Reading the experimental data
00672   for (child = node->children; child; child = child->next)
00673     {
00674       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00675         break;
00676 #if DEBUG
00677       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00678 #endif
00679       if (xmlHasProp (child, XML_NAME))
00680         {
00681           input->experiment
00682             = g_realloc (input->experiment,
00683                          (1 + input->nexperiments) * sizeof (char *));
00684           input->experiment[input->nexperiments]
00685             = (char *) xmlGetProp (child, XML_NAME);
00686         }
00687       else
00688         {
00689           show_error (gettext ("No experiment file name"));
00690           return 0;
00691         }
00692 #if DEBUG
00693       fprintf (stderr, "input_new: experiment=%s\n",
00694                input->experiment[input->nexperiments]);
00695 #endif
00696       input->weight = g_realloc (input->weight,
00697                                  (1 + input->nexperiments) * sizeof (double));
00698       if (xmlHasProp (child, XML_WEIGHT))
00699         input->weight[input->nexperiments]
00700           = xml_node_get_float (child, XML_WEIGHT, &error_code);
00701       else
00702         input->weight[input->nexperiments] = 1.;
00703 #if DEBUG
00704       fprintf (stderr, "input_new: weight=%lg\n",
00705                input->weight[input->nexperiments]);
00706 #endif
00707       if (!input->nexperiments)
00708         input->ninputs = 0;
00709 #if DEBUG
00710       fprintf (stderr, "input_new: template[0]\n");
00711 #endif
00712       if (xmlHasProp (child, XML_TEMPLATE1))
00713         {
00714           input->template[0]
00715             = (char **) g_realloc (input->template[0],
00716                                    (1 + input->nexperiments) * sizeof (char *));
00717           input->template[0][input->nexperiments]
00718             = (char *) xmlGetProp (child, template[0]);
00719 #if DEBUG
00720           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00721                    input->nexperiments,
00722                    input->template[0][input->nexperiments]);
00723 #endif
00724           if (!input->nexperiments)
00725             ++input->ninputs;
00726 #if DEBUG
00727           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00728 #endif
00729         }
00730       else
00731         {
00732           show_error (gettext ("No experiment template"));
00733           return 0;
00734         }
00735       for (i = 1; i < MAX_NINPUTS; ++i)
00736         {
00737 #if DEBUG
00738           fprintf (stderr, "input_new: template%u\n", i + 1);
00739 #endif
00740           if (xmlHasProp (child, template[i]))
00741             {
00742               if (input->nexperiments && input->ninputs < 2)
00743                 {
00744                   snprintf (buffer2, 64,
00745                             gettext ("Experiment %u: bad templates number"),
00746                             input->nexperiments + 1);
00747                   show_error (buffer2);
00748                   return 0;
00749                 }
00750               input->template[i] = (char **)
00751                 g_realloc (input->template[i],
00752                            (1 + input->nexperiments) * sizeof (char *));
00753               input->template[i][input->nexperiments]
00754                 = (char *) xmlGetProp (child, template[i]);
```

```
00755 #if DEBUG
00756               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00757                        input->nexperiments, i + 1,
00758                        input->template[i][input->nexperiments]);
00759 #endif
00760               if (!input->nexperiments)
00761                 ++input->ninputs;
00762 #if DEBUG
00763               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00764 #endif
00765             }
00766           else if (input->nexperiments && input->ninputs > 1)
00767             {
00768               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00769                         input->nexperiments + 1, i + 1);
00770               show_error (buffer2);
00771               return 0;
00772             }
00773           else
00774             break;
00775         }
00776       ++input->nexperiments;
00777 #if DEBUG
00778       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00779 #endif
00780     }
00781   if (!input->nexperiments)
00782     {
00783       show_error (gettext ("No calibration experiments"));
00784       return 0;
00785     }
00786
00787   // Reading the variables data
00788   for (; child; child = child->next)
00789     {
00790       if (xmlStrcmp (child->name, XML_VARIABLE))
00791         {
00792           show_error (gettext ("Bad XML node"));
00793           return 0;
00794         }
00795       if (xmlHasProp (child, XML_NAME))
00796         {
00797           input->label = g_realloc
00798             (input->label, (1 + input->nvariables) * sizeof (char *));
00799           input->label[input->nvariables]
00800             = (char *) xmlGetProp (child, XML_NAME);
00801         }
00802       else
00803         {
00804           show_error (gettext ("No variable name"));
00805           return 0;
00806         }
00807       if (xmlHasProp (child, XML_MINIMUM))
00808         {
00809           input->rangemin = g_realloc
00810             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00811           input->rangeminabs = g_realloc
00812             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00813           input->rangemin[input->nvariables]
00814             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00815           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00816             {
00817               input->rangeminabs[input->nvariables]
00818                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00819             }
00820           else
00821             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00822         }
00823       else
00824         {
00825           show_error (gettext ("No minimum range"));
00826           return 0;
00827         }
00828       if (xmlHasProp (child, XML_MAXIMUM))
00829         {
00830           input->rangemax = g_realloc
00831             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00832           input->rangemaxabs = g_realloc
00833             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00834           input->rangemax[input->nvariables]
00835             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00836           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00837             input->rangemaxabs[input->nvariables]
00838               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00839           else
```

```
00840             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00841         }
00842       else
00843         {
00844           show_error (gettext ("No maximum range"));
00845           return 0;
00846         }
00847       input->precision = g_realloc
00848         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00849       if (xmlHasProp (child, XML_PRECISION))
00850         input->precision[input->nvariables]
00851           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00852       else
00853         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00854       if (input->algorithm == ALGORITHM_SWEEP)
00855         {
00856           if (xmlHasProp (child, XML_NSWEEPS))
00857             {
00858               input->nsweeps = (unsigned int *)
00859                 g_realloc (input->nsweeps,
00860                            (1 + input->nvariables) * sizeof (unsigned int));
00861               input->nsweeps[input->nvariables]
00862                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00863             }
00864           else
00865             {
00866               show_error (gettext ("No sweeps number"));
00867               return 0;
00868             }
00869 #if DEBUG
00870           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00871                    input->nsweeps[input->nvariables], input->
    nsimulations);
00872 #endif
00873         }
00874       if (input->algorithm == ALGORITHM_GENETIC)
00875         {
00876           // Obtaining bits representing each variable
00877           if (xmlHasProp (child, XML_NBITS))
00878             {
00879               input->nbits = (unsigned int *)
00880                 g_realloc (input->nbits,
00881                            (1 + input->nvariables) * sizeof (unsigned int));
00882               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00883               if (error_code || !i)
00884                 {
00885                   show_error (gettext ("Invalid bit number"));
00886                   return 0;
00887                 }
00888               input->nbits[input->nvariables] = i;
00889             }
00890           else
00891             {
00892               show_error (gettext ("No bits number"));
00893               return 0;
00894             }
00895         }
00896       ++input->nvariables;
00897     }
00898   if (!input->nvariables)
00899     {
00900       show_error (gettext ("No calibration variables"));
00901       return 0;
00902     }
00903
00904   // Getting the working directory
00905   input->directory = g_path_get_dirname (filename);
00906   input->name = g_path_get_basename (filename);
00907
00908   // Closing the XML document
00909   xmlFreeDoc (doc);
00910
00911 #if DEBUG
00912   fprintf (stderr, "input_new: end\n");
00913 #endif
00914
00915   return 1;
00916 }
00917
00922 void
00923 input_free ()
00924 {
00925   unsigned int i, j;
00926 #if DEBUG
00927   fprintf (stderr, "input_free: start\n");
00928 #endif
```

```
00929   g_free (input->name);
00930   g_free (input->directory);
00931   for (i = 0; i < input->nexperiments; ++i)
00932     {
00933       xmlFree (input->experiment[i]);
00934       for (j = 0; j < input->ninputs; ++j)
00935         xmlFree (input->template[j][i]);
00936     }
00937   g_free (input->experiment);
00938   for (i = 0; i < input->ninputs; ++i)
00939     g_free (input->template[i]);
00940   for (i = 0; i < input->nvariables; ++i)
00941     xmlFree (input->label[i]);
00942   g_free (input->label);
00943   g_free (input->precision);
00944   g_free (input->rangemin);
00945   g_free (input->rangemax);
00946   g_free (input->rangeminabs);
00947   g_free (input->rangemaxabs);
00948   g_free (input->weight);
00949   g_free (input->nsweeps);
00950   g_free (input->nbits);
00951   xmlFree (input->evaluator);
00952   xmlFree (input->simulator);
00953   input->nexperiments = input->ninputs = input->nvariables = 0;
00954 #if DEBUG
00955   fprintf (stderr, "input_free: end\n");
00956 #endif
00957 }
00958
00970 void
00971 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
00972 {
00973   unsigned int i;
00974   char buffer[32], value[32], *buffer2, *buffer3, *content;
00975   FILE *file;
00976   gsize length;
00977   GRegex *regex;
00978
00979 #if DEBUG
00980   fprintf (stderr, "calibrate_input: start\n");
00981 #endif
00982
00983   // Checking the file
00984   if (!template)
00985     goto calibrate_input_end;
00986
00987   // Opening template
00988   content = g_mapped_file_get_contents (template);
00989   length = g_mapped_file_get_length (template);
00990 #if DEBUG
00991   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
00992            content);
00993 #endif
00994   file = fopen (input, "w");
00995
00996   // Parsing template
00997   for (i = 0; i < calibrate->nvariables; ++i)
00998     {
00999 #if DEBUG
01000       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01001 #endif
01002       snprintf (buffer, 32, "@variable%u@", i + 1);
01003       regex = g_regex_new (buffer, 0, 0, NULL);
01004       if (i == 0)
01005         {
01006           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01007                                              calibrate->label[i], 0, NULL);
01008 #if DEBUG
01009           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01010 #endif
01011         }
01012       else
01013         {
01014           length = strlen (buffer3);
01015           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01016                                              calibrate->label[i], 0, NULL);
01017           g_free (buffer3);
01018         }
01019       g_regex_unref (regex);
01020       length = strlen (buffer2);
01021       snprintf (buffer, 32, "@value%u@", i + 1);
01022       regex = g_regex_new (buffer, 0, 0, NULL);
01023       snprintf (value, 32, format[calibrate->precision[i]],
01024                 calibrate->value[simulation * calibrate->nvariables + i]);
01025
01026 #if DEBUG
```

```
01027       fprintf (stderr, "calibrate_input: value=%s\n", value);
01028 #endif
01029       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01030                                          0, NULL);
01031       g_free (buffer2);
01032       g_regex_unref (regex);
01033     }
01034
01035   // Saving input file
01036   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01037   g_free (buffer3);
01038   fclose (file);
01039
01040 calibrate_input_end:
01041 #if DEBUG
01042   fprintf (stderr, "calibrate_input: end\n");
01043 #endif
01044   return;
01045 }
01046
01057 double
01058 calibrate_parse (unsigned int simulation, unsigned int experiment)
01059 {
01060   unsigned int i;
01061   double e;
01062   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01063     *buffer3, *buffer4;
01064   FILE *file_result;
01065
01066 #if DEBUG
01067   fprintf (stderr, "calibrate_parse: start\n");
01068   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01069            experiment);
01070 #endif
01071
01072   // Opening input files
01073   for (i = 0; i < calibrate->ninputs; ++i)
01074     {
01075       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01076 #if DEBUG
01077       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01078 #endif
01079       calibrate_input (simulation, &input[i][0],
01080                        calibrate->file[i][experiment]);
01081     }
01082   for (; i < MAX_NINPUTS; ++i)
01083     strcpy (&input[i][0], "");
01084 #if DEBUG
01085   fprintf (stderr, "calibrate_parse: parsing end\n");
01086 #endif
01087
01088   // Performing the simulation
01089   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01090   buffer2 = g_path_get_dirname (calibrate->simulator);
01091   buffer3 = g_path_get_basename (calibrate->simulator);
01092   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01093   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01094             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01095             input[6], input[7], output);
01096   g_free (buffer4);
01097   g_free (buffer3);
01098   g_free (buffer2);
01099 #if DEBUG
01100   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01101 #endif
01102   system (buffer);
01103
01104   // Checking the objective value function
01105   if (calibrate->evaluator)
01106     {
01107       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01108       buffer2 = g_path_get_dirname (calibrate->evaluator);
01109       buffer3 = g_path_get_basename (calibrate->evaluator);
01110       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01111       snprintf (buffer, 512, "\"%s\" %s %s %s",
01112                 buffer4, output, calibrate->experiment[experiment], result);
01113       g_free (buffer4);
01114       g_free (buffer3);
01115       g_free (buffer2);
01116 #if DEBUG
01117       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01118 #endif
01119       system (buffer);
01120       file_result = fopen (result, "r");
01121       e = atof (fgets (buffer, 512, file_result));
01122       fclose (file_result);
01123     }
```

```
01124   else
01125     {
01126       strcpy (result, "");
01127       file_result = fopen (output, "r");
01128       e = atof (fgets (buffer, 512, file_result));
01129       fclose (file_result);
01130     }
01131
01132   // Removing files
01133 #if !DEBUG
01134   for (i = 0; i < calibrate->ninputs; ++i)
01135     {
01136       if (calibrate->file[i][0])
01137         {
01138           snprintf (buffer, 512, RM " %s", &input[i][0]);
01139           system (buffer);
01140         }
01141     }
01142   snprintf (buffer, 512, RM " %s %s", output, result);
01143   system (buffer);
01144 #endif
01145
01146 #if DEBUG
01147   fprintf (stderr, "calibrate_parse: end\n");
01148 #endif
01149
01150   // Returning the objective function
01151   return e * calibrate->weight[experiment];
01152 }
01153
01158 void
01159 calibrate_print ()
01160 {
01161   unsigned int i;
01162   char buffer[512];
01163 #if HAVE_MPI
01164   if (!calibrate->mpi_rank)
01165     {
01166 #endif
01167       printf ("THE BEST IS\n");
01168       fprintf (calibrate->file_result, "THE BEST IS\n");
01169       printf ("error=%.15le\n", calibrate->error_old[0]);
01170       fprintf (calibrate->file_result, "error=%.15le\n",
01171                calibrate->error_old[0]);
01172       for (i = 0; i < calibrate->nvariables; ++i)
01173         {
01174           snprintf (buffer, 512, "%s=%s\n",
01175                     calibrate->label[i], format[calibrate->precision[i]]);
01176           printf (buffer, calibrate->value_old[i]);
01177           fprintf (calibrate->file_result, buffer, calibrate->
    value_old[i]);
01178         }
01179       fflush (calibrate->file_result);
01180 #if HAVE_MPI
01181     }
01182 #endif
01183 }
01184
01193 void
01194 calibrate_save_variables (unsigned int simulation, double error)
01195 {
01196   unsigned int i;
01197   char buffer[64];
01198 #if DEBUG
01199   fprintf (stderr, "calibrate_save_variables: start\n");
01200 #endif
01201   for (i = 0; i < calibrate->nvariables; ++i)
01202     {
01203       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01204       fprintf (calibrate->file_variables, buffer,
01205                calibrate->value[simulation * calibrate->nvariables + i]);
01206     }
01207   fprintf (calibrate->file_variables, "%.14le\n", error);
01208 #if DEBUG
01209   fprintf (stderr, "calibrate_save_variables: end\n");
01210 #endif
01211 }
01212
01221 void
01222 calibrate_best_thread (unsigned int simulation, double value)
01223 {
01224   unsigned int i, j;
01225   double e;
01226 #if DEBUG
01227   fprintf (stderr, "calibrate_best_thread: start\n");
01228 #endif
01229   if (calibrate->nsaveds < calibrate->nbest
```

```
01230       || value < calibrate->error_best[calibrate->nsaveds - 1])
01231     {
01232       g_mutex_lock (mutex);
01233       if (calibrate->nsaveds < calibrate->nbest)
01234         ++calibrate->nsaveds;
01235       calibrate->error_best[calibrate->nsaveds - 1] = value;
01236       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01237       for (i = calibrate->nsaveds; --i;)
01238         {
01239           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01240             {
01241               j = calibrate->simulation_best[i];
01242               e = calibrate->error_best[i];
01243               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01244               calibrate->error_best[i] = calibrate->error_best[i - 1];
01245               calibrate->simulation_best[i - 1] = j;
01246               calibrate->error_best[i - 1] = e;
01247             }
01248           else
01249             break;
01250         }
01251       g_mutex_unlock (mutex);
01252     }
01253 #if DEBUG
01254   fprintf (stderr, "calibrate_best_thread: end\n");
01255 #endif
01256 }
01257
01266 void
01267 calibrate_best_sequential (unsigned int simulation, double value)
01268 {
01269   unsigned int i, j;
01270   double e;
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_best_sequential: start\n");
01273 #endif
01274   if (calibrate->nsaveds < calibrate->nbest
01275       || value < calibrate->error_best[calibrate->nsaveds - 1])
01276     {
01277       if (calibrate->nsaveds < calibrate->nbest)
01278         ++calibrate->nsaveds;
01279       calibrate->error_best[calibrate->nsaveds - 1] = value;
01280       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01281       for (i = calibrate->nsaveds; --i;)
01282         {
01283           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01284             {
01285               j = calibrate->simulation_best[i];
01286               e = calibrate->error_best[i];
01287               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01288               calibrate->error_best[i] = calibrate->error_best[i - 1];
01289               calibrate->simulation_best[i - 1] = j;
01290               calibrate->error_best[i - 1] = e;
01291             }
01292           else
01293             break;
01294         }
01295     }
01296 #if DEBUG
01297   fprintf (stderr, "calibrate_best_sequential: end\n");
01298 #endif
01299 }
01300
01308 void *
01309 calibrate_thread (ParallelData * data)
01310 {
01311   unsigned int i, j, thread;
01312   double e;
01313 #if DEBUG
01314   fprintf (stderr, "calibrate_thread: start\n");
01315 #endif
01316   thread = data->thread;
01317 #if DEBUG
01318   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01319            calibrate->thread[thread], calibrate->thread[thread + 1]);
01320 #endif
01321   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01322     {
01323       e = 0.;
01324       for (j = 0; j < calibrate->nexperiments; ++j)
01325         e += calibrate_parse (i, j);
01326       calibrate_best_thread (i, e);
01327       g_mutex_lock (mutex);
01328       calibrate_save_variables (i, e);
01329       g_mutex_unlock (mutex);
```

```
01330 #if DEBUG
01331       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01332 #endif
01333     }
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_thread: end\n");
01336 #endif
01337   g_thread_exit (NULL);
01338   return NULL;
01339 }
01340
01345 void
01346 calibrate_sequential ()
01347 {
01348   unsigned int i, j;
01349   double e;
01350 #if DEBUG
01351   fprintf (stderr, "calibrate_sequential: start\n");
01352   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01353            calibrate->nstart, calibrate->nend);
01354 #endif
01355   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01356     {
01357       e = 0.;
01358       for (j = 0; j < calibrate->nexperiments; ++j)
01359         e += calibrate_parse (i, j);
01360       calibrate_best_sequential (i, e);
01361       calibrate_save_variables (i, e);
01362 #if DEBUG
01363       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01364 #endif
01365     }
01366 #if DEBUG
01367   fprintf (stderr, "calibrate_sequential: end\n");
01368 #endif
01369 }
01370
01382 void
01383 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01384                  double *error_best)
01385 {
01386   unsigned int i, j, k, s[calibrate->nbest];
01387   double e[calibrate->nbest];
01388 #if DEBUG
01389   fprintf (stderr, "calibrate_merge: start\n");
01390 #endif
01391   i = j = k = 0;
01392   do
01393     {
01394       if (i == calibrate->nsaveds)
01395         {
01396           s[k] = simulation_best[j];
01397           e[k] = error_best[j];
01398           ++j;
01399           ++k;
01400           if (j == nsaveds)
01401             break;
01402         }
01403       else if (j == nsaveds)
01404         {
01405           s[k] = calibrate->simulation_best[i];
01406           e[k] = calibrate->error_best[i];
01407           ++i;
01408           ++k;
01409           if (i == calibrate->nsaveds)
01410             break;
01411         }
01412       else if (calibrate->error_best[i] > error_best[j])
01413         {
01414           s[k] = simulation_best[j];
01415           e[k] = error_best[j];
01416           ++j;
01417           ++k;
01418         }
01419       else
01420         {
01421           s[k] = calibrate->simulation_best[i];
01422           e[k] = calibrate->error_best[i];
01423           ++i;
01424           ++k;
01425         }
01426     }
01427   while (k < calibrate->nbest);
01428   calibrate->nsaveds = k;
01429   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01430   memcpy (calibrate->error_best, e, k * sizeof (double));
01431 #if DEBUG
```

```
01432   fprintf (stderr, "calibrate_merge: end\n");
01433 #endif
01434 }
01435
01440 #if HAVE_MPI
01441 void
01442 calibrate_synchronise ()
01443 {
01444   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01445   double error_best[calibrate->nbest];
01446   MPI_Status mpi_stat;
01447 #if DEBUG
01448   fprintf (stderr, "calibrate_synchronise: start\n");
01449 #endif
01450   if (calibrate->mpi_rank == 0)
01451     {
01452       for (i = 1; i < ntasks; ++i)
01453         {
01454           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01455           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01456                     MPI_COMM_WORLD, &mpi_stat);
01457           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01458                     MPI_COMM_WORLD, &mpi_stat);
01459           calibrate_merge (nsaveds, simulation_best, error_best);
01460         }
01461     }
01462   else
01463     {
01464       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01465       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01466                 MPI_COMM_WORLD);
01467       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01468                 MPI_COMM_WORLD);
01469     }
01470 #if DEBUG
01471   fprintf (stderr, "calibrate_synchronise: end\n");
01472 #endif
01473 }
01474 #endif
01475
01480 void
01481 calibrate_sweep ()
01482 {
01483   unsigned int i, j, k, l;
01484   double e;
01485   GThread *thread[nthreads];
01486   ParallelData data[nthreads];
01487 #if DEBUG
01488   fprintf (stderr, "calibrate_sweep: start\n");
01489 #endif
01490   for (i = 0; i < calibrate->nsimulations; ++i)
01491     {
01492       k = i;
01493       for (j = 0; j < calibrate->nvariables; ++j)
01494         {
01495           l = k % calibrate->nsweeps[j];
01496           k /= calibrate->nsweeps[j];
01497           e = calibrate->rangemin[j];
01498           if (calibrate->nsweeps[j] > 1)
01499             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01500               / (calibrate->nsweeps[j] - 1);
01501           calibrate->value[i * calibrate->nvariables + j] = e;
01502         }
01503     }
01504   calibrate->nsaveds = 0;
01505   if (nthreads <= 1)
01506     calibrate_sequential ();
01507   else
01508     {
01509       for (i = 0; i < nthreads; ++i)
01510         {
01511           data[i].thread = i;
01512           thread[i]
01513             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01514         }
01515       for (i = 0; i < nthreads; ++i)
01516         g_thread_join (thread[i]);
01517     }
01518 #if HAVE_MPI
01519   // Communicating tasks results
01520   calibrate_synchronise ();
01521 #endif
01522 #if DEBUG
01523   fprintf (stderr, "calibrate_sweep: end\n");
01524 #endif
01525 }
01526
```

```
01531 void
01532 calibrate_MonteCarlo ()
01533 {
01534   unsigned int i, j;
01535   GThread *thread[nthreads];
01536   ParallelData data[nthreads];
01537 #if DEBUG
01538   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01539 #endif
01540   for (i = 0; i < calibrate->nsimulations; ++i)
01541     for (j = 0; j < calibrate->nvariables; ++j)
01542       calibrate->value[i * calibrate->nvariables + j]
01543         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01544         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01545   calibrate->nsaveds = 0;
01546   if (nthreads <= 1)
01547     calibrate_sequential ();
01548   else
01549     {
01550       for (i = 0; i < nthreads; ++i)
01551         {
01552           data[i].thread = i;
01553           thread[i]
01554             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01555         }
01556       for (i = 0; i < nthreads; ++i)
01557         g_thread_join (thread[i]);
01558     }
01559 #if HAVE_MPI
01560   // Communicating tasks results
01561   calibrate_synchronise ();
01562 #endif
01563 #if DEBUG
01564   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01565 #endif
01566 }
01567
01575 double
01576 calibrate_genetic_objective (Entity * entity)
01577 {
01578   unsigned int j;
01579   double objective;
01580   char buffer[64];
01581 #if DEBUG
01582   fprintf (stderr, "calibrate_genetic_objective: start\n");
01583 #endif
01584   for (j = 0; j < calibrate->nvariables; ++j)
01585     {
01586       calibrate->value[entity->id * calibrate->nvariables + j]
01587         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01588     }
01589   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01590     objective += calibrate_parse (entity->id, j);
01591   g_mutex_lock (mutex);
01592   for (j = 0; j < calibrate->nvariables; ++j)
01593     {
01594       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01595       fprintf (calibrate->file_variables, buffer,
01596                genetic_get_variable (entity, calibrate->genetic_variable + j));
01597     }
01598   fprintf (calibrate->file_variables, "%.14le\n", objective);
01599   g_mutex_unlock (mutex);
01600 #if DEBUG
01601   fprintf (stderr, "calibrate_genetic_objective: end\n");
01602 #endif
01603   return objective;
01604 }
01605
01610 void
01611 calibrate_genetic ()
01612 {
01613   char *best_genome;
01614   double best_objective, *best_variable;
01615 #if DEBUG
01616   fprintf (stderr, "calibrate_genetic: start\n");
01617   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
01618            nthreads);
01619   fprintf (stderr,
01620            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
01621            calibrate->nvariables, calibrate->nsimulations,
01622            calibrate->niterations);
01623   fprintf (stderr,
01624            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01625            calibrate->mutation_ratio, calibrate->
      reproduction_ratio,
01626            calibrate->adaptation_ratio);
01627 #endif
```

```
01628   genetic_algorithm_default (calibrate->nvariables,
01629                              calibrate->genetic_variable,
01630                              calibrate->nsimulations,
01631                              calibrate->niterations,
01632                              calibrate->mutation_ratio,
01633                              calibrate->reproduction_ratio,
01634                              calibrate->adaptation_ratio,
01635                              &calibrate_genetic_objective,
01636                              &best_genome, &best_variable, &best_objective);
01637 #if DEBUG
01638   fprintf (stderr, "calibrate_genetic: the best\n");
01639 #endif
01640   calibrate->error_old = (double *) g_malloc (sizeof (double));
01641   calibrate->value_old
01642     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
01643   calibrate->error_old[0] = best_objective;
01644   memcpy (calibrate->value_old, best_variable,
01645           calibrate->nvariables * sizeof (double));
01646   g_free (best_genome);
01647   g_free (best_variable);
01648   calibrate_print ();
01649 #if DEBUG
01650   fprintf (stderr, "calibrate_genetic: end\n");
01651 #endif
01652 }
01653
01658 void
01659 calibrate_save_old ()
01660 {
01661   unsigned int i, j;
01662 #if DEBUG
01663   fprintf (stderr, "calibrate_save_old: start\n");
01664 #endif
01665   memcpy (calibrate->error_old, calibrate->error_best,
01666           calibrate->nbest * sizeof (double));
01667   for (i = 0; i < calibrate->nbest; ++i)
01668     {
01669       j = calibrate->simulation_best[i];
01670       memcpy (calibrate->value_old + i * calibrate->nvariables,
01671               calibrate->value + j * calibrate->nvariables,
01672               calibrate->nvariables * sizeof (double));
01673     }
01674 #if DEBUG
01675   for (i = 0; i < calibrate->nvariables; ++i)
01676     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
01677              i, calibrate->value_old[i]);
01678   fprintf (stderr, "calibrate_save_old: end\n");
01679 #endif
01680 }
01681
01687 void
01688 calibrate_merge_old ()
01689 {
01690   unsigned int i, j, k;
01691   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
01692     *enew, *eold;
01693 #if DEBUG
01694   fprintf (stderr, "calibrate_merge_old: start\n");
01695 #endif
01696   enew = calibrate->error_best;
01697   eold = calibrate->error_old;
01698   i = j = k = 0;
01699   do
01700     {
01701       if (*enew < *eold)
01702         {
01703           memcpy (v + k * calibrate->nvariables,
01704                   calibrate->value
01705                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
01706                   calibrate->nvariables * sizeof (double));
01707           e[k] = *enew;
01708           ++k;
01709           ++enew;
01710           ++i;
01711         }
01712       else
01713         {
01714           memcpy (v + k * calibrate->nvariables,
01715                   calibrate->value_old + j * calibrate->nvariables,
01716                   calibrate->nvariables * sizeof (double));
01717           e[k] = *eold;
01718           ++k;
01719           ++eold;
01720           ++j;
01721         }
```

```
01722     }
01723   while (k < calibrate->nbest);
01724   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
01725   memcpy (calibrate->error_old, e, k * sizeof (double));
01726 #if DEBUG
01727   fprintf (stderr, "calibrate_merge_old: end\n");
01728 #endif
01729 }
01730
01736 void
01737 calibrate_refine ()
01738 {
01739   unsigned int i, j;
01740   double d;
01741 #if HAVE_MPI
01742   MPI_Status mpi_stat;
01743 #endif
01744 #if DEBUG
01745   fprintf (stderr, "calibrate_refine: start\n");
01746 #endif
01747 #if HAVE_MPI
01748   if (!calibrate->mpi_rank)
01749     {
01750 #endif
01751       for (j = 0; j < calibrate->nvariables; ++j)
01752         {
01753           calibrate->rangemin[j] = calibrate->rangemax[j]
01754             = calibrate->value_old[j];
01755         }
01756       for (i = 0; ++i < calibrate->nbest;)
01757         {
01758           for (j = 0; j < calibrate->nvariables; ++j)
01759             {
01760               calibrate->rangemin[j]
01761                 = fmin (calibrate->rangemin[j],
01762                         calibrate->value_old[i * calibrate->nvariables + j]);
01763               calibrate->rangemax[j]
01764                 = fmax (calibrate->rangemax[j],
01765                         calibrate->value_old[i * calibrate->nvariables + j]);
01766             }
01767         }
01768       for (j = 0; j < calibrate->nvariables; ++j)
01769         {
01770           d = 0.5 * calibrate->tolerance
01771             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01772           calibrate->rangemin[j] -= d;
01773           calibrate->rangemin[j]
01774             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
01775           calibrate->rangemax[j] += d;
01776           calibrate->rangemax[j]
01777             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
01778           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
01779                   calibrate->rangemin[j], calibrate->rangemax[j]);
01780           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
01781                    calibrate->label[j], calibrate->rangemin[j],
01782                    calibrate->rangemax[j]);
01783         }
01784 #if HAVE_MPI
01785       for (i = 1; i < ntasks; ++i)
01786         {
01787           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
01788                     1, MPI_COMM_WORLD);
01789           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
01790                     1, MPI_COMM_WORLD);
01791         }
01792     }
01793   else
01794     {
01795       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01796                 MPI_COMM_WORLD, &mpi_stat);
01797       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01798                 MPI_COMM_WORLD, &mpi_stat);
01799     }
01800 #endif
01801 #if DEBUG
01802   fprintf (stderr, "calibrate_refine: end\n");
01803 #endif
01804 }
01805
01810 void
01811 calibrate_iterate ()
01812 {
01813   unsigned int i;
01814 #if DEBUG
01815   fprintf (stderr, "calibrate_iterate: start\n");
01816 #endif
01817   calibrate->error_old
```

```
01818     = (double *) g_malloc (calibrate->nbest * sizeof (double));
01819   calibrate->value_old = (double *)
01820     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
01821   calibrate_step ();
01822   calibrate_save_old ();
01823   calibrate_refine ();
01824   calibrate_print ();
01825   for (i = 1; i < calibrate->niterations; ++i)
01826     {
01827       calibrate_step ();
01828       calibrate_merge_old ();
01829       calibrate_refine ();
01830       calibrate_print ();
01831     }
01832 #if DEBUG
01833   fprintf (stderr, "calibrate_iterate: end\n");
01834 #endif
01835 }
01836
01841 void
01842 calibrate_free ()
01843 {
01844   unsigned int i, j;
01845 #if DEBUG
01846   fprintf (stderr, "calibrate_free: start\n");
01847 #endif
01848   for (i = 0; i < calibrate->nexperiments; ++i)
01849     {
01850       for (j = 0; j < calibrate->ninputs; ++j)
01851         g_mapped_file_unref (calibrate->file[j][i]);
01852     }
01853   for (i = 0; i < calibrate->ninputs; ++i)
01854     g_free (calibrate->file[i]);
01855   g_free (calibrate->error_old);
01856   g_free (calibrate->value_old);
01857   g_free (calibrate->value);
01858   g_free (calibrate->genetic_variable);
01859 #if DEBUG
01860   fprintf (stderr, "calibrate_free: end\n");
01861 #endif
01862 }
01863
01868 void
01869 calibrate_new ()
01870 {
01871   unsigned int i, j, *nbits;
01872
01873 #if DEBUG
01874   fprintf (stderr, "calibrate_new: start\n");
01875 #endif
01876
01877   // Replacing the working dir
01878   chdir (input->directory);
01879
01880   // Obtaining the simulator file
01881   calibrate->simulator = input->simulator;
01882
01883   // Obtaining the evaluator file
01884   calibrate->evaluator = input->evaluator;
01885
01886   // Reading the algorithm
01887   calibrate->algorithm = input->algorithm;
01888   switch (calibrate->algorithm)
01889     {
01890     case ALGORITHM_MONTE_CARLO:
01891       calibrate_step = calibrate_MonteCarlo;
01892       break;
01893     case ALGORITHM_SWEEP:
01894       calibrate_step = calibrate_sweep;
01895       break;
01896     default:
01897       calibrate_step = calibrate_genetic;
01898       calibrate->mutation_ratio = input->mutation_ratio;
01899       calibrate->reproduction_ratio = input->
      reproduction_ratio;
01900       calibrate->adaptation_ratio = input->adaptation_ratio;
01901     }
01902   calibrate->nsimulations = input->nsimulations;
01903   calibrate->niterations = input->niterations;
01904   calibrate->nbest = input->nbest;
01905   calibrate->tolerance = input->tolerance;
01906
01907   calibrate->simulation_best
01908     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
01909   calibrate->error_best
01910     = (double *) alloca (calibrate->nbest * sizeof (double));
01911
```

```
01912   // Reading the experimental data
01913 #if DEBUG
01914   fprintf (stderr, "calibrate_new: current directory=%s\n",
01915            g_get_current_dir ());
01916 #endif
01917   calibrate->nexperiments = input->nexperiments;
01918   calibrate->ninputs = input->ninputs;
01919   calibrate->experiment = input->experiment;
01920   calibrate->weight = input->weight;
01921   for (i = 0; i < input->ninputs; ++i)
01922     {
01923       calibrate->template[i] = input->template[i];
01924       calibrate->file[i]
01925         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
01926     }
01927   for (i = 0; i < input->nexperiments; ++i)
01928     {
01929 #if DEBUG
01930       fprintf (stderr, "calibrate_new: i=%u\n", i);
01931       fprintf (stderr, "calibrate_new: experiment=%s\n",
01932                calibrate->experiment[i]);
01933       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
01934 #endif
01935       for (j = 0; j < input->ninputs; ++j)
01936         {
01937 #if DEBUG
01938           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
01939           fprintf (stderr, "calibrate_new: experiment=%u template%u=%s\n",
01940                    i, j + 1, calibrate->template[j][i]);
01941 #endif
01942           calibrate->file[j][i]
01943             = g_mapped_file_new (input->template[j][i], 0, NULL);
01944         }
01945     }
01946
01947   // Reading the variables data
01948 #if DEBUG
01949   fprintf (stderr, "calibrate_new: reading variables\n");
01950 #endif
01951   calibrate->nvariables = input->nvariables;
01952   calibrate->label = input->label;
01953   calibrate->rangemin = input->rangemin;
01954   calibrate->rangeminabs = input->rangeminabs;
01955   calibrate->rangemax = input->rangemax;
01956   calibrate->rangemaxabs = input->rangemaxabs;
01957   calibrate->precision = input->precision;
01958   calibrate->nsweeps = input->nsweeps;
01959   nbits = input->nbits;
01960   if (input->algorithm == ALGORITHM_SWEEP)
01961     calibrate->nsimulations = 1;
01962   else if (input->algorithm == ALGORITHM_GENETIC)
01963     for (i = 0; i < input->nvariables; ++i)
01964       {
01965         if (calibrate->algorithm == ALGORITHM_SWEEP)
01966           {
01967             calibrate->nsimulations *= input->nsweeps[i];
01968 #if DEBUG
01969             fprintf (stderr, "calibrate_new: nsweeps=%u nsimulations=%u\n",
01970                      calibrate->nsweeps[i], calibrate->nsimulations);
01971 #endif
01972           }
01973       }
01974
01975   // Allocating values
01976 #if DEBUG
01977   fprintf (stderr, "calibrate_new: allocating variables\n");
01978   fprintf (stderr, "calibrate_new: nvariables=%u\n", calibrate->nvariables);
01979 #endif
01980   calibrate->genetic_variable = NULL;
01981   if (calibrate->algorithm == ALGORITHM_GENETIC)
01982     {
01983       calibrate->genetic_variable = (GeneticVariable *)
01984         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
01985       for (i = 0; i < calibrate->nvariables; ++i)
01986         {
01987 #if DEBUG
01988           fprintf (stderr, "calibrate_new: i=%u min=%lg max=%lg nbits=%u\n",
01989                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
01990 #endif
01991           calibrate->genetic_variable[i].minimum = calibrate->
      rangemin[i];
01992           calibrate->genetic_variable[i].maximum = calibrate->
      rangemax[i];
01993           calibrate->genetic_variable[i].nbits = nbits[i];
01994         }
01995     }
01996 #if DEBUG
```

```
01997   fprintf (stderr, "calibrate_new: nvariables=%u nsimulations=%u\n",
01998            calibrate->nvariables, calibrate->nsimulations);
01999 #endif
02000   calibrate->value = (double *) g_malloc (calibrate->nsimulations *
02001                                           calibrate->nvariables *
02002                                           sizeof (double));
02003
02004   // Calculating simulations to perform on each task
02005 #if HAVE_MPI
02006 #if DEBUG
02007   fprintf (stderr, "calibrate_new: rank=%u ntasks=%u\n",
02008            calibrate->mpi_rank, ntasks);
02009 #endif
02010   calibrate->nstart = calibrate->mpi_rank * calibrate->
      nsimulations / ntasks;
02011   calibrate->nend = (1 + calibrate->mpi_rank) * calibrate->
      nsimulations
02012     / ntasks;
02013 #else
02014   calibrate->nstart = 0;
02015   calibrate->nend = calibrate->nsimulations;
02016 #endif
02017 #if DEBUG
02018   fprintf (stderr, "calibrate_new: nstart=%u nend=%u\n", calibrate->nstart,
02019            calibrate->nend);
02020 #endif
02021
02022   // Calculating simulations to perform on each thread
02023   calibrate->thread
02024     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02025   for (i = 0; i <= nthreads; ++i)
02026     {
02027       calibrate->thread[i] = calibrate->nstart
02028         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02029 #if DEBUG
02030       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02031                calibrate->thread[i]);
02032 #endif
02033     }
02034
02035   // Opening result files
02036   calibrate->file_result = fopen ("result", "w");
02037   calibrate->file_variables = fopen ("variables", "w");
02038
02039   // Performing the algorithm
02040   switch (calibrate->algorithm)
02041     {
02042       // Genetic algorithm
02043     case ALGORITHM_GENETIC:
02044       calibrate_genetic ();
02045       break;
02046
02047       // Iterative algorithm
02048     default:
02049       calibrate_iterate ();
02050     }
02051
02052   // Closing result files
02053   fclose (calibrate->file_variables);
02054   fclose (calibrate->file_result);
02055
02056 #if DEBUG
02057   fprintf (stderr, "calibrate_new: end\n");
02058 #endif
02059 }
02060
02061 #if HAVE_GTK
02062
02069 void
02070 input_save (char *filename)
02071 {
02072   unsigned int i, j;
02073   char *buffer;
02074   xmlDoc *doc;
02075   xmlNode *node, *child;
02076   GFile *file, *file2;
02077
02078   // Getting the input file directory
02079   input->name = g_path_get_basename (filename);
02080   input->directory = g_path_get_dirname (filename);
02081   file = g_file_new_for_path (input->directory);
02082
02083   // Opening the input file
02084   doc = xmlNewDoc ((const xmlChar *) "1.0");
02085
02086   // Setting root XML node
02087   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
```

```
02088   xmlDocSetRootElement (doc, node);
02089
02090   // Adding properties to the root XML node
02091   file2 = g_file_new_for_path (input->simulator);
02092   buffer = g_file_get_relative_path (file, file2);
02093   g_object_unref (file2);
02094   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02095   g_free (buffer);
02096   if (input->evaluator)
02097     {
02098       file2 = g_file_new_for_path (input->evaluator);
02099       buffer = g_file_get_relative_path (file, file2);
02100       g_object_unref (file2);
02101       if (xmlStrlen ((xmlChar *) buffer))
02102         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02103       g_free (buffer);
02104     }
02105
02106   // Setting the algorithm
02107   buffer = (char *) g_malloc (64);
02108   switch (input->algorithm)
02109     {
02110     case ALGORITHM_MONTE_CARLO:
02111       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02112       snprintf (buffer, 64, "%u", input->nsimulations);
02113       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02114       snprintf (buffer, 64, "%u", input->niterations);
02115       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02116       snprintf (buffer, 64, "%.3lg", input->tolerance);
02117       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02118       snprintf (buffer, 64, "%u", input->nbest);
02119       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02120       break;
02121     case ALGORITHM_SWEEP:
02122       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02123       snprintf (buffer, 64, "%u", input->niterations);
02124       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%.3lg", input->tolerance);
02126       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02127       snprintf (buffer, 64, "%u", input->nbest);
02128       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02129       break;
02130     default:
02131       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02132       snprintf (buffer, 64, "%u", input->nsimulations);
02133       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->niterations);
02135       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02136       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02137       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02138       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02139       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02140       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02141       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02142       break;
02143     }
02144   g_free (buffer);
02145
02146   // Setting the experimental data
02147   for (i = 0; i < input->nexperiments; ++i)
02148     {
02149       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02150       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02151       if (input->weight[i] != 1.)
02152         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02153       for (j = 0; j < input->ninputs; ++j)
02154         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02155     }
02156
02157   // Setting the variables data
02158   for (i = 0; i < input->nvariables; ++i)
02159     {
02160       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02161       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02162       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02163       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02164         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02165       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02166       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02167         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02168       if (input->precision[i] != DEFAULT_PRECISION)
02169         xml_node_set_uint (child, XML_PRECISION, input->
```

```
      precision[i]);
02170       if (input->algorithm == ALGORITHM_SWEEP)
02171         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02172       else if (input->algorithm == ALGORITHM_GENETIC)
02173         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02174     }
02175
02176   // Saving the XML file
02177   xmlSaveFormatFile (filename, doc, 1);
02178
02179   // Freeing memory
02180   xmlFreeDoc (doc);
02181 }
02182
02187 void
02188 options_new ()
02189 {
02190   options->label_processors
02191     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02192   options->spin_processors
02193     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02194   gtk_spin_button_set_value (options->spin_processors, (gdouble)
      nthreads);
02195   options->grid = (GtkGrid *) gtk_grid_new ();
02196   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02197                    0, 0, 1, 1);
02198   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02199                    1, 0, 1, 1);
02200   gtk_widget_show_all (GTK_WIDGET (options->grid));
02201   options->dialog = (GtkDialog *)
02202     gtk_dialog_new_with_buttons (gettext ("Options"),
02203                                  window->window,
02204                                  GTK_DIALOG_MODAL,
02205                                  gettext ("_OK"), GTK_RESPONSE_OK,
02206                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02207                                  NULL);
02208   gtk_container_add
02209     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02210      GTK_WIDGET (options->grid));
02211   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02212     nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02213   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02214 }
02215
02220 void
02221 running_new ()
02222 {
02223   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02224   running->dialog = (GtkDialog *)
02225     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02226                                  window->window,
02227                                  GTK_DIALOG_DESTROY_WITH_PARENT, NULL, NULL);
02228   gtk_container_add
02229     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02230      GTK_WIDGET (running->label));
02231   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02232 }
02233
02239 int
02240 window_save ()
02241 {
02242   char *buffer;
02243   GtkFileChooserDialog *dlg;
02244
02245 #if DEBUG
02246   fprintf (stderr, "window_save: start\n");
02247 #endif
02248
02249   // Opening the saving dialog
02250   dlg = (GtkFileChooserDialog *)
02251     gtk_file_chooser_dialog_new (gettext ("Save file"),
02252                                  window->window,
02253                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02254                                  gettext ("_Cancel"),
02255                                  GTK_RESPONSE_CANCEL,
02256                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02257   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02258
02259   // If OK response then saving
02260   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02261     {
02262
02263       // Adding properties to the root XML node
02264       input->simulator = gtk_file_chooser_get_filename
02265         (GTK_FILE_CHOOSER (window->button_simulator));
```

```
02266       if (gtk_toggle_button_get_active
02267           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02268         input->evaluator = gtk_file_chooser_get_filename
02269           (GTK_FILE_CHOOSER (window->button_evaluator));
02270       else
02271         input->evaluator = NULL;
02272
02273       // Setting the algorithm
02274       switch (window_get_algorithm ())
02275         {
02276         case ALGORITHM_MONTE_CARLO:
02277           input->algorithm = ALGORITHM_MONTE_CARLO;
02278           input->nsimulations
02279             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02280           input->niterations
02281             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02282           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02283           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02284           break;
02285         case ALGORITHM_SWEEP:
02286           input->algorithm = ALGORITHM_SWEEP;
02287           input->niterations
02288             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02289           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02290           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02291           break;
02292         default:
02293           input->algorithm = ALGORITHM_GENETIC;
02294           input->nsimulations
02295             = gtk_spin_button_get_value_as_int (window->spin_population);
02296           input->niterations
02297             = gtk_spin_button_get_value_as_int (window->spin_generations);
02298           input->mutation_ratio
02299             = gtk_spin_button_get_value (window->spin_mutation);
02300           input->reproduction_ratio
02301             = gtk_spin_button_get_value (window->spin_reproduction);
02302           input->adaptation_ratio
02303             = gtk_spin_button_get_value (window->spin_adaptation);
02304           break;
02305         }
02306
02307       // Saving the XML file
02308       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02309       input_save (buffer);
02310
02311       // Closing and freeing memory
02312       g_free (buffer);
02313       gtk_widget_destroy (GTK_WIDGET (dlg));
02314 #if DEBUG
02315       fprintf (stderr, "window_save: end\n");
02316 #endif
02317       return 1;
02318     }
02319
02320   // Closing and freeing memory
02321   gtk_widget_destroy (GTK_WIDGET (dlg));
02322 #if DEBUG
02323   fprintf (stderr, "window_save: end\n");
02324 #endif
02325   return 0;
02326 }
02327
02332 void
02333 window_run ()
02334 {
02335   unsigned int i;
02336   char *msg, *msg2, buffer[64], buffer2[64];
02337 #if DEBUG
02338   fprintf (stderr, "window_run: start\n");
02339 #endif
02340   if (!window_save ())
02341     {
02342 #if DEBUG
02343       fprintf (stderr, "window_run: end\n");
02344 #endif
02345       return;
02346     }
02347   running_new ();
02348   while (g_main_context_pending (NULL))
02349     g_main_context_iteration (NULL, FALSE);
02350   calibrate_new ();
02351   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02352   snprintf (buffer, 64, "error=%.15le\n", calibrate->error_old[0]);
```

```
02353   msg2 = g_strdup (buffer);
02354   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02355     {
02356       snprintf (buffer, 64, "%s=%s\n",
02357                 calibrate->label[i], format[calibrate->precision[i]]);
02358       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02359       msg = g_strconcat (msg2, buffer2, NULL);
02360       g_free (msg2);
02361     }
02362   show_message (gettext ("Best result"), msg2, INFO_TYPE);
02363   g_free (msg2);
02364   calibrate_free ();
02365 #if DEBUG
02366   fprintf (stderr, "window_run: end\n");
02367 #endif
02368 }
02369
02374 void
02375 window_help ()
02376 {
02377   char *buffer, *buffer2;
02378   buffer2 = g_build_filename (current_directory, "manuals",
02379                               "user-manual.pdf", NULL);
02380   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02381   g_free (buffer2);
02382   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02383   g_free (buffer);
02384 }
02385
02390 void
02391 window_about ()
02392 {
02393   gchar *authors[] = {
02394     "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02395     "Borja Latorre Garcés (borja.latorre@csic.es)",
02396     NULL
02397   };
02398   gtk_show_about_dialog (window->window,
02399                          "program_name",
02400                          "Calibrator",
02401                          "comments",
02402                          gettext ("A software to make calibrations of "
02403                                   "empirical parameters"),
02404                          "authors", authors,
02405                          "translator-credits",
02406                          "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02407                          "version", "1.1.26", "copyright",
02408                          "Copyright 2012-2015 Javier Burguete Tolosa",
02409                          "logo", window->logo,
02410                          "website-label", gettext ("Website"),
02411                          "website",
02412                          "https://github.com/jburguete/calibrator", NULL);
02413 }
02414
02420 int
02421 window_get_algorithm ()
02422 {
02423   unsigned int i;
02424   for (i = 0; i < NALGORITHMS; ++i)
02425     if (gtk_toggle_button_get_active
02426         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02427       break;
02428   return i;
02429 }
02430
02435 void
02436 window_update ()
02437 {
02438   unsigned int i;
02439   gtk_widget_set_sensitive
02440     (GTK_WIDGET (window->button_evaluator),
02441      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02442                                    (window->check_evaluator)));
02443   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02444   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02445   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02446   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02447   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02448   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
02449   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02450   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02451   gtk_widget_hide (GTK_WIDGET (window->label_population));
02452   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02453   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02454   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02455   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02456   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
```

```
02457   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02458   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
02459   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
02460   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
02461   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
02462   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
02463   gtk_widget_hide (GTK_WIDGET (window->label_bits));
02464   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
02465   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
02466   switch (window_get_algorithm ())
02467     {
02468     case ALGORITHM_MONTE_CARLO:
02469       gtk_widget_show (GTK_WIDGET (window->label_simulations));
02470       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
02471       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02472       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02473       if (i > 1)
02474         {
02475           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02476           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02477           gtk_widget_show (GTK_WIDGET (window->label_bests));
02478           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02479         }
02480       break;
02481     case ALGORITHM_SWEEP:
02482       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02483       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02484       gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02485       gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02486       if (i > 1)
02487         {
02488           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02489           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02490           gtk_widget_show (GTK_WIDGET (window->label_bests));
02491           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02492         }
02493       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
02494       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
02495       break;
02496     default:
02497       gtk_widget_show (GTK_WIDGET (window->label_population));
02498       gtk_widget_show (GTK_WIDGET (window->spin_population));
02499       gtk_widget_show (GTK_WIDGET (window->label_generations));
02500       gtk_widget_show (GTK_WIDGET (window->spin_generations));
02501       gtk_widget_show (GTK_WIDGET (window->label_mutation));
02502       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
02503       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
02504       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
02505       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
02506       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
02507       gtk_widget_show (GTK_WIDGET (window->label_bits));
02508       gtk_widget_show (GTK_WIDGET (window->spin_bits));
02509     }
02510   gtk_widget_set_sensitive
02511     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
02512   gtk_widget_set_sensitive
02513     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
02514   for (i = 0; i < input->ninputs; ++i)
02515     {
02516       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02517       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02518       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
02519       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
02520       g_signal_handler_block
02521         (window->check_template[i], window->id_template[i]);
02522       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
02523       gtk_toggle_button_set_active
02524         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
02525       g_signal_handler_unblock
02526         (window->button_template[i], window->id_input[i]);
02527       g_signal_handler_unblock
02528         (window->check_template[i], window->id_template[i]);
02529     }
02530   if (i > 0)
02531     {
02532       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
02533       gtk_widget_set_sensitive
02534         (GTK_WIDGET (window->button_template[i - 1]),
02535          gtk_toggle_button_get_active
02536          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
02537     }
02538   if (i < MAX_NINPUTS)
02539     {
02540       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
```

```
02541       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02542       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
02543       gtk_widget_set_sensitive
02544         (GTK_WIDGET (window->button_template[i]),
02545          gtk_toggle_button_get_active
02546          GTK_TOGGLE_BUTTON (window->check_template[i]));
02547       g_signal_handler_block
02548         (window->check_template[i], window->id_template[i]);
02549       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02550       gtk_toggle_button_set_active
02551         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02552       g_signal_handler_unblock
02553         (window->button_template[i], window->id_input[i]);
02554       g_signal_handler_unblock
02555         (window->check_template[i], window->id_template[i]);
02556     }
02557   while (++i < MAX_NINPUTS)
02558     {
02559       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02560       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02561     }
02562   gtk_widget_set_sensitive
02563     (GTK_WIDGET (window->spin_minabs),
02564      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02565   gtk_widget_set_sensitive
02566     (GTK_WIDGET (window->spin_maxabs),
02567      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02568 }
02569
02574 void
02575 window_set_algorithm ()
02576 {
02577   unsigned int i;
02578   i = window_get_algorithm ();
02579   switch (i)
02580     {
02581     case ALGORITHM_SWEEP:
02582       input->nsweeps = (unsigned int *) g_realloc
02583         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02584       break;
02585     case ALGORITHM_GENETIC:
02586       input->nbits = (unsigned int *) g_realloc
02587         (input->nbits, input->nvariables * sizeof (unsigned int));
02588     }
02589   window_update ();
02590 }
02591
02596 void
02597 window_set_experiment ()
02598 {
02599   unsigned int i, j;
02600   char *buffer1, *buffer2;
02601 #if DEBUG
02602   fprintf (stderr, "window_set_experiment: start\n");
02603 #endif
02604   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02605   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02606   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02607   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02608   g_free (buffer1);
02609   g_signal_handler_block
02610     (window->button_experiment, window->id_experiment_name);
02611   gtk_file_chooser_set_filename
02612     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
02613   g_signal_handler_unblock
02614     (window->button_experiment, window->id_experiment_name);
02615   g_free (buffer2);
02616   for (j = 0; j < input->ninputs; ++j)
02617     {
02618       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02619       buffer2
02620         = g_build_filename (input->directory, input->template[j][i], NULL);
02621       gtk_file_chooser_set_filename
02622         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02623       g_free (buffer2);
02624       g_signal_handler_unblock
02625         (window->button_template[j], window->id_input[j]);
02626     }
02627 #if DEBUG
02628   fprintf (stderr, "window_set_experiment: end\n");
02629 #endif
02630 }
02631
02636 void
02637 window_remove_experiment ()
```

```
02638 {
02639   unsigned int i, j;
02640   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02641   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02642   gtk_combo_box_text_remove (window->combo_experiment, i);
02643   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02644   xmlFree (input->experiment[i]);
02645   --input->nexperiments;
02646   for (j = i; j < input->nexperiments; ++j)
02647     {
02648       input->experiment[j] = input->experiment[j + 1];
02649       input->weight[j] = input->weight[j + 1];
02650     }
02651   j = input->nexperiments - 1;
02652   if (i > j)
02653     i = j;
02654   for (j = 0; j < input->ninputs; ++j)
02655     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02656   g_signal_handler_block
02657     (window->button_experiment, window->id_experiment_name);
02658   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02659   g_signal_handler_unblock
02660     (window->button_experiment, window->id_experiment_name);
02661   for (j = 0; j < input->ninputs; ++j)
02662     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02663   window_update ();
02664 }
02665
02670 void
02671 window_add_experiment ()
02672 {
02673   unsigned int i, j;
02674   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02675   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02676   gtk_combo_box_text_insert_text
02677     (window->combo_experiment, i, input->experiment[i]);
02678   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02679   input->experiment = (char **) g_realloc
02680     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02681   input->weight = (double *) g_realloc
02682     (input->weight, (input->nexperiments + 1) * sizeof (double));
02683   for (j = input->nexperiments - 1; j > i; --j)
02684     {
02685       input->experiment[j + 1] = input->experiment[j];
02686       input->weight[j + 1] = input->weight[j];
02687     }
02688   input->experiment[j + 1]
02689     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02690   input->weight[j + 1] = input->weight[j];
02691   ++input->nexperiments;
02692   for (j = 0; j < input->ninputs; ++j)
02693     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02694   g_signal_handler_block
02695     (window->button_experiment, window->id_experiment_name);
02696   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02697   g_signal_handler_unblock
02698     (window->button_experiment, window->id_experiment_name);
02699   for (j = 0; j < input->ninputs; ++j)
02700     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02701   window_update ();
02702 }
02703
02708 void
02709 window_name_experiment ()
02710 {
02711   unsigned int i;
02712   char *buffer;
02713   GFile *file1, *file2;
02714 #if DEBUG
02715   fprintf (stderr, "window_name_experiment: start\n");
02716 #endif
02717   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02718   file1
02719     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02720   file2 = g_file_new_for_path (input->directory);
02721   buffer = g_file_get_relative_path (file2, file1);
02722   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02723   gtk_combo_box_text_remove (window->combo_experiment, i);
```

```
02724   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02725   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02726   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02727   g_free (buffer);
02728   g_object_unref (file2);
02729   g_object_unref (file1);
02730 #if DEBUG
02731   fprintf (stderr, "window_name_experiment: end\n");
02732 #endif
02733 }
02734
02739 void
02740 window_weight_experiment ()
02741 {
02742   unsigned int i;
02743 #if DEBUG
02744   fprintf (stderr, "window_weight_experiment: start\n");
02745 #endif
02746   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02747   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02748 #if DEBUG
02749   fprintf (stderr, "window_weight_experiment: end\n");
02750 #endif
02751 }
02752
02758 void
02759 window_inputs_experiment ()
02760 {
02761   unsigned int j;
02762 #if DEBUG
02763   fprintf (stderr, "window_inputs_experiment: start\n");
02764 #endif
02765   j = input->ninputs - 1;
02766   if (j
02767       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02768                                         (window->check_template[j])))
02769     --input->ninputs;
02770   if (input->ninputs < MAX_NINPUTS
02771       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02772                                        (window->check_template[j])))
02773     {
02774       ++input->ninputs;
02775       for (j = 0; j < input->ninputs; ++j)
02776         {
02777           input->template[j] = (char **)
02778             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02779         }
02780     }
02781   window_update ();
02782 #if DEBUG
02783   fprintf (stderr, "window_inputs_experiment: end\n");
02784 #endif
02785 }
02786
02794 void
02795 window_template_experiment (void *data)
02796 {
02797   unsigned int i, j;
02798   char *buffer;
02799   GFile *file1, *file2;
02800 #if DEBUG
02801   fprintf (stderr, "window_template_experiment: start\n");
02802 #endif
02803   i = (size_t) data;
02804   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02805   file1
02806     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02807   file2 = g_file_new_for_path (input->directory);
02808   buffer = g_file_get_relative_path (file2, file1);
02809   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02810   g_free (buffer);
02811   g_object_unref (file2);
02812   g_object_unref (file1);
02813 #if DEBUG
02814   fprintf (stderr, "window_template_experiment: end\n");
02815 #endif
02816 }
02817
02822 void
02823 window_set_variable ()
02824 {
02825   unsigned int i;
02826 #if DEBUG
02827   fprintf (stderr, "window_set_variable: start\n");
02828 #endif
02829   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
```

```
02830   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02831   gtk_entry_set_text (window->entry_variable, input->label[i]);
02832   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02833   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02834   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
02835   if (input->rangeminabs[i] != -G_MAXDOUBLE)
02836     {
02837       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
02838       gtk_toggle_button_set_active
02839         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
02840     }
02841   else
02842     {
02843       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
02844       gtk_toggle_button_set_active
02845         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
02846     }
02847   if (input->rangemaxabs[i] != G_MAXDOUBLE)
02848     {
02849       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
02850       gtk_toggle_button_set_active
02851         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
02852     }
02853   else
02854     {
02855       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
02856       gtk_toggle_button_set_active
02857         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
02858     }
02859   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
02860   switch (input->algorithm)
02861     {
02862     case ALGORITHM_SWEEP:
02863       gtk_spin_button_set_value (window->spin_sweeps,
02864                                  (gdouble) input->nsweeps[i]);
02865       break;
02866     case ALGORITHM_GENETIC:
02867       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02868       break;
02869     }
02870   window_update ();
02871 #if DEBUG
02872   fprintf (stderr, "window_set_variable: end\n");
02873 #endif
02874 }
02875
02880 void
02881 window_remove_variable ()
02882 {
02883   unsigned int i, j;
02884   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02885   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02886   gtk_combo_box_text_remove (window->combo_variable, i);
02887   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02888   xmlFree (input->label[i]);
02889   --input->nvariables;
02890   for (j = i; j < input->nvariables; ++j)
02891     {
02892       input->label[j] = input->label[j + 1];
02893       input->rangemin[j] = input->rangemin[j + 1];
02894       input->rangemax[j] = input->rangemax[j + 1];
02895       input->rangeminabs[j] = input->rangeminabs[j + 1];
02896       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
02897       input->precision[j] = input->precision[j + 1];
02898       switch (window_get_algorithm ())
02899         {
02900         case ALGORITHM_SWEEP:
02901           input->nsweeps[j] = input->nsweeps[j + 1];
02902           break;
02903         case ALGORITHM_GENETIC:
02904           input->nbits[j] = input->nbits[j + 1];
02905         }
02906     }
02907   j = input->nvariables - 1;
02908   if (i > j)
02909     i = j;
02910   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02911   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
```

```
02912   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02913   window_update ();
02914 }
02915
02920 void
02921 window_add_variable ()
02922 {
02923   unsigned int i, j;
02924 #if DEBUG
02925   fprintf (stderr, "window_add_variable: start\n");
02926 #endif
02927   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02928   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02929   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
02930   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02931   input->label = (char **) g_realloc
02932     (input->label, (input->nvariables + 1) * sizeof (char *));
02933   input->rangemin = (double *) g_realloc
02934     (input->rangemin, (input->nvariables + 1) * sizeof (double));
02935   input->rangemax = (double *) g_realloc
02936     (input->rangemax, (input->nvariables + 1) * sizeof (double));
02937   input->rangeminabs = (double *) g_realloc
02938     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
02939   input->rangemaxabs = (double *) g_realloc
02940     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
02941   input->precision = (unsigned int *) g_realloc
02942     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
02943   for (j = input->nvariables - 1; j > i; --j)
02944     {
02945       input->label[j + 1] = input->label[j];
02946       input->rangemin[j + 1] = input->rangemin[j];
02947       input->rangemax[j + 1] = input->rangemax[j];
02948       input->rangeminabs[j + 1] = input->rangeminabs[j];
02949       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02950       input->precision[j + 1] = input->precision[j];
02951     }
02952   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
02953   input->rangemin[j + 1] = input->rangemin[j];
02954   input->rangemax[j + 1] = input->rangemax[j];
02955   input->rangeminabs[j + 1] = input->rangeminabs[j];
02956   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02957   input->precision[j + 1] = input->precision[j];
02958   switch (window_get_algorithm ())
02959     {
02960     case ALGORITHM_SWEEP:
02961       input->nsweeps = (unsigned int *) g_realloc
02962         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
02963       for (j = input->nvariables - 1; j > i; --j)
02964         input->nsweeps[j + 1] = input->nsweeps[j];
02965       input->nsweeps[j + 1] = input->nsweeps[j];
02966       break;
02967     case ALGORITHM_GENETIC:
02968       input->nbits = (unsigned int *) g_realloc
02969         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
02970       for (j = input->nvariables - 1; j > i; --j)
02971         input->nbits[j + 1] = input->nbits[j];
02972       input->nbits[j + 1] = input->nbits[j];
02973     }
02974   ++input->nvariables;
02975   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02976   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
02977   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02978   window_update ();
02979 #if DEBUG
02980   fprintf (stderr, "window_add_variable: end\n");
02981 #endif
02982 }
02983
02988 void
02989 window_label_variable ()
02990 {
02991   unsigned int i;
02992   const char *buffer;
02993 #if DEBUG
02994   fprintf (stderr, "window_label_variable: start\n");
02995 #endif
02996   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02997   buffer = gtk_entry_get_text (window->entry_variable);
02998   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02999   gtk_combo_box_text_remove (window->combo_variable, i);
```

```
03000   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03001   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03002   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03003 #if DEBUG
03004   fprintf (stderr, "window_label_variable: end\n");
03005 #endif
03006 }
03007
03012 void
03013 window_precision_variable ()
03014 {
03015   unsigned int i;
03016 #if DEBUG
03017   fprintf (stderr, "window_precision_variable: start\n");
03018 #endif
03019   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03020   input->precision[i]
03021     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03022   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03023   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03024   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03025   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03026 #if DEBUG
03027   fprintf (stderr, "window_precision_variable: end\n");
03028 #endif
03029 }
03030
03035 void
03036 window_rangemin_variable ()
03037 {
03038   unsigned int i;
03039 #if DEBUG
03040   fprintf (stderr, "window_rangemin_variable: start\n");
03041 #endif
03042   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03043   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03044 #if DEBUG
03045   fprintf (stderr, "window_rangemin_variable: end\n");
03046 #endif
03047 }
03048
03053 void
03054 window_rangemax_variable ()
03055 {
03056   unsigned int i;
03057 #if DEBUG
03058   fprintf (stderr, "window_rangemax_variable: start\n");
03059 #endif
03060   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03061   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03062 #if DEBUG
03063   fprintf (stderr, "window_rangemax_variable: end\n");
03064 #endif
03065 }
03066
03071 void
03072 window_rangeminabs_variable ()
03073 {
03074   unsigned int i;
03075 #if DEBUG
03076   fprintf (stderr, "window_rangeminabs_variable: start\n");
03077 #endif
03078   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03079   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03080 #if DEBUG
03081   fprintf (stderr, "window_rangeminabs_variable: end\n");
03082 #endif
03083 }
03084
03089 void
03090 window_rangemaxabs_variable ()
03091 {
03092   unsigned int i;
03093 #if DEBUG
03094   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03095 #endif
03096   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03097   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03098 #if DEBUG
03099   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03100 #endif
03101 }
03102
03107 void
```

```
03108 window_update_variable ()
03109 {
03110   unsigned int i;
03111 #if DEBUG
03112   fprintf (stderr, "window_update_variable: start\n");
03113 #endif
03114   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03115   switch (window_get_algorithm ())
03116     {
03117     case ALGORITHM_SWEEP:
03118       input->nsweeps[i]
03119         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03120       break;
03121     case ALGORITHM_GENETIC:
03122       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03123     }
03124 #if DEBUG
03125   fprintf (stderr, "window_update_variable: end\n");
03126 #endif
03127 }
03128
03136 int
03137 window_read (char *filename)
03138 {
03139   unsigned int i;
03140   char *buffer;
03141 #if DEBUG
03142   fprintf (stderr, "window_read: start\n");
03143 #endif
03144   input_free ();
03145   if (!input_open (filename))
03146     {
03147 #if DEBUG
03148       fprintf (stderr, "window_read: end\n");
03149 #endif
03150       return 0;
03151     }
03152   buffer = g_build_filename (input->directory, input->simulator, NULL);
03153   puts (buffer);
03154   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03155                                  (window->button_simulator), buffer);
03156   g_free (buffer);
03157   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03158                                 (size_t) input->evaluator);
03159   if (input->evaluator)
03160     {
03161       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03162       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03163                                      (window->button_evaluator), buffer);
03164       g_free (buffer);
03165     }
03166   gtk_toggle_button_set_active
03167     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03168   switch (input->algorithm)
03169     {
03170     case ALGORITHM_MONTE_CARLO:
03171       gtk_spin_button_set_value (window->spin_simulations,
03172                                  (gdouble) input->nsimulations);
03173     case ALGORITHM_SWEEP:
03174       gtk_spin_button_set_value (window->spin_iterations,
03175                                  (gdouble) input->niterations);
03176       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03177       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03178       break;
03179     default:
03180       gtk_spin_button_set_value (window->spin_population,
03181                                  (gdouble) input->nsimulations);
03182       gtk_spin_button_set_value (window->spin_generations,
03183                                  (gdouble) input->niterations);
03184       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03185       gtk_spin_button_set_value (window->spin_reproduction,
03186                                  input->reproduction_ratio);
03187       gtk_spin_button_set_value (window->spin_adaptation,
03188                                  input->adaptation_ratio);
03189     }
03190   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03191   g_signal_handler_block (window->button_experiment,
03192                           window->id_experiment_name);
03193   gtk_combo_box_text_remove_all (window->combo_experiment);
03194   for (i = 0; i < input->nexperiments; ++i)
03195     gtk_combo_box_text_append_text (window->combo_experiment,
03196                                     input->experiment[i]);
```

```
03197   g_signal_handler_unblock
03198     (window->button_experiment, window->id_experiment_name);
03199   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03200   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03201   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03202   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03203   gtk_combo_box_text_remove_all (window->combo_variable);
03204   for (i = 0; i < input->nvariables; ++i)
03205     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03206   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03207   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03208   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03209   window_set_variable ();
03210   window_update ();
03211 #if DEBUG
03212   fprintf (stderr, "window_read: end\n");
03213 #endif
03214   return 1;
03215 }
03216
03221 void
03222 window_open ()
03223 {
03224   char *buffer;
03225   GtkFileChooserDialog *dlg;
03226   dlg = (GtkFileChooserDialog *)
03227     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03228                                  window->window,
03229                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03230                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03231                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03232   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03233     {
03234       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03235       if (!window_read (buffer))
03236         g_application_quit (G_APPLICATION (window->application));
03237       g_free (buffer);
03238     }
03239   gtk_widget_destroy (GTK_WIDGET (dlg));
03240 }
03241
03246 void
03247 window_new ()
03248 {
03249   unsigned int i;
03250   char *buffer, *buffer2, buffer3[64];
03251   GtkViewport *viewport;
03252   char *label_algorithm[NALGORITHMS] = {
03253     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03254   };
03255
03256   // Creating the window
03257   window->window
03258     = (GtkWindow *) gtk_application_window_new (window->application);
03259
03260   // Setting the window title
03261   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03262
03263   // Creating the open button
03264   window->button_open = (GtkToolButton *) gtk_tool_button_new
03265     (gtk_image_new_from_icon_name ("document-open",
03266                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03267      gettext ("Open"));
03268   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03269
03270   // Creating the save button
03271   window->button_save = (GtkToolButton *) gtk_tool_button_new
03272     (gtk_image_new_from_icon_name ("document-save",
03273                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03274      gettext ("Save"));
03275   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
03276                     NULL);
03277
03278   // Creating the run button
03279   window->button_run = (GtkToolButton *) gtk_tool_button_new
03280     (gtk_image_new_from_icon_name ("system-run",
03281                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03282      gettext ("Run"));
03283   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03284
```

```
03285   // Creating the options button
03286   window->button_options = (GtkToolButton *) gtk_tool_button_new
03287     (gtk_image_new_from_icon_name ("preferences-system",
03288                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03289      gettext ("Options"));
03290   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03291
03292   // Creating the help button
03293   window->button_help = (GtkToolButton *) gtk_tool_button_new
03294     (gtk_image_new_from_icon_name ("help-browser",
03295                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03296      gettext ("Help"));
03297   g_signal_connect (window->button_help, "clicked", window_help, NULL);
03298
03299   // Creating the about button
03300   window->button_about = (GtkToolButton *) gtk_tool_button_new
03301     (gtk_image_new_from_icon_name ("help-about",
03302                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03303      gettext ("About"));
03304   g_signal_connect (window->button_about, "clicked", window_about, NULL);
03305
03306   // Creating the exit button
03307   window->button_exit = (GtkToolButton *) gtk_tool_button_new
03308     (gtk_image_new_from_icon_name ("application-exit",
03309                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03310      gettext ("Exit"));
03311   g_signal_connect_swapped (window->button_exit, "clicked",
03312                             (void (*)) gtk_widget_destroy, window->window);
03313
03314   // Creating the buttons bar
03315   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
03316   gtk_toolbar_insert
03317     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
03318   gtk_toolbar_insert
03319     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
03320   gtk_toolbar_insert
03321     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
03322   gtk_toolbar_insert
03323     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
03324   gtk_toolbar_insert
03325     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
03326   gtk_toolbar_insert
03327     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
03328   gtk_toolbar_insert
03329     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
03330   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
03331
03332   // Creating the simulator program label and entry
03333   window->label_simulator
03334     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
03335   window->button_simulator = (GtkFileChooserButton *)
03336     gtk_file_chooser_button_new (gettext ("Simulator program"),
03337                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03338   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
03339                                gettext ("Simulator program executable file"));
03340
03341   // Creating the evaluator program label and entry
03342   window->check_evaluator = (GtkCheckButton *)
03343     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
03344   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
03345   window->button_evaluator = (GtkFileChooserButton *)
03346     gtk_file_chooser_button_new (gettext ("Evaluator program"),
03347                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03348   gtk_widget_set_tooltip_text
03349     (GTK_WIDGET (window->button_evaluator),
03350      gettext ("Optional evaluator program executable file"));
03351
03352   // Creating the algorithm properties
03353   window->label_simulations = (GtkLabel *) gtk_label_new
03354     (gettext ("Simulations number"));
03355   window->spin_simulations
03356     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03357   window->label_iterations = (GtkLabel *)
03358     gtk_label_new (gettext ("Iterations number"));
03359   window->spin_iterations
03360     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03361   g_signal_connect
03362     (window->spin_iterations, "value-changed", window_update, NULL);
03363   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
03364   window->spin_tolerance
03365     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03366   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
03367   window->spin_bests
03368     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03369   window->label_population
03370     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
```

```
03371   window->spin_population
03372     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03373   window->label_generations
03374     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
03375   window->spin_generations
03376     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03377   window->label_mutation
03378     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
03379   window->spin_mutation
03380     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03381   window->label_reproduction
03382     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
03383   window->spin_reproduction
03384     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03385   window->label_adaptation
03386     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
03387   window->spin_adaptation
03388     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03389
03390   // Creating the array of algorithms
03391   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
03392   window->button_algorithm[0] = (GtkRadioButton *)
03393     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
03394   gtk_grid_attach (window->grid_algorithm,
03395                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
03396   g_signal_connect (window->button_algorithm[0], "clicked",
03397                     window_set_algorithm, NULL);
03398   for (i = 0; ++i < NALGORITHMS;)
03399     {
03400       window->button_algorithm[i] = (GtkRadioButton *)
03401         gtk_radio_button_new_with_mnemonic
03402         (gtk_radio_button_get_group (window->button_algorithm[0]),
03403          label_algorithm[i]);
03404       gtk_grid_attach (window->grid_algorithm,
03405                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
03406       g_signal_connect (window->button_algorithm[i], "clicked",
03407                         window_set_algorithm, NULL);
03408     }
03409   gtk_grid_attach (window->grid_algorithm,
03410                    GTK_WIDGET (window->label_simulations), 0,
03411                    NALGORITHMS, 1, 1);
03412   gtk_grid_attach (window->grid_algorithm,
03413                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
03414   gtk_grid_attach (window->grid_algorithm,
03415                    GTK_WIDGET (window->label_iterations), 0,
03416                    NALGORITHMS + 1, 1, 1);
03417   gtk_grid_attach (window->grid_algorithm,
03418                    GTK_WIDGET (window->spin_iterations), 1,
03419                    NALGORITHMS + 1, 1, 1);
03420   gtk_grid_attach (window->grid_algorithm,
03421                    GTK_WIDGET (window->label_tolerance), 0,
03422                    NALGORITHMS + 2, 1, 1);
03423   gtk_grid_attach (window->grid_algorithm,
03424                    GTK_WIDGET (window->spin_tolerance), 1,
03425                    NALGORITHMS + 2, 1, 1);
03426   gtk_grid_attach (window->grid_algorithm,
03427                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
03428   gtk_grid_attach (window->grid_algorithm,
03429                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
03430   gtk_grid_attach (window->grid_algorithm,
03431                    GTK_WIDGET (window->label_population), 0,
03432                    NALGORITHMS + 4, 1, 1);
03433   gtk_grid_attach (window->grid_algorithm,
03434                    GTK_WIDGET (window->spin_population), 1,
03435                    NALGORITHMS + 4, 1, 1);
03436   gtk_grid_attach (window->grid_algorithm,
03437                    GTK_WIDGET (window->label_generations), 0,
03438                    NALGORITHMS + 5, 1, 1);
03439   gtk_grid_attach (window->grid_algorithm,
03440                    GTK_WIDGET (window->spin_generations), 1,
03441                    NALGORITHMS + 5, 1, 1);
03442   gtk_grid_attach (window->grid_algorithm,
03443                    GTK_WIDGET (window->label_mutation), 0,
03444                    NALGORITHMS + 6, 1, 1);
03445   gtk_grid_attach (window->grid_algorithm,
03446                    GTK_WIDGET (window->spin_mutation), 1,
03447                    NALGORITHMS + 6, 1, 1);
03448   gtk_grid_attach (window->grid_algorithm,
03449                    GTK_WIDGET (window->label_reproduction), 0,
03450                    NALGORITHMS + 7, 1, 1);
03451   gtk_grid_attach (window->grid_algorithm,
03452                    GTK_WIDGET (window->spin_reproduction), 1,
03453                    NALGORITHMS + 7, 1, 1);
03454   gtk_grid_attach (window->grid_algorithm,
03455                    GTK_WIDGET (window->label_adaptation), 0,
03456                    NALGORITHMS + 8, 1, 1);
03457   gtk_grid_attach (window->grid_algorithm,
```

```
03458                   GTK_WIDGET (window->spin_adaptation), 1,
03459                   NALGORITHMS + 8, 1, 1);
03460   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
03461   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
03462                      GTK_WIDGET (window->grid_algorithm));
03463
03464   // Creating the variable widgets
03465   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
03466   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_variable),
03467                                gettext ("Variables selector"));
03468   window->id_variable = g_signal_connect
03469     (window->combo_variable, "changed", window_set_variable, NULL);
03470   window->button_add_variable
03471     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03472                                                    GTK_ICON_SIZE_BUTTON);
03473   g_signal_connect
03474     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
03475   gtk_widget_set_tooltip_text (GTK_WIDGET
03476                                (window->button_add_variable),
03477                                gettext ("Add variable"));
03478   window->button_remove_variable
03479     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03480                                                    GTK_ICON_SIZE_BUTTON);
03481   g_signal_connect
03482     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
03483   gtk_widget_set_tooltip_text (GTK_WIDGET
03484                                (window->button_remove_variable),
03485                                gettext ("Remove variable"));
03486   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
03487   window->entry_variable = (GtkEntry *) gtk_entry_new ();
03488   window->id_variable_label = g_signal_connect
03489     (window->entry_variable, "changed", window_label_variable, NULL);
03490   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
03491   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
03492     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03493   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03494   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
03495   window->scrolled_min
03496     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03497   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
03498                      GTK_WIDGET (viewport));
03499   g_signal_connect (window->spin_min, "value-changed",
03500                     window_rangemin_variable, NULL);
03501   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
03502   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
03503     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03504   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03505   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
03506   window->scrolled_max
03507     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03508   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
03509                      GTK_WIDGET (viewport));
03510   g_signal_connect (window->spin_max, "value-changed",
03511                     window_rangemax_variable, NULL);
03512   window->check_minabs = (GtkCheckButton *)
03513     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
03514   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
03515   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03516     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03517   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03518   gtk_container_add (GTK_CONTAINER (viewport),
03519                      GTK_WIDGET (window->spin_minabs));
03520   window->scrolled_minabs
03521     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03522   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
03523                      GTK_WIDGET (viewport));
03524   g_signal_connect (window->spin_minabs, "value-changed",
03525                     window_rangeminabs_variable, NULL);
03526   window->check_maxabs = (GtkCheckButton *)
03527     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
03528   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
03529   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03530     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03531   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03532   gtk_container_add (GTK_CONTAINER (viewport),
03533                      GTK_WIDGET (window->spin_maxabs));
03534   window->scrolled_maxabs
03535     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03536   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
03537                      GTK_WIDGET (viewport));
03538   g_signal_connect (window->spin_maxabs, "value-changed",
03539                     window_rangemaxabs_variable, NULL);
03540   window->label_precision
03541     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
03542   window->spin_precision = (GtkSpinButton *)
```

```
03543     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
03544   g_signal_connect (window->spin_precision, "value-changed",
03545                     window_precision_variable, NULL);
03546   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
03547   window->spin_sweeps
03548     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03549   g_signal_connect
03550     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
03551   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
03552   window->spin_bits
03553     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03554   g_signal_connect
03555     (window->spin_bits, "value-changed", window_update_variable, NULL);
03556   window->grid_variable = (GtkGrid *) gtk_grid_new ();
03557   gtk_grid_attach (window->grid_variable,
03558                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
03559   gtk_grid_attach (window->grid_variable,
03560                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
03561   gtk_grid_attach (window->grid_variable,
03562                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
03563   gtk_grid_attach (window->grid_variable,
03564                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
03565   gtk_grid_attach (window->grid_variable,
03566                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
03567   gtk_grid_attach (window->grid_variable,
03568                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
03569   gtk_grid_attach (window->grid_variable,
03570                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
03571   gtk_grid_attach (window->grid_variable,
03572                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
03573   gtk_grid_attach (window->grid_variable,
03574                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
03575   gtk_grid_attach (window->grid_variable,
03576                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
03577   gtk_grid_attach (window->grid_variable,
03578                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
03579   gtk_grid_attach (window->grid_variable,
03580                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
03581   gtk_grid_attach (window->grid_variable,
03582                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
03583   gtk_grid_attach (window->grid_variable,
03584                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
03585   gtk_grid_attach (window->grid_variable,
03586                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
03587   gtk_grid_attach (window->grid_variable,
03588                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
03589   gtk_grid_attach (window->grid_variable,
03590                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
03591   gtk_grid_attach (window->grid_variable,
03592                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
03593   gtk_grid_attach (window->grid_variable,
03594                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
03595   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
03596   gtk_container_add (GTK_CONTAINER (window->frame_variable),
03597                      GTK_WIDGET (window->grid_variable));
03598
03599   // Creating the experiment widgets
03600   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03601   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03602                                gettext ("Experiment selector"));
03603   window->id_experiment = g_signal_connect
03604     (window->combo_experiment, "changed", window_set_experiment, NULL)
    ;
03605   window->button_add_experiment
03606     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03607                                                    GTK_ICON_SIZE_BUTTON);
03608   g_signal_connect
03609     (window->button_add_experiment, "clicked",
     window_add_experiment, NULL);
03610   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03611                                gettext ("Add experiment"));
03612   window->button_remove_experiment
03613     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03614                                                    GTK_ICON_SIZE_BUTTON);
03615   g_signal_connect (window->button_remove_experiment, "clicked",
03616                     window_remove_experiment, NULL);
03617   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03618                                gettext ("Remove experiment"));
03619   window->label_experiment
03620     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
03621   window->button_experiment = (GtkFileChooserButton *)
03622     gtk_file_chooser_button_new (gettext ("Experimental data file"),
03623                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03624   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03625                                gettext ("Experimental data file"));
03626   window->id_experiment_name
03627     = g_signal_connect (window->button_experiment, "selection-changed",
```

```
03628                         window_name_experiment, NULL);
03629   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
03630   window->spin_weight
03631     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03632   gtk_widget_set_tooltip_text
03633     (GTK_WIDGET (window->spin_weight),
03634      gettext ("Weight factor to build the objective function"));
03635   g_signal_connect
03636     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
03637   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03638   gtk_grid_attach (window->grid_experiment,
03639                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03640   gtk_grid_attach (window->grid_experiment,
03641                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03642   gtk_grid_attach (window->grid_experiment,
03643                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03644   gtk_grid_attach (window->grid_experiment,
03645                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03646   gtk_grid_attach (window->grid_experiment,
03647                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03648   gtk_grid_attach (window->grid_experiment,
03649                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03650   gtk_grid_attach (window->grid_experiment,
03651                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03652   for (i = 0; i < MAX_NINPUTS; ++i)
03653     {
03654       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03655       window->check_template[i] = (GtkCheckButton *)
03656         gtk_check_button_new_with_label (buffer3);
03657       window->id_template[i]
03658         = g_signal_connect (window->check_template[i], "toggled",
03659                             window_inputs_experiment, NULL);
03660       gtk_grid_attach (window->grid_experiment,
03661                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03662       window->button_template[i] = (GtkFileChooserButton *)
03663         gtk_file_chooser_button_new (gettext ("Input template"),
03664                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03665       gtk_widget_set_tooltip_text
03666         (GTK_WIDGET (window->button_template[i]),
03667          gettext ("Experimental input template file"));
03668       window->id_input[i]
03669         = g_signal_connect_swapped (window->button_template[i],
03670                                     "selection-changed",
03671                                     (void (*)) window_template_experiment,
03672                                     (void *) (size_t) i);
03673       gtk_grid_attach (window->grid_experiment,
03674                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03675     }
03676   window->frame_experiment
03677     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03678   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
03679                      GTK_WIDGET (window->grid_experiment));
03680
03681   // Creating the grid and attaching the widgets to the grid
03682   window->grid = (GtkGrid *) gtk_grid_new ();
03683   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 6, 1);
03684   gtk_grid_attach (window->grid,
03685                    GTK_WIDGET (window->label_simulator), 0, 1, 1, 1);
03686   gtk_grid_attach (window->grid,
03687                    GTK_WIDGET (window->button_simulator), 1, 1, 1, 1);
03688   gtk_grid_attach (window->grid,
03689                    GTK_WIDGET (window->check_evaluator), 2, 1, 1, 1);
03690   gtk_grid_attach (window->grid,
03691                    GTK_WIDGET (window->button_evaluator), 3, 1, 1, 1);
03692   gtk_grid_attach (window->grid,
03693                    GTK_WIDGET (window->frame_algorithm), 0, 2, 2, 1);
03694   gtk_grid_attach (window->grid,
03695                    GTK_WIDGET (window->frame_variable), 2, 2, 2, 1);
03696   gtk_grid_attach (window->grid,
03697                    GTK_WIDGET (window->frame_experiment), 4, 2, 2, 1);
03698   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
03699
03700   // Setting the window logo
03701   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03702   gtk_window_set_icon (window->window, window->logo);
03703
03704   // Showing the window
03705   gtk_widget_show_all (GTK_WIDGET (window->window));
03706
03707   // In Windows the default scrolled size is wrong
03708 #ifdef G_OS_WIN32
03709   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03710   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03711   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03712   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
```

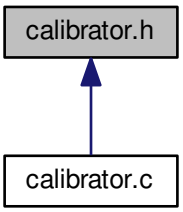
```
03713 #endif
03714
03715   // Reading initial example
03716   buffer2 = g_get_current_dir ();
03717   buffer = g_build_filename (buffer2, "tests", "test1", INPUT_FILE, NULL);
03718   g_free (buffer2);
03719   window_read (buffer);
03720   g_free (buffer);
03721 }
03722
03723 #endif
03724
03730 int
03731 cores_number ()
03732 {
03733 #ifdef G_OS_WIN32
03734   SYSTEM_INFO sysinfo;
03735   GetSystemInfo (&sysinfo);
03736   return sysinfo.dwNumberOfProcessors;
03737 #else
03738   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03739 #endif
03740 }
03741
03751 int
03752 main (int argn, char **argc)
03753 {
03754 #if HAVE_GTK
03755   int status;
03756 #endif
03757 #if HAVE_MPI
03758   // Starting MPI
03759   MPI_Init (&argn, &argc);
03760   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03761   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03762   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03763 #else
03764   ntasks = 1;
03765 #endif
03766   // Starting pseudo-random numbers generator
03767   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03768   gsl_rng_set (calibrate->rng, DEFAULT_RANDOM_SEED);
03769   // Allowing spaces in the XML data file
03770   xmlKeepBlanksDefault (0);
03771
03772 #if HAVE_GTK
03773
03774   nthreads = cores_number ();
03775   setlocale (LC_ALL, "");
03776   setlocale (LC_NUMERIC, "C");
03777   current_directory = g_get_current_dir ();
03778   bindtextdomain
03779     (PROGRAM_INTERFACE, g_build_filename (current_directory,
      LOCALE_DIR, NULL));
03780   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03781   textdomain (PROGRAM_INTERFACE);
03782   gtk_disable_setlocale ();
03783   window->application = gtk_application_new ("git.jburguete.calibrator",
03784                                              G_APPLICATION_FLAGS_NONE);
03785   g_signal_connect (window->application, "activate", window_new, NULL);
03786   status = g_application_run (G_APPLICATION (window->application), argn, argc);
03787   g_object_unref (window->application);
03788
03789 #else
03790
03791   // Checking syntax
03792   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03793     {
03794       printf ("The syntax is:\ncalibrator [-nthreads x] data_file\n");
03795 #if HAVE_MPI
03796       // Closing MPI
03797       MPI_Finalize ();
03798 #endif
03799       return 1;
03800     }
03801   // Getting threads number
03802   if (argn == 2)
03803     nthreads = cores_number ();
03804   else
03805     nthreads = atoi (argc[2]);
03806   printf ("nthreads=%u\n", nthreads);
03807   // Making calibration
03808   if (input_open (argc[argn - 1]))
03809     calibrate_new ();
03810   // Freeing memory
03811   calibrate_free ();
03812
```

```
03813 #endif
03814
03815   // Freeing memory
03816   gsl_rng_free (calibrate->rng);
03817 #if HAVE_MPI
03818   // Closing MPI
03819   MPI_Finalize ();
03820 #endif
03821
03822 #if HAVE_GTK
03823   g_free (current_directory);
03824   return status;
03825 #else
03826   return 0;
03827 #endif
03828 }
```

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.
- int input_open (char *filename)

  Function to open the input file.
- void input_free ()

  Function to free the memory of the input file data.
- 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 49 of file calibrator.h.

```
00050 {
00051   ALGORITHM_MONTE_CARLO = 0,
00052   ALGORITHM_SWEEP = 1,
00053   ALGORITHM_GENETIC = 2
00054 };
```

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 1267 of file calibrator.c.

```
01268 {
01269   unsigned int i, j;
01270   double e;
01271 #if DEBUG
01272   fprintf (stderr, "calibrate_best_sequential: start\n");
01273 #endif
01274   if (calibrate->nsaveds < calibrate->nbest
01275       || value < calibrate->error_best[calibrate->nsaveds - 1])
01276     {
01277       if (calibrate->nsaveds < calibrate->nbest)
01278         ++calibrate->nsaveds;
01279       calibrate->error_best[calibrate->nsaveds - 1] = value;
01280       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01281       for (i = calibrate->nsaveds; --i;)
01282         {
01283           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01284             {
01285               j = calibrate->simulation_best[i];
01286               e = calibrate->error_best[i];
01287               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01288               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01289               calibrate->simulation_best[i - 1] = j;
01290               calibrate->error_best[i - 1] = e;
01291             }
01292           else
01293             break;
01294         }
01295     }
01296 #if DEBUG
01297   fprintf (stderr, "calibrate_best_sequential: end\n");
01298 #endif
01299 }
```

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 1222 of file calibrator.c.

```
01223 {
01224   unsigned int i, j;
01225   double e;
01226 #if DEBUG
01227   fprintf (stderr, "calibrate_best_thread: start\n");
01228 #endif
01229   if (calibrate->nsaveds < calibrate->nbest
01230       || value < calibrate->error_best[calibrate->nsaveds - 1])
01231     {
01232       g_mutex_lock (mutex);
01233       if (calibrate->nsaveds < calibrate->nbest)
01234         ++calibrate->nsaveds;
01235       calibrate->error_best[calibrate->nsaveds - 1] = value;
01236       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01237       for (i = calibrate->nsaveds; --i;)
01238         {
01239           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01240             {
01241               j = calibrate->simulation_best[i];
01242               e = calibrate->error_best[i];
01243               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01244               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01245               calibrate->simulation_best[i - 1] = j;
01246               calibrate->error_best[i - 1] = e;
01247             }
01248           else
01249             break;
01250         }
01251       g_mutex_unlock (mutex);
```

```
01252     }
01253 #if DEBUG
01254   fprintf (stderr, "calibrate_best_thread: end\n");
01255 #endif
01256 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

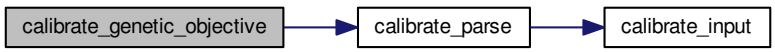
Returns

objective function value.

Definition at line 1576 of file calibrator.c.

```
01577 {
01578   unsigned int j;
01579   double objective;
01580   char buffer[64];
01581 #if DEBUG
01582   fprintf (stderr, "calibrate_genetic_objective: start\n");
01583 #endif
01584   for (j = 0; j < calibrate->nvariables; ++j)
01585     {
01586       calibrate->value[entity->id * calibrate->nvariables + j]
01587         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01588     }
01589   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01590     objective += calibrate_parse (entity->id, j);
01591   g_mutex_lock (mutex);
01592   for (j = 0; j < calibrate->nvariables; ++j)
01593     {
01594       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01595       fprintf (calibrate->file_variables, buffer,
01596                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01597     }
01598   fprintf (calibrate->file_variables, "%.14le\n", objective);
01599   g_mutex_unlock (mutex);
01600 #if DEBUG
01601   fprintf (stderr, "calibrate_genetic_objective: end\n");
01602 #endif
01603   return objective;
01604 }
```

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 971 of file calibrator.c.

```
00972 {
00973   unsigned int i;
00974   char buffer[32], value[32], *buffer2, *buffer3, *content;
00975   FILE *file;
00976   gsize length;
00977   GRegex *regex;
00978
00979 #if DEBUG
00980   fprintf (stderr, "calibrate_input: start\n");
00981 #endif
00982
00983   // Checking the file
00984   if (!template)
00985     goto calibrate_input_end;
00986
00987   // Opening template
00988   content = g_mapped_file_get_contents (template);
00989   length = g_mapped_file_get_length (template);
00990 #if DEBUG
00991   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
00992            content);
00993 #endif
00994   file = fopen (input, "w");
00995
00996   // Parsing template
00997   for (i = 0; i < calibrate->nvariables; ++i)
00998     {
00999 #if DEBUG
01000       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01001 #endif
01002       snprintf (buffer, 32, "@variable%u@", i + 1);
01003       regex = g_regex_new (buffer, 0, 0, NULL);
01004       if (i == 0)
01005         {
01006           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01007                                              calibrate->label[i], 0, NULL);
01008 #if DEBUG
01009           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01010 #endif
01011         }
01012       else
01013         {
01014           length = strlen (buffer3);
01015           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01016                                              calibrate->label[i], 0, NULL);
01017           g_free (buffer3);
01018         }
01019       g_regex_unref (regex);
01020       length = strlen (buffer2);
01021       snprintf (buffer, 32, "@value%u@", i + 1);
01022       regex = g_regex_new (buffer, 0, 0, NULL);
01023       snprintf (value, 32, format[calibrate->precision[i]],
01024                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01025
01026 #if DEBUG
01027       fprintf (stderr, "calibrate_input: value=%s\n", value);
01028 #endif
01029       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01030                                          0, NULL);
01031       g_free (buffer2);
01032       g_regex_unref (regex);
01033     }
01034
01035   // Saving input file
01036   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01037   g_free (buffer3);
01038   fclose (file);
01039
01040 calibrate_input_end:
01041 #if DEBUG
01042   fprintf (stderr, "calibrate_input: end\n");
01043 #endif
01044   return;
01045 }
```

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 1383 of file calibrator.c.

```
01385 {
01386   unsigned int i, j, k, s[calibrate->nbest];
01387   double e[calibrate->nbest];
01388 #if DEBUG
01389   fprintf (stderr, "calibrate_merge: start\n");
01390 #endif
01391   i = j = k = 0;
01392   do
01393     {
01394       if (i == calibrate->nsaveds)
01395         {
01396           s[k] = simulation_best[j];
01397           e[k] = error_best[j];
01398           ++j;
01399           ++k;
01400           if (j == nsaveds)
01401             break;
01402         }
01403       else if (j == nsaveds)
01404         {
01405           s[k] = calibrate->simulation_best[i];
01406           e[k] = calibrate->error_best[i];
01407           ++i;
01408           ++k;
01409           if (i == calibrate->nsaveds)
01410             break;
01411         }
01412       else if (calibrate->error_best[i] > error_best[j])
01413         {
01414           s[k] = simulation_best[j];
01415           e[k] = error_best[j];
01416           ++j;
01417           ++k;
01418         }
01419       else
01420         {
01421           s[k] = calibrate->simulation_best[i];
01422           e[k] = calibrate->error_best[i];
01423           ++i;
01424           ++k;
01425         }
01426     }
01427   while (k < calibrate->nbest);
01428   calibrate->nsaveds = k;
01429   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01430   memcpy (calibrate->error_best, e, k * sizeof (double));
01431 #if DEBUG
01432   fprintf (stderr, "calibrate_merge: end\n");
01433 #endif
01434 }
```

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 1058 of file calibrator.c.

```
01059 {
01060   unsigned int i;
01061   double e;
01062   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01063     *buffer3, *buffer4;
01064   FILE *file_result;
01065
01066 #if DEBUG
01067   fprintf (stderr, "calibrate_parse: start\n");
01068   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01069            experiment);
01070 #endif
01071
01072   // Opening input files
01073   for (i = 0; i < calibrate->ninputs; ++i)
01074     {
01075       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01076 #if DEBUG
01077       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01078 #endif
01079       calibrate_input (simulation, &input[i][0],
01080                        calibrate->file[i][experiment]);
01081     }
01082   for (; i < MAX_NINPUTS; ++i)
01083     strcpy (&input[i][0], "");
01084 #if DEBUG
01085   fprintf (stderr, "calibrate_parse: parsing end\n");
01086 #endif
01087
01088   // Performing the simulation
01089   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01090   buffer2 = g_path_get_dirname (calibrate->simulator);
01091   buffer3 = g_path_get_basename (calibrate->simulator);
01092   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01093   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01094             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01095             input[6], input[7], output);
01096   g_free (buffer4);
01097   g_free (buffer3);
01098   g_free (buffer2);
01099 #if DEBUG
01100   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01101 #endif
01102   system (buffer);
01103
01104   // Checking the objective value function
01105   if (calibrate->evaluator)
01106     {
01107       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01108       buffer2 = g_path_get_dirname (calibrate->evaluator);
01109       buffer3 = g_path_get_basename (calibrate->evaluator);
01110       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01111       snprintf (buffer, 512, "\"%s\" %s %s %s",
01112                 buffer4, output, calibrate->experiment[experiment], result);
01113       g_free (buffer4);
01114       g_free (buffer3);
01115       g_free (buffer2);
01116 #if DEBUG
01117       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01118 #endif
01119       system (buffer);
01120       file_result = fopen (result, "r");
01121       e = atof (fgets (buffer, 512, file_result));
01122       fclose (file_result);
01123     }
01124   else
01125     {
01126       strcpy (result, "");
01127       file_result = fopen (output, "r");
01128       e = atof (fgets (buffer, 512, file_result));
01129       fclose (file_result);
01130     }
01131
01132   // Removing files
01133 #if !DEBUG
01134   for (i = 0; i < calibrate->ninputs; ++i)
01135     {
01136       if (calibrate->file[i][0])
01137         {
```

```
01138           snprintf (buffer, 512, RM " %s", &input[i][0]);
01139           system (buffer);
01140         }
01141     }
01142   snprintf (buffer, 512, RM " %s %s", output, result);
01143   system (buffer);
01144 #endif
01145
01146 #if DEBUG
01147   fprintf (stderr, "calibrate_parse: end\n");
01148 #endif
01149
01150   // Returning the objective function
01151   return e * calibrate->weight[experiment];
01152 }
```

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 1194 of file calibrator.c.

```
01195 {
01196   unsigned int i;
01197   char buffer[64];
01198 #if DEBUG
01199   fprintf (stderr, "calibrate_save_variables: start\n");
01200 #endif
01201   for (i = 0; i < calibrate->nvariables; ++i)
01202     {
01203       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01204       fprintf (calibrate->file_variables, buffer,
01205                calibrate->value[simulation * calibrate->
      nvariables + i]);
01206     }
01207   fprintf (calibrate->file_variables, "%.14le\n", error);
01208 #if DEBUG
01209   fprintf (stderr, "calibrate_save_variables: end\n");
01210 #endif
01211 }
```

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

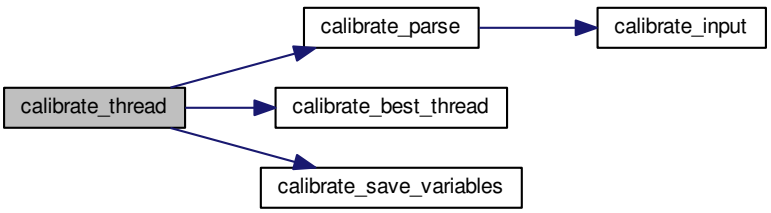
Returns

NULL

Definition at line 1309 of file calibrator.c.

```
01310 {
01311   unsigned int i, j, thread;
01312   double e;
01313 #if DEBUG
01314   fprintf (stderr, "calibrate_thread: start\n");
01315 #endif
01316   thread = data->thread;
01317 #if DEBUG
01318   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01319            calibrate->thread[thread], calibrate->thread[thread + 1]);
01320 #endif
01321   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01322     {
01323       e = 0.;
01324       for (j = 0; j < calibrate->nexperiments; ++j)
01325         e += calibrate_parse (i, j);
01326       calibrate_best_thread (i, e);
01327       g_mutex_lock (mutex);
01328       calibrate_save_variables (i, e);
01329       g_mutex_unlock (mutex);
01330 #if DEBUG
01331       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01332 #endif
01333     }
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_thread: end\n");
01336 #endif
01337   g_thread_exit (NULL);
01338   return NULL;
01339 }
```

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 458 of file calibrator.c.

```
00459 {
00460   int error_code;
00461   unsigned int i;
00462   char buffer2[64];
00463   xmlChar *buffer;
00464   xmlDoc *doc;
00465   xmlNode *node, *child;
00466
00467 #if DEBUG
00468   fprintf (stderr, "input_new: start\n");
00469 #endif
00470
00471   // Resetting input data
00472   input_new ();
00473
00474   // Parsing the input file
00475   doc = xmlParseFile (filename);
00476   if (!doc)
00477     {
00478       show_error (gettext ("Unable to parse the input file"));
00479       return 0;
00480     }
00481
00482   // Getting the root node
00483   node = xmlDocGetRootElement (doc);
00484   if (xmlStrcmp (node->name, XML_CALIBRATE))
00485     {
00486       show_error (gettext ("Bad root XML node"));
00487       return 0;
00488     }
00489
00490   // Opening simulator program name
00491   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00492   if (!input->simulator)
00493     {
00494       show_error (gettext ("Bad simulator program"));
00495       return 0;
00496     }
00497
00498   // Opening evaluator program name
00499   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00500
00501   // Opening algorithm
00502   buffer = xmlGetProp (node, XML_ALGORITHM);
00503   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00504     {
00505       input->algorithm = ALGORITHM_MONTE_CARLO;
00506
00507       // Obtaining simulations number
00508       input->nsimulations
00509         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00510       if (error_code)
00511         {
00512           show_error (gettext ("Bad simulations number"));
00513           return 0;
00514         }
00515     }
00516   else if (!xmlStrcmp (buffer, XML_SWEEP))
00517     input->algorithm = ALGORITHM_SWEEP;
00518   else if (!xmlStrcmp (buffer, XML_GENETIC))
00519     {
00520       input->algorithm = ALGORITHM_GENETIC;
00521
00522       // Obtaining population
00523       if (xmlHasProp (node, XML_NPOPULATION))
00524         {
00525           input->nsimulations
00526             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00527           if (error_code || input->nsimulations < 3)
00528             {
00529               show_error (gettext ("Invalid population number"));
00530               return 0;
00531             }
00532         }
00533       else
```

```
00534         {
00535           show_error (gettext ("No population number"));
00536           return 0;
00537         }
00538
00539       // Obtaining generations
00540       if (xmlHasProp (node, XML_NGENERATIONS))
00541         {
00542           input->niterations
00543             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00544           if (error_code || !input->niterations)
00545             {
00546               show_error (gettext ("Invalid generation number"));
00547               return 0;
00548             }
00549         }
00550       else
00551         {
00552           show_error (gettext ("No generation number"));
00553           return 0;
00554         }
00555
00556       // Obtaining mutation probability
00557       if (xmlHasProp (node, XML_MUTATION))
00558         {
00559           input->mutation_ratio
00560             = xml_node_get_float (node, XML_MUTATION, &error_code);
00561           if (error_code || input->mutation_ratio < 0.
00562               || input->mutation_ratio >= 1.)
00563             {
00564               show_error (gettext ("Invalid mutation probability"));
00565               return 0;
00566             }
00567         }
00568       else
00569         {
00570           show_error (gettext ("No mutation probability"));
00571           return 0;
00572         }
00573
00574       // Obtaining reproduction probability
00575       if (xmlHasProp (node, XML_REPRODUCTION))
00576         {
00577           input->reproduction_ratio
00578             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00579           if (error_code || input->reproduction_ratio < 0.
00580               || input->reproduction_ratio >= 1.0)
00581             {
00582               show_error (gettext ("Invalid reproduction probability"));
00583               return 0;
00584             }
00585         }
00586       else
00587         {
00588           show_error (gettext ("No reproduction probability"));
00589           return 0;
00590         }
00591
00592       // Obtaining adaptation probability
00593       if (xmlHasProp (node, XML_ADAPTATION))
00594         {
00595           input->adaptation_ratio
00596             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00597           if (error_code || input->adaptation_ratio < 0.
00598               || input->adaptation_ratio >= 1.)
00599             {
00600               show_error (gettext ("Invalid adaptation probability"));
00601               return 0;
00602             }
00603         }
00604       else
00605         {
00606           show_error (gettext ("No adaptation probability"));
00607           return 0;
00608         }
00609
00610       // Checking survivals
00611       i = input->mutation_ratio * input->nsimulations;
00612       i += input->reproduction_ratio * input->
     nsimulations;
00613       i += input->adaptation_ratio * input->
     nsimulations;
00614       if (i > input->nsimulations - 2)
00615         {
00616           show_error
00617             (gettext
00618              ("No enough survival entities to reproduce the population"));
```

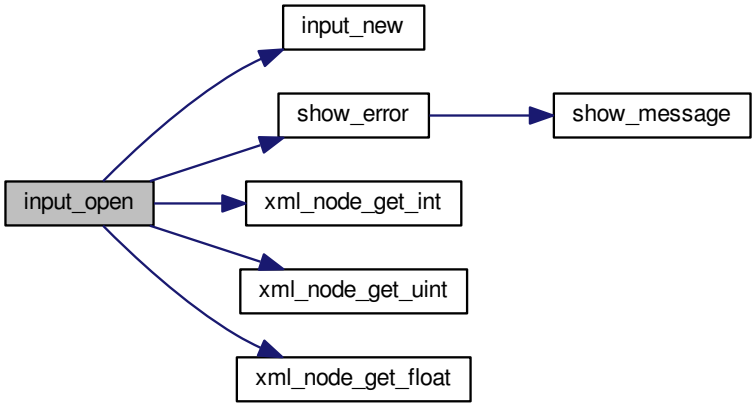
```
00619           return 0;
00620         }
00621     }
00622   else
00623     {
00624       show_error (gettext ("Unknown algorithm"));
00625       return 0;
00626     }
00627
00628   if (input->algorithm == ALGORITHM_MONTE_CARLO
00629       || input->algorithm == ALGORITHM_SWEEP)
00630     {
00631
00632       // Obtaining iterations number
00633       input->niterations
00634         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00635       if (error_code == 1)
00636         input->niterations = 1;
00637       else if (error_code)
00638         {
00639           show_error (gettext ("Bad iterations number"));
00640           return 0;
00641         }
00642
00643       // Obtaining best number
00644       if (xmlHasProp (node, XML_NBEST))
00645         {
00646           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00647           if (error_code || !input->nbest)
00648             {
00649               show_error (gettext ("Invalid best number"));
00650               return 0;
00651             }
00652         }
00653       else
00654         input->nbest = 1;
00655
00656       // Obtaining tolerance
00657       if (xmlHasProp (node, XML_TOLERANCE))
00658         {
00659           input->tolerance
00660             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00661           if (error_code || input->tolerance < 0.)
00662             {
00663               show_error (gettext ("Invalid tolerance"));
00664               return 0;
00665             }
00666         }
00667       else
00668         input->tolerance = 0.;
00669     }
00670
00671   // Reading the experimental data
00672   for (child = node->children; child; child = child->next)
00673     {
00674       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00675         break;
00676 #if DEBUG
00677       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00678 #endif
00679       if (xmlHasProp (child, XML_NAME))
00680         {
00681           input->experiment
00682             = g_realloc (input->experiment,
00683                          (1 + input->nexperiments) * sizeof (char *));
00684           input->experiment[input->nexperiments]
00685             = (char *) xmlGetProp (child, XML_NAME);
00686         }
00687       else
00688         {
00689           show_error (gettext ("No experiment file name"));
00690           return 0;
00691         }
00692 #if DEBUG
00693       fprintf (stderr, "input_new: experiment=%s\n",
00694                input->experiment[input->nexperiments]);
00695 #endif
00696       input->weight = g_realloc (input->weight,
00697                                  (1 + input->nexperiments) * sizeof (double));
00698       if (xmlHasProp (child, XML_WEIGHT))
00699         input->weight[input->nexperiments]
00700           = xml_node_get_float (child, XML_WEIGHT, &error_code);
00701       else
00702         input->weight[input->nexperiments] = 1.;
00703 #if DEBUG
00704       fprintf (stderr, "input_new: weight=%lg\n",
```

```
00705                 input->weight[input->nexperiments]);
00706 #endif
00707       if (!input->nexperiments)
00708         input->ninputs = 0;
00709 #if DEBUG
00710       fprintf (stderr, "input_new: template[0]\n");
00711 #endif
00712       if (xmlHasProp (child, XML_TEMPLATE1))
00713         {
00714           input->template[0]
00715             = (char **) g_realloc (input->template[0],
00716                                    (1 + input->nexperiments) * sizeof (char *));
00717           input->template[0][input->nexperiments]
00718             = (char *) xmlGetProp (child, template[0]);
00719 #if DEBUG
00720           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00721                    input->nexperiments,
00722                    input->template[0][input->nexperiments]);
00723 #endif
00724           if (!input->nexperiments)
00725             ++input->ninputs;
00726 #if DEBUG
00727           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00728 #endif
00729         }
00730       else
00731         {
00732           show_error (gettext ("No experiment template"));
00733           return 0;
00734         }
00735       for (i = 1; i < MAX_NINPUTS; ++i)
00736         {
00737 #if DEBUG
00738           fprintf (stderr, "input_new: template%u\n", i + 1);
00739 #endif
00740           if (xmlHasProp (child, template[i]))
00741             {
00742               if (input->nexperiments && input->ninputs < 2)
00743                 {
00744                   snprintf (buffer2, 64,
00745                             gettext ("Experiment %u: bad templates number"),
00746                             input->nexperiments + 1);
00747                   show_error (buffer2);
00748                   return 0;
00749                 }
00750               input->template[i] = (char **)
00751                 g_realloc (input->template[i],
00752                            (1 + input->nexperiments) * sizeof (char *));
00753               input->template[i][input->nexperiments]
00754                 = (char *) xmlGetProp (child, template[i]);
00755 #if DEBUG
00756               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00757                        input->nexperiments, i + 1,
00758                        input->template[i][input->nexperiments]);
00759 #endif
00760               if (!input->nexperiments)
00761                 ++input->ninputs;
00762 #if DEBUG
00763               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00764 #endif
00765             }
00766           else if (input->nexperiments && input->ninputs > 1)
00767             {
00768               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00769                         input->nexperiments + 1, i + 1);
00770               show_error (buffer2);
00771               return 0;
00772             }
00773           else
00774             break;
00775         }
00776       ++input->nexperiments;
00777 #if DEBUG
00778       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00779 #endif
00780     }
00781   if (!input->nexperiments)
00782     {
00783       show_error (gettext ("No calibration experiments"));
00784       return 0;
00785     }
00786
00787   // Reading the variables data
00788   for (; child; child = child->next)
00789     {
00790       if (xmlStrcmp (child->name, XML_VARIABLE))
00791         {
```

```
00792           show_error (gettext ("Bad XML node"));
00793           return 0;
00794         }
00795       if (xmlHasProp (child, XML_NAME))
00796         {
00797           input->label = g_realloc
00798             (input->label, (1 + input->nvariables) * sizeof (char *));
00799           input->label[input->nvariables]
00800             = (char *) xmlGetProp (child, XML_NAME);
00801         }
00802       else
00803         {
00804           show_error (gettext ("No variable name"));
00805           return 0;
00806         }
00807       if (xmlHasProp (child, XML_MINIMUM))
00808         {
00809           input->rangemin = g_realloc
00810             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00811           input->rangeminabs = g_realloc
00812             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00813           input->rangemin[input->nvariables]
00814             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00815           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00816             {
00817               input->rangeminabs[input->nvariables]
00818                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00819             }
00820           else
00821             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00822         }
00823       else
00824         {
00825           show_error (gettext ("No minimum range"));
00826           return 0;
00827         }
00828       if (xmlHasProp (child, XML_MAXIMUM))
00829         {
00830           input->rangemax = g_realloc
00831             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00832           input->rangemaxabs = g_realloc
00833             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00834           input->rangemax[input->nvariables]
00835             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00836           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00837             input->rangemaxabs[input->nvariables]
00838               = xml_node_get_float (child,
    XML_ABSOLUTE_MAXIMUM, &error_code);
00839           else
00840             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00841         }
00842       else
00843         {
00844           show_error (gettext ("No maximum range"));
00845           return 0;
00846         }
00847       input->precision = g_realloc
00848         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00849       if (xmlHasProp (child, XML_PRECISION))
00850         input->precision[input->nvariables]
00851           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00852       else
00853         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00854       if (input->algorithm == ALGORITHM_SWEEP)
00855         {
00856           if (xmlHasProp (child, XML_NSWEEPS))
00857             {
00858               input->nsweeps = (unsigned int *)
00859                 g_realloc (input->nsweeps,
00860                            (1 + input->nvariables) * sizeof (unsigned int));
00861               input->nsweeps[input->nvariables]
00862                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00863             }
00864           else
00865             {
00866               show_error (gettext ("No sweeps number"));
00867               return 0;
00868             }
00869 #if DEBUG
00870           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00871                    input->nsweeps[input->nvariables],
    input->nsimulations);
00872 #endif
00873         }
00874       if (input->algorithm == ALGORITHM_GENETIC)
```

```
00875         {
00876           // Obtaining bits representing each variable
00877           if (xmlHasProp (child, XML_NBITS))
00878             {
00879               input->nbits = (unsigned int *)
00880                 g_realloc (input->nbits,
00881                            (1 + input->nvariables) * sizeof (unsigned int));
00882               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00883               if (error_code || !i)
00884                 {
00885                   show_error (gettext ("Invalid bit number"));
00886                   return 0;
00887                 }
00888               input->nbits[input->nvariables] = i;
00889             }
00890           else
00891             {
00892               show_error (gettext ("No bits number"));
00893               return 0;
00894             }
00895         }
00896       ++input->nvariables;
00897     }
00898   if (!input->nvariables)
00899     {
00900       show_error (gettext ("No calibration variables"));
00901       return 0;
00902     }
00903
00904   // Getting the working directory
00905   input->directory = g_path_get_dirname (filename);
00906   input->name = g_path_get_basename (filename);
00907
00908   // Closing the XML document
00909   xmlFreeDoc (doc);
00910
00911 #if DEBUG
00912   fprintf (stderr, "input_new: end\n");
00913 #endif
00914
00915   return 1;
00916 }
```

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.

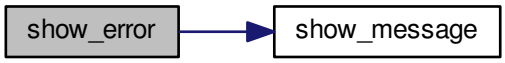
Parameters

msg	Error message.

Definition at line 273 of file calibrator.c.

```
00274 {
00275   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00276 }
```

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 243 of file calibrator.c.

```
00244 {
00245 #if HAVE_GTK
00246   GtkMessageDialog *dlg;
00247
00248   // Creating the dialog
00249   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00250     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00251
00252   // Setting the dialog title
00253   gtk_window_set_title (GTK_WINDOW (dlg), title);
00254
00255   // Showing the dialog and waiting response
00256   gtk_dialog_run (GTK_DIALOG (dlg));
00257
00258   // Closing and freeing memory
00259   gtk_widget_destroy (GTK_WIDGET (dlg));
00260
00261 #else
00262   printf ("%s: %s\n", title, msg);
00263 #endif
00264 }
```

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 352 of file calibrator.c.

```
00353 {
00354   double x = 0.;
00355   xmlChar *buffer;
00356   buffer = xmlGetProp (node, prop);
00357   if (!buffer)
00358     *error_code = 1;
00359   else
00360     {
00361       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00362         *error_code = 2;
00363       else
00364         *error_code = 0;
00365       xmlFree (buffer);
00366     }
00367   return x;
00368 }
```

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 290 of file calibrator.c.

```
00291 {
00292   int i = 0;
00293   xmlChar *buffer;
00294   buffer = xmlGetProp (node, prop);
00295   if (!buffer)
00296     *error_code = 1;
00297   else
00298     {
00299       if (sscanf ((char *) buffer, "%d", &i) != 1)
00300         *error_code = 2;
00301       else
00302         *error_code = 0;
00303       xmlFree (buffer);
00304     }
00305   return i;
00306 }
```

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 321 of file calibrator.c.

```
00322 {
00323   unsigned int i = 0;
00324   xmlChar *buffer;
00325   buffer = xmlGetProp (node, prop);
00326   if (!buffer)
00327     *error_code = 1;
00328   else
00329     {
00330       if (sscanf ((char *) buffer, "%u", &i) != 1)
00331         *error_code = 2;
00332       else
00333         *error_code = 0;
00334       xmlFree (buffer);
00335     }
00336   return i;
00337 }
```

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 419 of file calibrator.c.

```
00420 {
00421   xmlChar buffer[64];
00422   snprintf ((char *) buffer, 64, "%.14lg", value);
00423   xmlSetProp (node, prop, buffer);
00424 }
```

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 381 of file calibrator.c.

```
00382 {
00383   xmlChar buffer[64];
00384   snprintf ((char *) buffer, 64, "%d", value);
00385   xmlSetProp (node, prop, buffer);
00386 }
```

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 400 of file calibrator.c.

```
00401 {
00402   xmlChar buffer[64];
00403   snprintf ((char *) buffer, 64, "%u", value);
00404   xmlSetProp (node, prop, buffer);
00405 }
```

5.4 calibrator.h

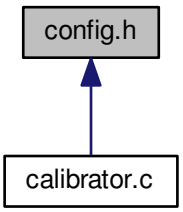
```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided 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
00049 enum Algorithm
00050 {
00051   ALGORITHM_MONTE_CARLO = 0,
00052   ALGORITHM_SWEEP = 1,
00053   ALGORITHM_GENETIC = 2
00054 };
00055
00060 typedef struct
00061 {
00116   char *simulator, *evaluator, **experiment, **template[MAX_NINPUTS], **label,
00117     *directory, *name;
00118   double *rangemin, *rangemax, *rangeminabs, *rangemaxabs, *weight, tolerance,
00119     mutation_ratio, reproduction_ratio, adaptation_ratio;
00120   unsigned int nvariables, nexperiments, ninputs, nsimulations, algorithm,
00121     *precision, *nsweeps, *nbits, niterations, nbest;
00122 } Input;
00123
00128 typedef struct
00129 {
00208   char *simulator, *evaluator, **experiment, **template[MAX_NINPUTS], **label;
00209   unsigned int nvariables, nexperiments, ninputs, nsimulations, algorithm,
00210     *precision, *nsweeps, nstart, nend, *thread, niterations, nbest, nsaveds,
00211     *simulation_best;
00212   double *value, *rangemin, *rangemax, *rangeminabs, *rangemaxabs, *error_best,
00213     *weight, *value_old, *error_old, tolerance, mutation_ratio,
00214     reproduction_ratio, adaptation_ratio;
00215   FILE *file_result, *file_variables;
00216   gsl_rng *rng;
00217   GMappedFile **file[MAX_NINPUTS];
00218   GeneticVariable *genetic_variable;
00219 #if HAVE_MPI
00220   int mpi_rank;
```

```
00221 #endif
00222 } Calibrate;
00223
00228 typedef struct
00229 {
00234   unsigned int thread;
00235 } ParallelData;
00236
00237 // Public functions
00238 void show_message (char *title, char *msg, int type);
00239 void show_error (char *msg);
00240 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00241 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00242                                 int *error_code);
00243 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00244                            int *error_code);
00245 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00246 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00247                         unsigned int value);
00248 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00249 void input_new ();
00250 int input_open (char *filename);
00251 void input_free ();
00252 void calibrate_input (unsigned int simulation, char *input,
00253                       GMappedFile * template);
00254 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00255 void calibrate_print ();
00256 void calibrate_save_variables (unsigned int simulation, double error);
00257 void calibrate_best_thread (unsigned int simulation, double value);
00258 void calibrate_best_sequential (unsigned int simulation, double value);
00259 void *calibrate_thread (ParallelData * data);
00260 void calibrate_sequential ();
00261 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00262                       double *error_best);
00263 #if HAVE_MPI
00264 void calibrate_synchronise ();
00265 #endif
00266 void calibrate_sweep ();
00267 void calibrate_MonteCarlo ();
00268 double calibrate_genetic_objective (Entity * entity);
00269 void calibrate_genetic ();
00270 void calibrate_save_old ();
00271 void calibrate_merge_old ();
00272 void calibrate_refine ();
00273 void calibrate_iterate ();
00274 void calibrate_new ();
00275
00276 #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_ALGORITHM "Monte-Carlo"

  Macro to set the default algorithm.
- #define DEFAULT_PRECISION (NPRECISIONS - 1)

  Macro to set the default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

  Macro to set the 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_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

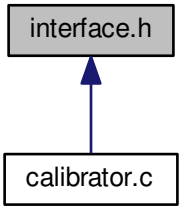
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary 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
00048 #define MAX_NINPUTS 8
00049 #define NALGORITHMS 3
00050 #define NPRECISIONS 15
00051
00052 // Default choices
00053
00062 #define DEFAULT_ALGORITHM "Monte-Carlo"
00063 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00064 #define DEFAULT_RANDOM_SEED 7007
00065
00066 // Interface labels
00067
00074 #define LOCALE_DIR "locales"
00075 #define PROGRAM_INTERFACE "calibrator"
00076
00077 // XML labels
00078
00151 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00152 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00153 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00154 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00155 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00156 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00157 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00158 #define XML_GENETIC (const xmlChar*)"genetic"
00159 #define XML_MINIMUM (const xmlChar*)"minimum"
00160 #define XML_MAXIMUM (const xmlChar*)"maximum"
00161 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00162 #define XML_MUTATION (const xmlChar*)"mutation"
00163 #define XML_NAME (const xmlChar*)"name"
00164 #define XML_NBEST (const xmlChar*)"nbest"
00165 #define XML_NBITS (const xmlChar*)"nbits"
00166 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00167 #define XML_NITERATIONS (const xmlChar*)"niterations"
00168 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00169 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00170 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00171 #define XML_PRECISION (const xmlChar*)"precision"
00172 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00173 #define XML_SIMULATOR (const xmlChar*)"simulator"
00174 #define XML_SWEEP (const xmlChar*)"sweep"
00175 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00176 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00177 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00178 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00179 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00180 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00181 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00182 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00183 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00184 #define XML_VARIABLE (const xmlChar*)"variable"
```

```
00185 #define XML_WEIGHT (const xmlChar*)"weight"
00186
00187 #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 3731 of file calibrator.c.

```
03732 {
03733 #ifdef G_OS_WIN32
03734   SYSTEM_INFO sysinfo;
03735   GetSystemInfo (&sysinfo);
03736   return sysinfo.dwNumberOfProcessors;
03737 #else
03738   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03739 #endif
03740 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

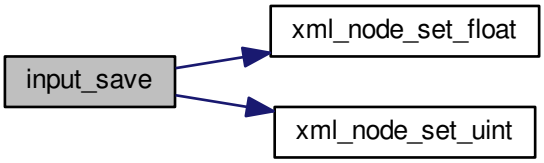
Definition at line 2070 of file calibrator.c.

```
02071 {
02072   unsigned int i, j;
02073   char *buffer;
02074   xmlDoc *doc;
02075   xmlNode *node, *child;
02076   GFile *file, *file2;
02077
02078   // Getting the input file directory
02079   input->name = g_path_get_basename (filename);
02080   input->directory = g_path_get_dirname (filename);
02081   file = g_file_new_for_path (input->directory);
02082
02083   // Opening the input file
02084   doc = xmlNewDoc ((const xmlChar *) "1.0");
02085
02086   // Setting root XML node
02087   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02088   xmlDocSetRootElement (doc, node);
02089
02090   // Adding properties to the root XML node
02091   file2 = g_file_new_for_path (input->simulator);
```

```
02092   buffer = g_file_get_relative_path (file, file2);
02093   g_object_unref (file2);
02094   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02095   g_free (buffer);
02096   if (input->evaluator)
02097     {
02098       file2 = g_file_new_for_path (input->evaluator);
02099       buffer = g_file_get_relative_path (file, file2);
02100       g_object_unref (file2);
02101       if (xmlStrlen ((xmlChar *) buffer))
02102         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02103       g_free (buffer);
02104     }
02105
02106   // Setting the algorithm
02107   buffer = (char *) g_malloc (64);
02108   switch (input->algorithm)
02109     {
02110     case ALGORITHM_MONTE_CARLO:
02111       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02112       snprintf (buffer, 64, "%u", input->nsimulations);
02113       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02114       snprintf (buffer, 64, "%u", input->niterations);
02115       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02116       snprintf (buffer, 64, "%.3lg", input->tolerance);
02117       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02118       snprintf (buffer, 64, "%u", input->nbest);
02119       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02120       break;
02121     case ALGORITHM_SWEEP:
02122       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02123       snprintf (buffer, 64, "%u", input->niterations);
02124       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%.3lg", input->tolerance);
02126       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02127       snprintf (buffer, 64, "%u", input->nbest);
02128       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02129       break;
02130     default:
02131       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02132       snprintf (buffer, 64, "%u", input->nsimulations);
02133       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->niterations);
02135       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02136       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02137       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02138       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02139       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02140       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02141       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02142       break;
02143     }
02144   g_free (buffer);
02145
02146   // Setting the experimental data
02147   for (i = 0; i < input->nexperiments; ++i)
02148     {
02149       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02150       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02151       if (input->weight[i] != 1.)
02152         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02153       for (j = 0; j < input->ninputs; ++j)
02154         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02155     }
02156
02157   // Setting the variables data
02158   for (i = 0; i < input->nvariables; ++i)
02159     {
02160       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02161       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02162       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02163       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02164         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02165       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02166       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02167         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02168       if (input->precision[i] != DEFAULT_PRECISION)
02169         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02170       if (input->algorithm == ALGORITHM_SWEEP)
02171         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
```

```
02172       else if (input->algorithm == ALGORITHM_GENETIC)
02173         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02174     }
02175
02176   // Saving the XML file
02177   xmlSaveFormatFile (filename, doc, 1);
02178
02179   // Freeing memory
02180   xmlFreeDoc (doc);
02181 }
```

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 2421 of file calibrator.c.

```
02422 {
02423   unsigned int i;
02424   for (i = 0; i < NALGORITHMS; ++i)
02425     if (gtk_toggle_button_get_active
02426         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02427       break;
02428   return i;
02429 }
```

5.7.2.4 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

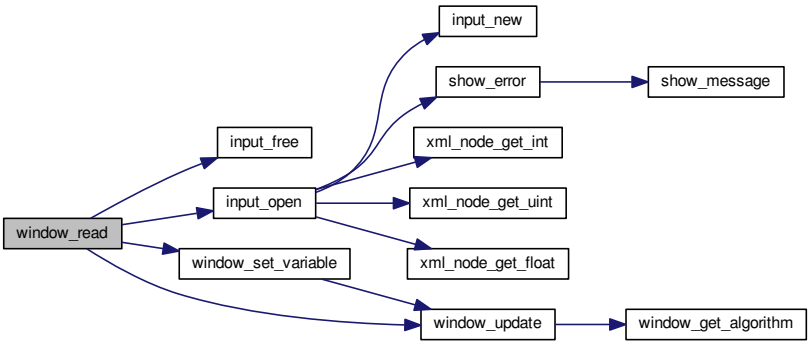
1 on succes, 0 on error.

Definition at line 3137 of file calibrator.c.

```
03138 {
03139   unsigned int i;
03140   char *buffer;
03141 #if DEBUG
03142   fprintf (stderr, "window_read: start\n");
03143 #endif
03144   input_free ();
03145   if (!input_open (filename))
03146     {
03147 #if DEBUG
03148       fprintf (stderr, "window_read: end\n");
03149 #endif
03150       return 0;
03151     }
03152   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03153   puts (buffer);
03154   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03155                                  (window->button_simulator), buffer);
03156   g_free (buffer);
03157   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03158                                 (size_t) input->evaluator);
03159   if (input->evaluator)
03160     {
03161       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03162       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03163                                      (window->button_evaluator), buffer);
03164       g_free (buffer);
03165     }
03166   gtk_toggle_button_set_active
03167     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03168   switch (input->algorithm)
03169     {
03170     case ALGORITHM_MONTE_CARLO:
03171       gtk_spin_button_set_value (window->spin_simulations,
03172                                  (gdouble) input->nsimulations);
03173     case ALGORITHM_SWEEP:
03174       gtk_spin_button_set_value (window->spin_iterations,
03175                                  (gdouble) input->niterations);
03176       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03177       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03178       break;
03179     default:
03180       gtk_spin_button_set_value (window->spin_population,
03181                                  (gdouble) input->nsimulations);
03182       gtk_spin_button_set_value (window->spin_generations,
03183                                  (gdouble) input->niterations);
03184       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03185       gtk_spin_button_set_value (window->spin_reproduction,
03186                                  input->reproduction_ratio);
03187       gtk_spin_button_set_value (window->spin_adaptation,
03188                                  input->adaptation_ratio);
03189     }
03190   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03191   g_signal_handler_block (window->button_experiment,
03192                           window->id_experiment_name);
03193   gtk_combo_box_text_remove_all (window->combo_experiment);
03194   for (i = 0; i < input->nexperiments; ++i)
03195     gtk_combo_box_text_append_text (window->combo_experiment,
03196                                     input->experiment[i]);
03197   g_signal_handler_unblock
03198     (window->button_experiment, window->
      id_experiment_name);
03199   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03200   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03201   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03202   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03203   gtk_combo_box_text_remove_all (window->combo_variable);
03204   for (i = 0; i < input->nvariables; ++i)
03205     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03206   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03207   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03208   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03209   window_set_variable ();
03210   window_update ();
```

```
03211 #if DEBUG
03212   fprintf (stderr, "window_read: end\n");
03213 #endif
03214   return 1;
03215 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

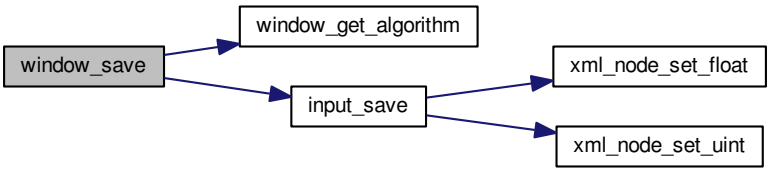
1 on OK, 0 on Cancel.

Definition at line 2240 of file calibrator.c.

```
02241 {
02242   char *buffer;
02243   GtkFileChooserDialog *dlg;
02244
02245 #if DEBUG
02246   fprintf (stderr, "window_save: start\n");
02247 #endif
02248
02249   // Opening the saving dialog
02250   dlg = (GtkFileChooserDialog *)
02251     gtk_file_chooser_dialog_new (gettext ("Save file"),
02252                                  window->window,
02253                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02254                                  gettext ("_Cancel"),
02255                                  GTK_RESPONSE_CANCEL,
02256                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02257   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02258
02259   // If OK response then saving
02260   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02261     {
02262
02263       // Adding properties to the root XML node
02264       input->simulator = gtk_file_chooser_get_filename
02265         (GTK_FILE_CHOOSER (window->button_simulator));
02266       if (gtk_toggle_button_get_active
02267           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02268         input->evaluator = gtk_file_chooser_get_filename
02269           (GTK_FILE_CHOOSER (window->button_evaluator));
02270       else
02271         input->evaluator = NULL;
02272
02273       // Setting the algorithm
02274       switch (window_get_algorithm ())
02275         {
```

```
02276         case ALGORITHM_MONTE_CARLO:
02277           input->algorithm = ALGORITHM_MONTE_CARLO;
02278           input->nsimulations
02279             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02280           input->niterations
02281             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02282           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02283           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02284           break;
02285         case ALGORITHM_SWEEP:
02286           input->algorithm = ALGORITHM_SWEEP;
02287           input->niterations
02288             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02289           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02290           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02291           break;
02292         default:
02293           input->algorithm = ALGORITHM_GENETIC;
02294           input->nsimulations
02295             = gtk_spin_button_get_value_as_int (window->spin_population);
02296           input->niterations
02297             = gtk_spin_button_get_value_as_int (window->spin_generations);
02298           input->mutation_ratio
02299             = gtk_spin_button_get_value (window->spin_mutation);
02300           input->reproduction_ratio
02301             = gtk_spin_button_get_value (window->spin_reproduction);
02302           input->adaptation_ratio
02303             = gtk_spin_button_get_value (window->spin_adaptation);
02304           break;
02305         }
02306
02307       // Saving the XML file
02308       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02309       input_save (buffer);
02310
02311       // Closing and freeing memory
02312       g_free (buffer);
02313       gtk_widget_destroy (GTK_WIDGET (dlg));
02314 #if DEBUG
02315       fprintf (stderr, "window_save: end\n");
02316 #endif
02317       return 1;
02318     }
02319
02320   // Closing and freeing memory
02321   gtk_widget_destroy (GTK_WIDGET (dlg));
02322 #if DEBUG
02323   fprintf (stderr, "window_save: end\n");
02324 #endif
02325   return 0;
02326 }
```

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 2795 of file calibrator.c.

```
02796 {
02797   unsigned int i, j;
02798   char *buffer;
02799   GFile *file1, *file2;
02800 #if DEBUG
02801   fprintf (stderr, "window_template_experiment: start\n");
02802 #endif
02803   i = (size_t) data;
02804   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02805   file1
02806     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02807   file2 = g_file_new_for_path (input->directory);
02808   buffer = g_file_get_relative_path (file2, file1);
02809   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02810   g_free (buffer);
02811   g_object_unref (file2);
02812   g_object_unref (file1);
02813 #if DEBUG
02814   fprintf (stderr, "window_template_experiment: end\n");
02815 #endif
02816 }
```

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
00043 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00044
00049 typedef struct
00050 {
00059   char *name, *template[MAX_NINPUTS];
00060   double weight;
00061 } Experiment;
00062
00067 typedef struct
00068 {
00087   char *label;
00088   double rangemin, rangemax, rangeminabs, rangemaxabs;
00089   unsigned int precision, nsweeps, nbits;
00090 } Variable;
00091
00096 typedef struct
00097 {
00108   GtkLabel *label_processors;
00109   GtkSpinButton *spin_processors;
00110   GtkGrid *grid;
```

```
00111   GtkDialog *dialog;
00112 } Options;
00113
00118 typedef struct
00119 {
00126   GtkLabel *label;
00127   GtkDialog *dialog;
00128 } Running;
00129
00134 typedef struct
00135 {
00304   GtkToolButton *button_open, *button_save, *button_run, *button_options,
00305     *button_help, *button_about, *button_exit;
00306   GtkButton *button_add_variable, *button_remove_variable,
00307     *button_add_experiment, *button_remove_experiment;
00308   GtkRadioButton *button_algorithm[NALGORITHMS];
00309   GtkCheckButton *check_evaluator, *check_minabs, *check_maxabs,
00310     *check_template[MAX_NINPUTS];
00311   GtkLabel *label_simulator, *label_simulations, *label_iterations,
00312     *label_tolerance, *label_bests, *label_population, *label_generations,
00313     *label_mutation, *label_reproduction, *label_adaptation, *label_variable,
00314     *label_min, *label_max, *label_precision, *label_sweeps, *label_bits,
00315     *label_experiment, *label_weight;
00316   GtkEntry *entry_variable;
00317   GtkComboBoxText *combo_variable, *combo_experiment;
00318   GtkFileChooserButton *button_simulator, *button_evaluator, *button_experiment,
00319     *button_template[MAX_NINPUTS];
00320   GtkSpinButton *spin_min, *spin_max, *spin_minabs, *spin_maxabs,
00321     *spin_simulations, *spin_iterations, *spin_tolerance, *spin_bests,
00322     *spin_population, *spin_generations, *spin_mutation, *spin_reproduction,
00323     *spin_adaptation, *spin_precision, *spin_sweeps, *spin_bits, *spin_weight;
00324   GtkToolbar *bar_buttons;
00325   GtkGrid *grid, *grid_algorithm, *grid_variable, *grid_experiment;
00326   GtkFrame *frame_algorithm, *frame_variable, *frame_experiment;
00327   GdkPixbuf *logo;
00328   GtkScrolledWindow *scrolled_min, *scrolled_max, *scrolled_minabs,
00329     *scrolled_maxabs;
00330   GtkWindow *window;
00331   GtkApplication *application;
00332   Experiment *experiment;
00333   Variable *variable;
00334   gulong id_experiment, id_experiment_name, id_variable, id_variable_label,
00335     id_template[MAX_NINPUTS], id_input[MAX_NINPUTS];
00336   unsigned int nexperiments, nvariables;
00337 } Window;
00338
00339 // Public functions
00340 void input_save (char *filename);
00341 void options_new ();
00342 void running_new ();
00343 int window_save ();
00344 void window_run ();
00345 void window_help ();
00346 int window_get_algorithm ();
00347 void window_update ();
00348 void window_set_algorithm ();
00349 void window_set_experiment ();
00350 void window_remove_experiment ();
00351 void window_add_experiment ();
00352 void window_name_experiment ();
00353 void window_weight_experiment ();
00354 void window_inputs_experiment ();
00355 void window_template_experiment (void *data);
00356 void window_set_variable ();
00357 void window_remove_variable ();
00358 void window_add_variable ();
00359 void window_label_variable ();
00360 void window_precision_variable ();
00361 void window_rangemin_variable ();
00362 void window_rangemax_variable ();
00363 void window_rangeminabs_variable ();
00364 void window_rangemaxabs_variable ();
00365 void window_update_variable ();
00366 int window_read (char *filename);
00367 void window_open ();
00368 void window_new ();
00369 int cores_number ();
00370
00371 #endif
```

Index