Calibrator

1.1.28

Generated by Doxygen 1.8.8

Sun Nov 1 2015 19:27:25

Contents

Chapter 1

CALIBRATOR (1.1.28 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)

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

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

OPTIONAL TOOLS AND LIBRARIES

- gtk+ (to create the interactive GUI tool)
- openmpi or mpich (to run in parallelized tasks on multiple computers)
- doxygen (standard comments format to generate documentation)
- latex (to build the PDF manuals)

FILES

The source code has to have the following files:

- configure.ac: configure generator.
- Makefile.in: Makefile generator.
- config.h.in: config header generator.
- calibrator.c: main source code.
- calibrator.h: main header code.
- interface.h: interface header code.
- build: script to build all.
- logo.png: logo figure.
- logo2.png: alternative logo figure.
- Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- README.md: this file.
- tests/testX/∗: several tests to check the program working.
- locales/∗/LC_MESSAGES/calibrator.po: translation files.
- manuals/∗.png: manual figures.
- manuals/∗.tex: documentation source files.
- applications/∗/∗: several practical application cases.
- check_errors/∗.xml: several mistaken files to check error handling.

BUILDING INSTRUCTIONS

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

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

FreeBSD 10.2

NetBSD 7.0

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/1.1.28
   $ ln -s ../../genetic/0.6.1 genetic
   ```

4. Build doing on a terminal:

$ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

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

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

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

   $ make windist

MAKING REFERENCE MANUAL INSTRUCTIONS

On UNIX type systems you need texlive installed. On Windows systems you need MiKTeX.

USER INSTRUCTIONS

- Command line in sequential mode:

  $ ./calibratorbin [-nthreads X] input_file.xml

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

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

- The syntax of the simulator has to be:

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

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

  $ ./evaluator_name simulated_file data_file results_file

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

  $ ./calibrator

INPUT FILE FORMAT

```
<?xml version="1.0"/>
<calibrate simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm_type" nsimulations="simu
    <experiment name="data_file_1" template1="template_1_1" template2="template_1_2" ... weight="weight_1"/>
    ...
    <experiment name="data_file_N" template1="template_N_1" template2="template_N_2" ... weight="weight_N"/>
    <variable name="variable_1" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="s
    ...
    <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="s
</calibrate>
```

- ∗"precision"∗ defined for each variable. Number of precision digits to evaluate the variable. 0 apply for integer numbers.
- ∗"weight"∗ defined for each experiment. Multiplies the objective value obtained for each experiment in the final objective function value.

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.
- unsigned long int seed

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

  Array of variable values.
- double * rangemin

  Array of minimum variable values.
- double * rangemax

  Array of maximum variable values.
- double * rangeminabs

  Array of absolute minimum variable values.
- double * rangemaxabs

  Array of absolute maximum variable values.
- double * error_best

  Array of the best minimum errors.
- double * weight

  Array of the experiment weights.
- double * value_old

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

  Array of the best minimum errors on the previous step.
- double tolerance

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- FILE * file_result

  Result file.
- FILE * file_variables

  Variables file.
- gsl_rng * rng

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

  Matrix of input template files.
- GeneticVariable * genetic_variable

  Array of variables for the genetic algorithm.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 92 of file calibrator.h.

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

- calibrator.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

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

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <calibrator.h>`

Data Fields

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

- char * directory
  Working directory.
- char * name
  Input data file name.
- double * rangemin
  Array of minimum variable values.
- double * rangemax
  Array of maximum variable values.
- double * rangeminabs
  Array of absolute minimum variable values.
- double * rangemaxabs
  Array of absolute maximum variable values.
- double * weight
  Array of the experiment weights.
- double tolerance
  Algorithm tolerance.
- double mutation_ratio
  Mutation probability.
- double reproduction_ratio
  Reproduction probability.
- double adaptation_ratio
  Adaptation probability.
- unsigned long int seed
  Seed of the pseudo-random numbers generator.
- unsigned int nvariables
  Variables number.
- unsigned int nexperiments
  Experiments number.
- unsigned int ninputs
  Number of input files to the simulator.
- unsigned int nsimulations
  Simulations number per experiment.
- unsigned int algorithm
  Algorithm type.
- unsigned int * precision
  Array of variable precisions.
- unsigned int * nsweeps
  Array of sweeps of the sweep algorithm.
- unsigned int * nbits
  Array of bits numbers of the genetic algorithm.
- unsigned int niterations
  Number of algorithm iterations.
- unsigned int nbest
  Number of best simulations.

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 54 of file calibrator.h.

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

- calibrator.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_processors

  Processors number GtkLabel.
- GtkSpinButton * spin_processors

  Processors number GtkSpinButton.
- GtkLabel * label_seed

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

  Pseudo-random numbers generator seed GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 74 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <calibrator.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 147 of file calibrator.h.

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

- calibrator.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkLabel * label

  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 90 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label

  Variable label.
- double rangemin

  Minimum value.
- double rangemax

  Maximum value.
- double rangeminabs

  Minimum allowed value.
- double rangemaxabs

  Maximum allowed value.
- unsigned int precision

  Precision digits.
- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file interface.h.

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

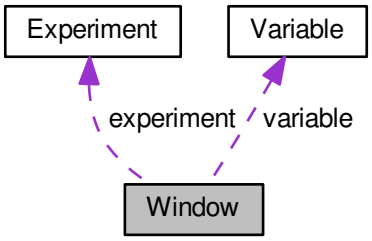
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

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

Exit GtkToolButton.

- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

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

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

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.

- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.
- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkFrame * frame_experiment

Experiment GtkFrame.

- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

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

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

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

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

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.
- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

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

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

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

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 100 of file interface.h.

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

- interface.h

Chapter 5

File Documentation

5.1 calibrator.c File Reference

Source file of the calibrator.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <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 1

  Macro to debug.
- #define ERROR_TYPE GTK_MESSAGE_ERROR

  Macro to define the error message type.
- #define INFO_TYPE GTK_MESSAGE_INFO

  Macro to define the information message type.
- #define INPUT_FILE "test-ga.xml"

  Macro to define the initial input file.

- #define RM "rm"

  Macro to define the shell remove command.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Function to calibrate with the Monte-Carlo algorithm.

- double calibrate_genetic_objective (Entity *entity)

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_free ()

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

  Function to open and perform a calibration.
- void input_save (char *filename)

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

  Function to show an about dialog.
- int window_get_algorithm ()

  Function to get the algorithm number.
- void window_update ()

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

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

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

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

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

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

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

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

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

- void window_set_variable ()

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

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

  Function to add a variable in the main window.
- void window_label_variable ()

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

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

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

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

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

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

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

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

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

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

  Function to obtain the cores number.
- int main (int argn, char **argc)

  Main function.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- GMutex mutex [1]

  Mutex struct.
- void(* calibrate_step)()

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

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

  Calibration data.
- const xmlChar * template [MAX_NINPUTS]

  Array of xmlChar strings with template labels.
- const char * format [NPRECISIONS]

  Array of C-strings with variable formats.
- const double precision [NPRECISIONS]

Array of variable precisions.

- const char * logo []

  Logo pixmap.

- Options options [1]

  Options struct to define the options dialog.

- Running running [1]

  Running struct to define the running dialog.

- Window window [1]

  Window struct to define the main interface window.

5.1.1 Detailed Description

Source file of the calibrator.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file calibrator.c.

5.1.2 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1266 of file calibrator.c.

```
01267 {
01268   unsigned int i, j;
01269   double e;
01270 #if DEBUG
01271   fprintf (stderr, "calibrate_best_sequential: start\n");
01272 #endif
01273   if (calibrate->nsaveds < calibrate->nbest
01274       || value < calibrate->error_best[calibrate->nsaveds - 1])
01275     {
01276       if (calibrate->nsaveds < calibrate->nbest)
01277         ++calibrate->nsaveds;
01278       calibrate->error_best[calibrate->nsaveds - 1] = value;
01279       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01280       for (i = calibrate->nsaveds; --i;)
01281         {
01282           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01283             {
01284               j = calibrate->simulation_best[i];
01285               e = calibrate->error_best[i];
01286               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01287               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01288               calibrate->simulation_best[i - 1] = j;
01289               calibrate->error_best[i - 1] = e;
01290             }
01291           else
```

```
01292             break;
01293         }
01294     }
01295 #if DEBUG
01296   fprintf (stderr, "calibrate_best_sequential: end\n");
01297 #endif
01298 }
```

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 1221 of file calibrator.c.

```
01222 {
01223   unsigned int i, j;
01224   double e;
01225 #if DEBUG
01226   fprintf (stderr, "calibrate_best_thread: start\n");
01227 #endif
01228   if (calibrate->nsaveds < calibrate->nbest
01229       || value < calibrate->error_best[calibrate->nsaveds - 1])
01230     {
01231       g_mutex_lock (mutex);
01232       if (calibrate->nsaveds < calibrate->nbest)
01233         ++calibrate->nsaveds;
01234       calibrate->error_best[calibrate->nsaveds - 1] = value;
01235       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01236       for (i = calibrate->nsaveds; --i;)
01237         {
01238           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01239             {
01240               j = calibrate->simulation_best[i];
01241               e = calibrate->error_best[i];
01242               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01243               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01244               calibrate->simulation_best[i - 1] = j;
01245               calibrate->error_best[i - 1] = e;
01246             }
01247           else
01248             break;
01249         }
01250       g_mutex_unlock (mutex);
01251     }
01252 #if DEBUG
01253   fprintf (stderr, "calibrate_best_thread: end\n");
01254 #endif
01255 }
```

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

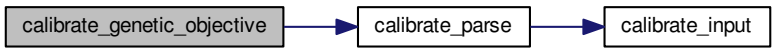
Returns

objective function value.

Definition at line 1575 of file calibrator.c.

```
01576 {
01577   unsigned int j;
01578   double objective;
01579   char buffer[64];
01580 #if DEBUG
01581   fprintf (stderr, "calibrate_genetic_objective: start\n");
01582 #endif
01583   for (j = 0; j < calibrate->nvariables; ++j)
01584     {
01585       calibrate->value[entity->id * calibrate->nvariables + j]
01586         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01587     }
01588   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01589     objective += calibrate_parse (entity->id, j);
01590   g_mutex_lock (mutex);
01591   for (j = 0; j < calibrate->nvariables; ++j)
01592     {
01593       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01594       fprintf (calibrate->file_variables, buffer,
01595                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01596     }
01597   fprintf (calibrate->file_variables, "%.14le\n", objective);
01598   g_mutex_unlock (mutex);
01599 #if DEBUG
01600   fprintf (stderr, "calibrate_genetic_objective: end\n");
01601 #endif
01602   return objective;
01603 }
```

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 970 of file calibrator.c.

```
00971 {
00972   unsigned int i;
00973   char buffer[32], value[32], *buffer2, *buffer3, *content;
00974   FILE *file;
00975   gsize length;
00976   GRegex *regex;
00977
00978 #if DEBUG
00979   fprintf (stderr, "calibrate_input: start\n");
00980 #endif
00981
00982   // Checking the file
00983   if (!template)
00984     goto calibrate_input_end;
00985
00986   // Opening template
00987   content = g_mapped_file_get_contents (template);
00988   length = g_mapped_file_get_length (template);
00989 #if DEBUG
00990   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
00991            content);
```

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

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 1382 of file calibrator.c.

```
01384 {
01385   unsigned int i, j, k, s[calibrate->nbest];
01386   double e[calibrate->nbest];
01387 #if DEBUG
01388   fprintf (stderr, "calibrate_merge: start\n");
01389 #endif
01390   i = j = k = 0;
01391   do
01392     {
01393       if (i == calibrate->nsaveds)
01394         {
01395           s[k] = simulation_best[j];
01396           e[k] = error_best[j];
```

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

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 1057 of file calibrator.c.

```
01058 {
01059   unsigned int i;
01060   double e;
01061   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01062     *buffer3, *buffer4;
01063   FILE *file_result;
01064
01065 #if DEBUG
01066   fprintf (stderr, "calibrate_parse: start\n");
01067   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01068            experiment);
01069 #endif
01070
01071   // Opening input files
01072   for (i = 0; i < calibrate->ninputs; ++i)
01073     {
01074       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01075 #if DEBUG
01076       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01077 #endif
01078       calibrate_input (simulation, &input[i][0],
01079                        calibrate->file[i][experiment]);
01080     }
01081   for (; i < MAX_NINPUTS; ++i)
01082     strcpy (&input[i][0], "");
```

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

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 1193 of file calibrator.c.

```
01194 {
01195   unsigned int i;
01196   char buffer[64];
01197 #if DEBUG
01198   fprintf (stderr, "calibrate_save_variables: start\n");
01199 #endif
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
01202       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01203       fprintf (calibrate->file_variables, buffer,
01204                calibrate->value[simulation * calibrate->
      nvariables + i]);
01205     }
01206   fprintf (calibrate->file_variables, "%.14le\n", error);
01207 #if DEBUG
01208   fprintf (stderr, "calibrate_save_variables: end\n");
01209 #endif
01210 }
```

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

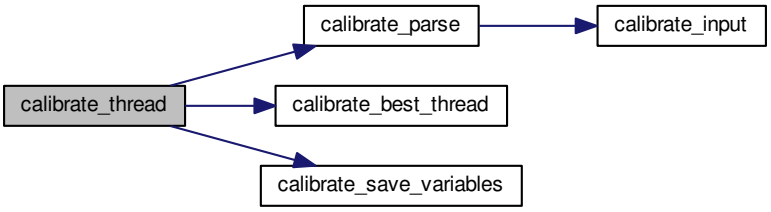
Returns

NULL

Definition at line 1308 of file calibrator.c.

```
01309 {
01310   unsigned int i, j, thread;
01311   double e;
01312 #if DEBUG
01313   fprintf (stderr, "calibrate_thread: start\n");
01314 #endif
01315   thread = data->thread;
01316 #if DEBUG
01317   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01318            calibrate->thread[thread], calibrate->thread[thread + 1]);
01319 #endif
01320   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01321     {
01322       e = 0.;
01323       for (j = 0; j < calibrate->nexperiments; ++j)
01324         e += calibrate_parse (i, j);
01325       calibrate_best_thread (i, e);
01326       g_mutex_lock (mutex);
01327       calibrate_save_variables (i, e);
01328       g_mutex_unlock (mutex);
01329 #if DEBUG
01330       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01331 #endif
01332     }
01333 #if DEBUG
01334   fprintf (stderr, "calibrate_thread: end\n");
01335 #endif
01336   g_thread_exit (NULL);
01337   return NULL;
01338 }
```

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 3775 of file calibrator.c.

```
03776 {
03777 #ifdef G_OS_WIN32
03778   SYSTEM_INFO sysinfo;
03779   GetSystemInfo (&sysinfo);
03780   return sysinfo.dwNumberOfProcessors;
03781 #else
03782   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03783 #endif
03784 }
```

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 471 of file calibrator.c.

```
00472 {
00473   char buffer2[64];
00474   xmlDoc *doc;
00475   xmlNode *node, *child;
00476   xmlChar *buffer;
00477   char *msg;
00478   int error_code;
00479   unsigned int i;
00480
00481 #if DEBUG
00482   fprintf (stderr, "input_new: start\n");
00483 #endif
00484
00485   // Resetting input data
```

```
00486   input_new ();
00487
00488   // Parsing the input file
00489   doc = xmlParseFile (filename);
00490   if (!doc)
00491     {
00492       msg = gettext ("Unable to parse the input file");
00493       goto exit_on_error;
00494     }
00495
00496   // Getting the root node
00497   node = xmlDocGetRootElement (doc);
00498   if (xmlStrcmp (node->name, XML_CALIBRATE))
00499     {
00500       msg = gettext ("Bad root XML node");
00501       goto exit_on_error;
00502     }
00503
00504   // Opening simulator program name
00505   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00506   if (!input->simulator)
00507     {
00508       msg = gettext ("Bad simulator program");
00509       goto exit_on_error;
00510     }
00511
00512   // Opening evaluator program name
00513   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00514
00515   // Obtaining pseudo-random numbers generator seed
00516   if (!xmlHasProp (node, XML_SEED))
00517     input->seed = DEFAULT_RANDOM_SEED;
00518   else
00519     {
00520       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00521       if (error_code)
00522         {
00523           msg = gettext ("Bad pseudo-random numbers generator seed");
00524           goto exit_on_error;
00525         }
00526     }
00527
00528   // Opening algorithm
00529   buffer = xmlGetProp (node, XML_ALGORITHM);
00530   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00531     {
00532       input->algorithm = ALGORITHM_MONTE_CARLO;
00533
00534       // Obtaining simulations number
00535       input->nsimulations
00536         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00537       if (error_code)
00538         {
00539           msg = gettext ("Bad simulations number");
00540           goto exit_on_error;
00541         }
00542     }
00543   else if (!xmlStrcmp (buffer, XML_SWEEP))
00544     input->algorithm = ALGORITHM_SWEEP;
00545   else if (!xmlStrcmp (buffer, XML_GENETIC))
00546     {
00547       input->algorithm = ALGORITHM_GENETIC;
00548
00549       // Obtaining population
00550       if (xmlHasProp (node, XML_NPOPULATION))
00551         {
00552           input->nsimulations
00553             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00554           if (error_code || input->nsimulations < 3)
00555             {
00556               msg = gettext ("Invalid population number");
00557               goto exit_on_error;
00558             }
00559         }
00560       else
00561         {
00562           msg = gettext ("No population number");
00563           goto exit_on_error;
00564         }
00565
00566       // Obtaining generations
00567       if (xmlHasProp (node, XML_NGENERATIONS))
00568         {
00569           input->niterations
00570             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00571           if (error_code || !input->niterations)
00572             {
```

```
00573               msg = gettext ("Invalid generations number");
00574               goto exit_on_error;
00575             }
00576         }
00577       else
00578         {
00579           msg = gettext ("No generations number");
00580           goto exit_on_error;
00581         }
00582
00583       // Obtaining mutation probability
00584       if (xmlHasProp (node, XML_MUTATION))
00585         {
00586           input->mutation_ratio
00587             = xml_node_get_float (node, XML_MUTATION, &error_code);
00588           if (error_code || input->mutation_ratio < 0.
00589               || input->mutation_ratio >= 1.)
00590             {
00591               msg = gettext ("Invalid mutation probability");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No mutation probability");
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining reproduction probability
00602       if (xmlHasProp (node, XML_REPRODUCTION))
00603         {
00604           input->reproduction_ratio
00605             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00606           if (error_code || input->reproduction_ratio < 0.
00607               || input->reproduction_ratio >= 1.0)
00608             {
00609               msg = gettext ("Invalid reproduction probability");
00610               goto exit_on_error;
00611             }
00612         }
00613       else
00614         {
00615           msg = gettext ("No reproduction probability");
00616           goto exit_on_error;
00617         }
00618
00619       // Obtaining adaptation probability
00620       if (xmlHasProp (node, XML_ADAPTATION))
00621         {
00622           input->adaptation_ratio
00623             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00624           if (error_code || input->adaptation_ratio < 0.
00625               || input->adaptation_ratio >= 1.)
00626             {
00627               msg = gettext ("Invalid adaptation probability");
00628               goto exit_on_error;
00629             }
00630         }
00631       else
00632         {
00633           msg = gettext ("No adaptation probability");
00634           goto exit_on_error;
00635         }
00636
00637       // Checking survivals
00638       i = input->mutation_ratio * input->nsimulations;
00639       i += input->reproduction_ratio * input->
     nsimulations;
00640       i += input->adaptation_ratio * input->
     nsimulations;
00641       if (i > input->nsimulations - 2)
00642         {
00643           msg = gettext
00644             ("No enough survival entities to reproduce the population");
00645           goto exit_on_error;
00646         }
00647     }
00648   else
00649     {
00650       msg = gettext ("Unknown algorithm");
00651       goto exit_on_error;
00652     }
00653
00654   if (input->algorithm == ALGORITHM_MONTE_CARLO
00655       || input->algorithm == ALGORITHM_SWEEP)
00656     {
00657
```

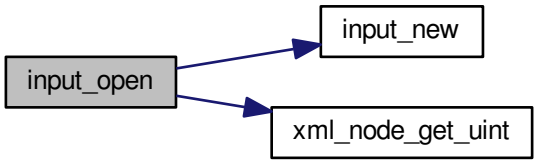
```
00658       // Obtaining iterations number
00659       input->niterations
00660         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00661       if (error_code == 1)
00662         input->niterations = 1;
00663       else if (error_code)
00664         {
00665           msg = gettext ("Bad iterations number");
00666           goto exit_on_error;
00667         }
00668
00669       // Obtaining best number
00670       if (xmlHasProp (node, XML_NBEST))
00671         {
00672           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00673           if (error_code || !input->nbest)
00674             {
00675               msg = gettext ("Invalid best number");
00676               goto exit_on_error;
00677             }
00678         }
00679       else
00680         input->nbest = 1;
00681
00682       // Obtaining tolerance
00683       if (xmlHasProp (node, XML_TOLERANCE))
00684         {
00685           input->tolerance
00686             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00687           if (error_code || input->tolerance < 0.)
00688             {
00689               msg = gettext ("Invalid tolerance");
00690               goto exit_on_error;
00691             }
00692         }
00693       else
00694         input->tolerance = 0.;
00695     }
00696
00697   // Reading the experimental data
00698   for (child = node->children; child; child = child->next)
00699     {
00700       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00701         break;
00702 #if DEBUG
00703       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00704 #endif
00705       if (xmlHasProp (child, XML_NAME))
00706         {
00707           input->experiment
00708             = g_realloc (input->experiment,
00709                          (1 + input->nexperiments) * sizeof (char *));
00710           input->experiment[input->nexperiments]
00711             = (char *) xmlGetProp (child, XML_NAME);
00712         }
00713       else
00714         {
00715           msg = gettext ("No experiment file name");
00716           goto exit_on_error;
00717         }
00718 #if DEBUG
00719       fprintf (stderr, "input_new: experiment=%s\n",
00720                input->experiment[input->nexperiments]);
00721 #endif
00722       input->weight = g_realloc (input->weight,
00723                                  (1 + input->nexperiments) * sizeof (double));
00724       if (xmlHasProp (child, XML_WEIGHT))
00725         {
00726           input->weight[input->nexperiments]
00727             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00728           if (error_code)
00729             {
00730               msg = gettext ("Bad weight");
00731               goto exit_on_error;
00732             }
00733         }
00734       else
00735         input->weight[input->nexperiments] = 1.;
00736 #if DEBUG
00737       fprintf (stderr, "input_new: weight=%lg\n",
00738                input->weight[input->nexperiments]);
00739 #endif
00740       if (!input->nexperiments)
00741         input->ninputs = 0;
00742 #if DEBUG
00743       fprintf (stderr, "input_new: template[0]\n");
```

```
00744 #endif
00745       if (xmlHasProp (child, XML_TEMPLATE1))
00746         {
00747           input->template[0]
00748             = (char **) g_realloc (input->template[0],
00749                                    (1 + input->nexperiments) * sizeof (char *));
00750           input->template[0][input->nexperiments]
00751             = (char *) xmlGetProp (child, template[0]);
00752 #if DEBUG
00753           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00754                    input->nexperiments,
00755                    input->template[0][input->nexperiments]);
00756 #endif
00757           if (!input->nexperiments)
00758             ++input->ninputs;
00759 #if DEBUG
00760           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00761 #endif
00762         }
00763       else
00764         {
00765           msg = gettext ("No experiment template");
00766           goto exit_on_error;
00767         }
00768       for (i = 1; i < MAX_NINPUTS; ++i)
00769         {
00770 #if DEBUG
00771           fprintf (stderr, "input_new: template%u\n", i + 1);
00772 #endif
00773           if (xmlHasProp (child, template[i]))
00774             {
00775               if (input->nexperiments && input->ninputs < 2)
00776                 {
00777                   snprintf (buffer2, 64,
00778                             gettext ("Experiment %u: bad templates number"),
00779                             input->nexperiments + 1);
00780                   msg = buffer2;
00781                   goto exit_on_error;
00782                 }
00783               input->template[i] = (char **)
00784                 g_realloc (input->template[i],
00785                            (1 + input->nexperiments) * sizeof (char *));
00786               input->template[i][input->nexperiments]
00787                 = (char *) xmlGetProp (child, template[i]);
00788 #if DEBUG
00789               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00790                        input->nexperiments, i + 1,
00791                        input->template[i][input->nexperiments]);
00792 #endif
00793               if (!input->nexperiments)
00794                 ++input->ninputs;
00795 #if DEBUG
00796               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00797 #endif
00798             }
00799           else if (input->nexperiments && input->ninputs > 1)
00800             {
00801               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00802                         input->nexperiments + 1, i + 1);
00803               msg = buffer2;
00804               goto exit_on_error;
00805             }
00806           else
00807             break;
00808         }
00809       ++input->nexperiments;
00810 #if DEBUG
00811       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00812 #endif
00813     }
00814   if (!input->nexperiments)
00815     {
00816       msg = gettext ("No calibration experiments");
00817       goto exit_on_error;
00818     }
00819
00820   // Reading the variables data
00821   for (; child; child = child->next)
00822     {
00823       if (xmlStrcmp (child->name, XML_VARIABLE))
00824         {
00825           msg = gettext ("Bad XML node");
00826           goto exit_on_error;
00827         }
00828       if (xmlHasProp (child, XML_NAME))
00829         {
00830           input->label = g_realloc
```

```
00831             (input->label, (1 + input->nvariables) * sizeof (char *));
00832           input->label[input->nvariables]
00833             = (char *) xmlGetProp (child, XML_NAME);
00834         }
00835       else
00836         {
00837           msg = gettext ("No variable name");
00838           goto exit_on_error;
00839         }
00840       if (xmlHasProp (child, XML_MINIMUM))
00841         {
00842           input->rangemin = g_realloc
00843             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00844           input->rangeminabs = g_realloc
00845             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00846           input->rangemin[input->nvariables]
00847             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00848           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00849             {
00850               input->rangeminabs[input->nvariables]
00851                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00852             }
00853           else
00854             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00855         }
00856       else
00857         {
00858           msg = gettext ("No minimum range");
00859           goto exit_on_error;
00860         }
00861       if (xmlHasProp (child, XML_MAXIMUM))
00862         {
00863           input->rangemax = g_realloc
00864             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00865           input->rangemaxabs = g_realloc
00866             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00867           input->rangemax[input->nvariables]
00868             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00869           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00870             input->rangemaxabs[input->nvariables]
00871               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00872           else
00873             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00874         }
00875       else
00876         {
00877           msg = gettext ("No maximum range");
00878           goto exit_on_error;
00879         }
00880       input->precision = g_realloc
00881         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00882       if (xmlHasProp (child, XML_PRECISION))
00883         input->precision[input->nvariables]
00884           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00885       else
00886         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00887       if (input->algorithm == ALGORITHM_SWEEP)
00888         {
00889           if (xmlHasProp (child, XML_NSWEEPS))
00890             {
00891               input->nsweeps = (unsigned int *)
00892                 g_realloc (input->nsweeps,
00893                            (1 + input->nvariables) * sizeof (unsigned int));
00894               input->nsweeps[input->nvariables]
00895                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00896             }
00897           else
00898             {
00899               msg = gettext ("No sweeps number");
00900               goto exit_on_error;
00901             }
00902 #if DEBUG
00903           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00904                    input->nsweeps[input->nvariables],
      input->nsimulations);
00905 #endif
00906         }
00907       if (input->algorithm == ALGORITHM_GENETIC)
00908         {
00909           // Obtaining bits representing each variable
00910           if (xmlHasProp (child, XML_NBITS))
00911             {
00912               input->nbits = (unsigned int *)
00913                 g_realloc (input->nbits,
```

```
00914                           (1 + input->nvariables) * sizeof (unsigned int));
00915               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00916               if (error_code || !i)
00917                 {
00918                   msg = gettext ("Invalid bit number");
00919                   goto exit_on_error;
00920                 }
00921               input->nbits[input->nvariables] = i;
00922             }
00923           else
00924             {
00925               msg = gettext ("No bits number");
00926               goto exit_on_error;
00927             }
00928         }
00929       ++input->nvariables;
00930     }
00931   if (!input->nvariables)
00932     {
00933       msg = gettext ("No calibration variables");
00934       goto exit_on_error;
00935     }
00936
00937   // Getting the working directory
00938   input->directory = g_path_get_dirname (filename);
00939   input->name = g_path_get_basename (filename);
00940
00941   // Closing the XML document
00942   xmlFreeDoc (doc);
00943
00944 #if DEBUG
00945   fprintf (stderr, "input_new: end\n");
00946 #endif
00947   return 1;
00948
00949 exit_on_error:
00950   show_error (msg);
00951   input_free ();
00952 #if DEBUG
00953   fprintf (stderr, "input_new: end\n");
00954 #endif
00955   return 0;
00956 }
```

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.

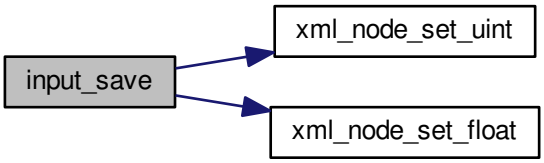
Definition at line 2075 of file calibrator.c.

```
02076 {
02077   unsigned int i, j;
02078   char *buffer;
```

```
02079   xmlDoc *doc;
02080   xmlNode *node, *child;
02081   GFile *file, *file2;
02082
02083   // Getting the input file directory
02084   input->name = g_path_get_basename (filename);
02085   input->directory = g_path_get_dirname (filename);
02086   file = g_file_new_for_path (input->directory);
02087
02088   // Opening the input file
02089   doc = xmlNewDoc ((const xmlChar *) "1.0");
02090
02091   // Setting root XML node
02092   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02093   xmlDocSetRootElement (doc, node);
02094
02095   // Adding properties to the root XML node
02096   file2 = g_file_new_for_path (input->simulator);
02097   buffer = g_file_get_relative_path (file, file2);
02098   g_object_unref (file2);
02099   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02100   g_free (buffer);
02101   if (input->evaluator)
02102     {
02103       file2 = g_file_new_for_path (input->evaluator);
02104       buffer = g_file_get_relative_path (file, file2);
02105       g_object_unref (file2);
02106       if (xmlStrlen ((xmlChar *) buffer))
02107         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02108       g_free (buffer);
02109     }
02110   if (input->seed != DEFAULT_RANDOM_SEED)
02111     xml_node_set_uint (node, XML_SEED, input->seed);
02112
02113   // Setting the algorithm
02114   buffer = (char *) g_malloc (64);
02115   switch (input->algorithm)
02116     {
02117     case ALGORITHM_MONTE_CARLO:
02118       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02119       snprintf (buffer, 64, "%u", input->nsimulations);
02120       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02121       snprintf (buffer, 64, "%u", input->niterations);
02122       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02123       snprintf (buffer, 64, "%.3lg", input->tolerance);
02124       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%u", input->nbest);
02126       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02127       break;
02128     case ALGORITHM_SWEEP:
02129       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02130       snprintf (buffer, 64, "%u", input->niterations);
02131       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02132       snprintf (buffer, 64, "%.3lg", input->tolerance);
02133       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->nbest);
02135       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02136       break;
02137     default:
02138       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02139       snprintf (buffer, 64, "%u", input->nsimulations);
02140       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02141       snprintf (buffer, 64, "%u", input->niterations);
02142       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02143       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02144       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02145       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02146       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02147       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02148       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02149       break;
02150     }
02151   g_free (buffer);
02152
02153   // Setting the experimental data
02154   for (i = 0; i < input->nexperiments; ++i)
02155     {
02156       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02157       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02158       if (input->weight[i] != 1.)
02159         xml_node_set_float (child, XML_WEIGHT, input->
    weight[i]);
02160       for (j = 0; j < input->ninputs; ++j)
02161         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02162     }
02163
02164   // Setting the variables data
```

```
02165   for (i = 0; i < input->nvariables; ++i)
02166     {
02167       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02168       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02169       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02170       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02171         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02172       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02173       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02174         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02175       if (input->precision[i] != DEFAULT_PRECISION)
02176         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02177       if (input->algorithm == ALGORITHM_SWEEP)
02178         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02179       else if (input->algorithm == ALGORITHM_GENETIC)
02180         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02181     }
02182
02183   // Saving the XML file
02184   xmlSaveFormatFile (filename, doc, 1);
02185
02186   // Freeing memory
02187   xmlFreeDoc (doc);
02188 }
```

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

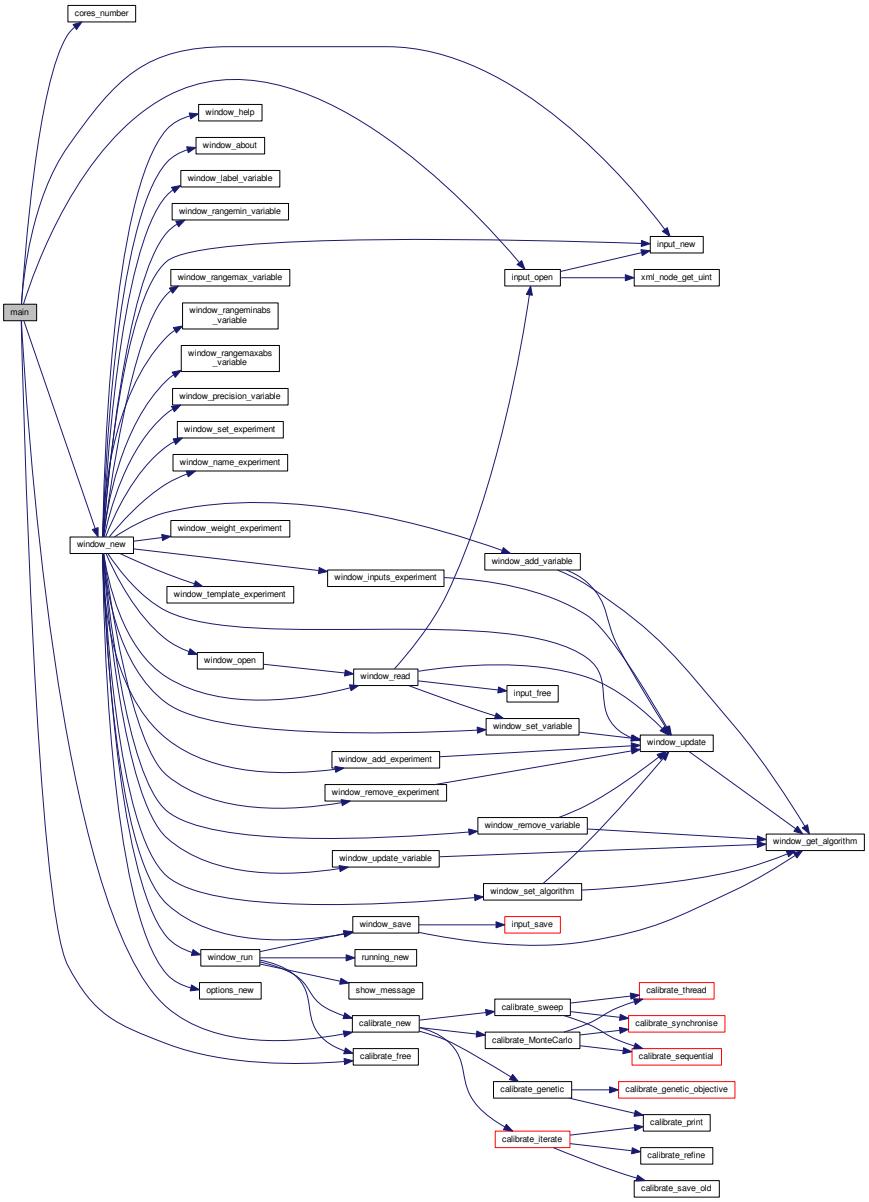
0 on success, >0 on error.

Definition at line 3796 of file calibrator.c.

```
03797 {
03798   // Starting pseudo-random numbers generator
03799   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03800   calibrate->seed = DEFAULT_RANDOM_SEED;
03801
03802   // Allowing spaces in the XML data file
03803   xmlKeepBlanksDefault (0);
```

```
03804
03805   // Starting MPI
03806 #if HAVE_MPI
03807   MPI_Init (&argn, &argc);
03808   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03809   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03810   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03811 #else
03812   ntasks = 1;
03813 #endif
03814
03815 #if HAVE_GTK
03816
03817   // Getting threads number
03818   nthreads = cores_number ();
03819
03820   // Setting local language and international floating point numbers notation
03821   setlocale (LC_ALL, "");
03822   setlocale (LC_NUMERIC, "C");
03823   window->application_directory = g_get_current_dir ();
03824   bindtextdomain (PROGRAM_INTERFACE,
03825                   g_build_filename (window->application_directory,
03826                                     LOCALE_DIR, NULL));
03827   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03828   textdomain (PROGRAM_INTERFACE);
03829
03830   // Initing GTK+
03831   gtk_disable_setlocale ();
03832   gtk_init (&argn, &argc);
03833
03834   // Opening the main window
03835   window_new ();
03836   gtk_main ();
03837
03838   // Freeing memory
03839   gtk_widget_destroy (GTK_WIDGET (window->window));
03840   g_free (window->application_directory);
03841
03842 #else
03843
03844   // Checking syntax
03845   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03846     {
03847       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03848       return 1;
03849     }
03850
03851   // Getting threads number
03852   if (argn == 2)
03853     nthreads = cores_number ();
03854   else
03855     nthreads = atoi (argc[2]);
03856   printf ("nthreads=%u\n", nthreads);
03857
03858   // Making calibration
03859   input_new ();
03860   if (input_open (argc[argn - 1]))
03861     calibrate_new ();
03862
03863   // Freeing memory
03864   calibrate_free ();
03865
03866 #endif
03867
03868   // Closing MPI
03869 #if HAVE_MPI
03870   MPI_Finalize ();
03871 #endif
03872
03873   // Freeing memory
03874   gsl_rng_free (calibrate->rng);
03875
03876   // Closing
03877   return 0;
03878 }
```

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.

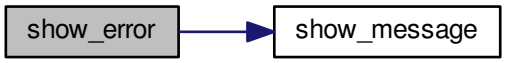
Parameters

msg	Error message.

Definition at line 245 of file calibrator.c.

```
00246 {
00247   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00248 }
```

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 215 of file calibrator.c.

```
00216 {
00217 #if HAVE_GTK
00218   GtkMessageDialog *dlg;
00219
00220   // Creating the dialog
00221   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00222     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00223
00224   // Setting the dialog title
00225   gtk_window_set_title (GTK_WINDOW (dlg), title);
00226
00227   // Showing the dialog and waiting response
00228   gtk_dialog_run (GTK_DIALOG (dlg));
00229
00230   // Closing and freeing memory
00231   gtk_widget_destroy (GTK_WIDGET (dlg));
00232
00233 #else
00234   printf ("%s: %s\n", title, msg);
00235 #endif
00236 }
```

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2444 of file calibrator.c.

```
02445 {
02446   unsigned int i;
02447   for (i = 0; i < NALGORITHMS; ++i)
02448     if (gtk_toggle_button_get_active
02449         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02450       break;
02451   return i;
02452 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

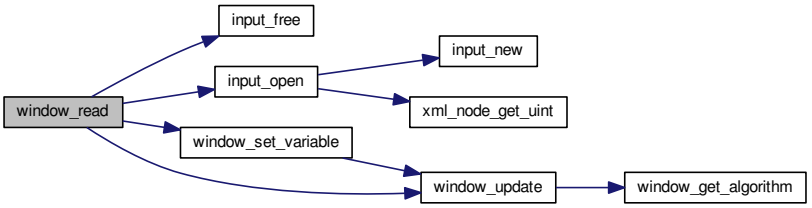
1 on succes, 0 on error.

Definition at line 3160 of file calibrator.c.

```
03161 {
03162   unsigned int i;
03163   char *buffer, *directory, *name;
03164 #if DEBUG
03165   fprintf (stderr, "window_read: start\n");
03166 #endif
03167   directory = name = NULL;
03168   if (input->directory) directory = g_strdup (input->
      directory);
03169   if (input->name) name = g_strdup (input->name);
03170   input_free ();
03171   if (!input_open (filename))
03172     {
03173 #if DEBUG
03174           fprintf (stderr, "window_read: error reading input file\n");
03175 #endif
03176       buffer = g_build_filename (directory, name, NULL);
03177       if (!input_open (buffer))
03178         {
03179 #if DEBUG
03180           fprintf (stderr, "window_read: error reading backup file\n");
03181 #endif
03182           g_free (buffer);
03183           g_free (name);
03184           g_free (directory);
03185 #if DEBUG
03186           fprintf (stderr, "window_read: end\n");
03187 #endif
03188           return 0;
03189         }
03190       g_free (buffer);
03191     }
03192   g_free (name);
03193   g_free (directory);
03194   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03195   puts (buffer);
03196   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03197                                  (window->button_simulator), buffer);
03198   g_free (buffer);
03199   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03200                                 (size_t) input->evaluator);
03201   if (input->evaluator)
03202     {
03203       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03204       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03205                                      (window->button_evaluator), buffer);
03206       g_free (buffer);
03207     }
03208   gtk_toggle_button_set_active
03209     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03210   switch (input->algorithm)
03211     {
03212     case ALGORITHM_MONTE_CARLO:
03213       gtk_spin_button_set_value (window->spin_simulations,
03214                                  (gdouble) input->nsimulations);
03215     case ALGORITHM_SWEEP:
03216       gtk_spin_button_set_value (window->spin_iterations,
03217                                  (gdouble) input->niterations);
03218       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03219       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03220       break;
03221     default:
03222       gtk_spin_button_set_value (window->spin_population,
03223                                  (gdouble) input->nsimulations);
03224       gtk_spin_button_set_value (window->spin_generations,
03225                                  (gdouble) input->niterations);
03226       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
```

```
03227       gtk_spin_button_set_value (window->spin_reproduction,
03228                                  input->reproduction_ratio);
03229       gtk_spin_button_set_value (window->spin_adaptation,
03230                                  input->adaptation_ratio);
03231     }
03232   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03233   g_signal_handler_block (window->button_experiment,
03234                           window->id_experiment_name);
03235   gtk_combo_box_text_remove_all (window->combo_experiment);
03236   for (i = 0; i < input->nexperiments; ++i)
03237     gtk_combo_box_text_append_text (window->combo_experiment,
03238                                     input->experiment[i]);
03239   g_signal_handler_unblock
03240     (window->button_experiment, window->
      id_experiment_name);
03241   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03242   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03243   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03244   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03245   gtk_combo_box_text_remove_all (window->combo_variable);
03246   for (i = 0; i < input->nvariables; ++i)
03247     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03248   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03249   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03250   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03251   window_set_variable ();
03252   window_update ();
03253 #if DEBUG
03254   fprintf (stderr, "window_read: end\n");
03255 #endif
03256   return 1;
03257 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

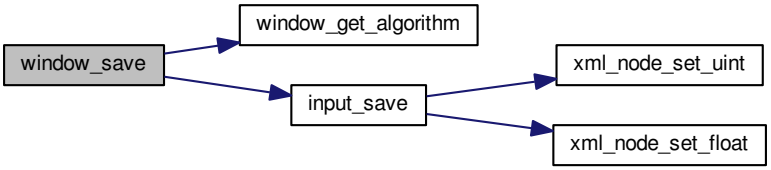
1 on OK, 0 on Cancel.

Definition at line 2263 of file calibrator.c.

```
02264 {
02265   char *buffer;
02266   GtkFileChooserDialog *dlg;
02267
02268 #if DEBUG
02269   fprintf (stderr, "window_save: start\n");
02270 #endif
```

```
02271
02272   // Opening the saving dialog
02273   dlg = (GtkFileChooserDialog *)
02274     gtk_file_chooser_dialog_new (gettext ("Save file"),
02275                                  window->window,
02276                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02277                                  gettext ("_Cancel"),
02278                                  GTK_RESPONSE_CANCEL,
02279                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02280   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02281
02282   // If OK response then saving
02283   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02284     {
02285
02286       // Adding properties to the root XML node
02287       input->simulator = gtk_file_chooser_get_filename
02288         (GTK_FILE_CHOOSER (window->button_simulator));
02289       if (gtk_toggle_button_get_active
02290           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02291         input->evaluator = gtk_file_chooser_get_filename
02292           (GTK_FILE_CHOOSER (window->button_evaluator));
02293       else
02294         input->evaluator = NULL;
02295
02296       // Setting the algorithm
02297       switch (window_get_algorithm ())
02298         {
02299         case ALGORITHM_MONTE_CARLO:
02300           input->algorithm = ALGORITHM_MONTE_CARLO;
02301           input->nsimulations
02302             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02303           input->niterations
02304             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02305           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02306           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02307           break;
02308         case ALGORITHM_SWEEP:
02309           input->algorithm = ALGORITHM_SWEEP;
02310           input->niterations
02311             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02312           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02313           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02314           break;
02315         default:
02316           input->algorithm = ALGORITHM_GENETIC;
02317           input->nsimulations
02318             = gtk_spin_button_get_value_as_int (window->spin_population);
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_generations);
02321           input->mutation_ratio
02322             = gtk_spin_button_get_value (window->spin_mutation);
02323           input->reproduction_ratio
02324             = gtk_spin_button_get_value (window->spin_reproduction);
02325           input->adaptation_ratio
02326             = gtk_spin_button_get_value (window->spin_adaptation);
02327           break;
02328         }
02329
02330       // Saving the XML file
02331       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02332       input_save (buffer);
02333
02334       // Closing and freeing memory
02335       g_free (buffer);
02336       gtk_widget_destroy (GTK_WIDGET (dlg));
02337 #if DEBUG
02338       fprintf (stderr, "window_save: end\n");
02339 #endif
02340       return 1;
02341     }
02342
02343   // Closing and freeing memory
02344   gtk_widget_destroy (GTK_WIDGET (dlg));
02345 #if DEBUG
02346   fprintf (stderr, "window_save: end\n");
02347 #endif
02348   return 0;
02349 }
```

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 2818 of file calibrator.c.

```
02819 {
02820   unsigned int i, j;
02821   char *buffer;
02822   GFile *file1, *file2;
02823 #if DEBUG
02824   fprintf (stderr, "window_template_experiment: start\n");
02825 #endif
02826   i = (size_t) data;
02827   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02828   file1
02829     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02830   file2 = g_file_new_for_path (input->directory);
02831   buffer = g_file_get_relative_path (file2, file1);
02832   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02833   g_free (buffer);
02834   g_object_unref (file2);
02835   g_object_unref (file1);
02836 #if DEBUG
02837   fprintf (stderr, "window_template_experiment: end\n");
02838 #endif
02839 }
```

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 324 of file calibrator.c.

```
00325 {
```

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

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 262 of file calibrator.c.

```
00263 {
00264   int i = 0;
00265   xmlChar *buffer;
00266   buffer = xmlGetProp (node, prop);
00267   if (!buffer)
00268     *error_code = 1;
00269   else
00270     {
00271       if (sscanf ((char *) buffer, "%d", &i) != 1)
00272         *error_code = 2;
00273       else
00274         *error_code = 0;
00275       xmlFree (buffer);
00276     }
00277   return i;
00278 }
```

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 293 of file calibrator.c.

```
00294 {
00295   unsigned int i = 0;
00296   xmlChar *buffer;
00297   buffer = xmlGetProp (node, prop);
00298   if (!buffer)
00299     *error_code = 1;
00300   else
00301     {
00302       if (sscanf ((char *) buffer, "%u", &i) != 1)
00303         *error_code = 2;
00304       else
00305         *error_code = 0;
00306       xmlFree (buffer);
00307     }
00308   return i;
00309 }
```

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 391 of file calibrator.c.

```
00392 {
00393   xmlChar buffer[64];
00394   snprintf ((char *) buffer, 64, "%.14lg", value);
00395   xmlSetProp (node, prop, buffer);
00396 }
```

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 353 of file calibrator.c.

```
00354 {
00355   xmlChar buffer[64];
00356   snprintf ((char *) buffer, 64, "%d", value);
00357   xmlSetProp (node, prop, buffer);
00358 }
```

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 372 of file calibrator.c.

```
00373 {
00374   xmlChar buffer[64];
00375   snprintf ((char *) buffer, 64, "%u", value);
00376   xmlSetProp (node, prop, buffer);
00377 }
```

5.1.3 Variable Documentation

5.1.3.1 const char* format[NPRECISIONS]

Initial value:

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

Array of C-strings with variable formats.

Definition at line 106 of file calibrator.c.

5.1.3.2 const double precision[NPRECISIONS]

Initial value:

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

Array of variable precisions.

Definition at line 111 of file calibrator.c.

5.1.3.3 const xmlChar* template[MAX_NINPUTS]

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 99 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
00074 #define DEBUG 1
00075 #if HAVE_GTK
00076 #define ERROR_TYPE GTK_MESSAGE_ERROR
00077 #define INFO_TYPE GTK_MESSAGE_INFO
00078 #else
00079 #define ERROR_TYPE 0
00080 #define INFO_TYPE 0
00081 #endif
00082 #ifdef G_OS_WIN32
00083 #define INPUT_FILE "test-ga-win.xml"
00084 #define RM "del"
00085 #else
00086 #define INPUT_FILE "test-ga.xml"
00087 #define RM "rm"
00088 #endif
00089
00090 int ntasks;
00091 unsigned int nthreads;
00092 GMutex mutex[1];
00093 void (*calibrate_step) ();
00095 Input input[1];
00097 Calibrate calibrate[1];
00098
00099 const xmlChar *template[MAX_NINPUTS] = {
00100   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00101   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00102 };
00103
00105
00106 const char *format[NPRECISIONS] = {
00107   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00108   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00109 };
00110
00111 const double precision[NPRECISIONS] = {
```

```
00112   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,
00113   1e-13, 1e-14
00114 };
00115
00116 const char *logo[] = {
00117   "32 32 3 1",
00118   "      c None",
00119   ".     c #0000FF",
00120   "+     c #FF0000",
00121   "                                ",
00122   "                                ",
00123   "                                ",
00124   "      .       .       .       ",
00125   "      .       .       .       ",
00126   "      .       .       .       ",
00127   "      .       .       .       ",
00128   "      .       .      +++      ",
00129   "      .       .     +++++     ",
00130   "      .       .     +++++     ",
00131   "      .       .     +++++     ",
00132   "     +++      .      +++     +++     ",
00133   "    +++++     .       .     +++++    ",
00134   "    +++++     .       .     +++++    ",
00135   "    +++++     .       .     +++++    ",
00136   "     +++      .       .      +++     ",
00137   "      .       .       .       .      ",
00138   "      .      +++      .       .      ",
00139   "      .     +++++     .       .      ",
00140   "      .     +++++     .       .      ",
00141   "      .     +++++     .       .      ",
00142   "      .      +++      .       .      ",
00143   "      .       .       .       .      ",
00144   "      .       .       .       .      ",
00145   "      .       .       .       .      ",
00146   "      .       .       .       .      ",
00147   "      .       .       .       .      ",
00148   "      .       .       .       .      ",
00149   "      .       .       .       .      ",
00150   "                                ",
00151   "                                ",
00152   "                                "
00153 };
00154
00155 /*
00156 const char * logo[] = {
00157 "32 32 3 1",
00158 "   c #FFFFFFFFFFFF",
00159 ".  c #00000000FFFF",
00160 "X  c #FFFF00000000",
00161 "                                ",
00162 "                                ",
00163 "                                ",
00164 "      .       .       .       ",
00165 "      .       .       .       ",
00166 "      .       .       .       ",
00167 "      .       .       .       ",
00168 "      .       .      XXX      ",
00169 "      .       .     XXXXX     ",
00170 "      .       .     XXXXX     ",
00171 "      .       .     XXXXX     ",
00172 "     XXX      .      XXX     XXX     ",
00173 "    XXXXX     .       .     XXXXX    ",
00174 "    XXXXX     .       .     XXXXX    ",
00175 "    XXXXX     .       .     XXXXX    ",
00176 "     XXX      .       .      XXX     ",
00177 "      .       .       .       .      ",
00178 "      .      XXX      .       .      ",
00179 "      .     XXXXX     .       .      ",
00180 "      .     XXXXX     .       .      ",
00181 "      .     XXXXX     .       .      ",
00182 "      .      XXX      .       .      ",
00183 "      .       .       .       .      ",
00184 "      .       .       .       .      ",
00185 "      .       .       .       .      ",
00186 "      .       .       .       .      ",
00187 "      .       .       .       .      ",
00188 "      .       .       .       .      ",
00189 "      .       .       .       .      ",
00190 "                                ",
00191 "                                ",
00192 "                                "};
00193 */
00194
00195 #if HAVE_GTK
00196 Options options[1];
00198 Running running[1];
00200 Window window[1];
```

```
00202 #endif
00203
00214 void
00215 show_message (char *title, char *msg, int type)
00216 {
00217 #if HAVE_GTK
00218   GtkMessageDialog *dlg;
00219
00220   // Creating the dialog
00221   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00222     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00223
00224   // Setting the dialog title
00225   gtk_window_set_title (GTK_WINDOW (dlg), title);
00226
00227   // Showing the dialog and waiting response
00228   gtk_dialog_run (GTK_DIALOG (dlg));
00229
00230   // Closing and freeing memory
00231   gtk_widget_destroy (GTK_WIDGET (dlg));
00232
00233 #else
00234   printf ("%s: %s\n", title, msg);
00235 #endif
00236 }
00237
00244 void
00245 show_error (char *msg)
00246 {
00247   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00248 }
00249
00261 int
00262 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00263 {
00264   int i = 0;
00265   xmlChar *buffer;
00266   buffer = xmlGetProp (node, prop);
00267   if (!buffer)
00268     *error_code = 1;
00269   else
00270     {
00271       if (sscanf ((char *) buffer, "%d", &i) != 1)
00272         *error_code = 2;
00273       else
00274         *error_code = 0;
00275       xmlFree (buffer);
00276     }
00277   return i;
00278 }
00279
00292 unsigned int
00293 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00294 {
00295   unsigned int i = 0;
00296   xmlChar *buffer;
00297   buffer = xmlGetProp (node, prop);
00298   if (!buffer)
00299     *error_code = 1;
00300   else
00301     {
00302       if (sscanf ((char *) buffer, "%u", &i) != 1)
00303         *error_code = 2;
00304       else
00305         *error_code = 0;
00306       xmlFree (buffer);
00307     }
00308   return i;
00309 }
00310
00323 double
00324 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00325 {
00326   double x = 0.;
00327   xmlChar *buffer;
00328   buffer = xmlGetProp (node, prop);
00329   if (!buffer)
00330     *error_code = 1;
00331   else
00332     {
00333       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00334         *error_code = 2;
00335       else
00336         *error_code = 0;
00337       xmlFree (buffer);
00338     }
00339   return x;
```

```
00340 }
00341
00352 void
00353 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00354 {
00355   xmlChar buffer[64];
00356   snprintf ((char *) buffer, 64, "%d", value);
00357   xmlSetProp (node, prop, buffer);
00358 }
00359
00371 void
00372 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00373 {
00374   xmlChar buffer[64];
00375   snprintf ((char *) buffer, 64, "%u", value);
00376   xmlSetProp (node, prop, buffer);
00377 }
00378
00390 void
00391 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00392 {
00393   xmlChar buffer[64];
00394   snprintf ((char *) buffer, 64, "%.14lg", value);
00395   xmlSetProp (node, prop, buffer);
00396 }
00397
00402 void
00403 input_new ()
00404 {
00405   unsigned int i;
00406 #if DEBUG
00407   fprintf (stderr, "input_init: start\n");
00408 #endif
00409   input->nvariables = input->nexperiments = input->ninputs = 0;
00410   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00411   input->experiment = input->label = NULL;
00412   input->precision = input->nsweeps = input->nbits = NULL;
00413   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00414     = input->weight = NULL;
00415   for (i = 0; i < MAX_NINPUTS; ++i)
00416     input->template[i] = NULL;
00417 #if DEBUG
00418   fprintf (stderr, "input_init: end\n");
00419 #endif
00420 }
00421
00426 void
00427 input_free ()
00428 {
00429   unsigned int i, j;
00430 #if DEBUG
00431   fprintf (stderr, "input_free: start\n");
00432 #endif
00433   g_free (input->name);
00434   g_free (input->directory);
00435   for (i = 0; i < input->nexperiments; ++i)
00436     {
00437       xmlFree (input->experiment[i]);
00438       for (j = 0; j < input->ninputs; ++j)
00439         xmlFree (input->template[j][i]);
00440     }
00441   g_free (input->experiment);
00442   for (i = 0; i < input->ninputs; ++i)
00443     g_free (input->template[i]);
00444   for (i = 0; i < input->nvariables; ++i)
00445     xmlFree (input->label[i]);
00446   g_free (input->label);
00447   g_free (input->precision);
00448   g_free (input->rangemin);
00449   g_free (input->rangemax);
00450   g_free (input->rangeminabs);
00451   g_free (input->rangemaxabs);
00452   g_free (input->weight);
00453   g_free (input->nsweeps);
00454   g_free (input->nbits);
00455   xmlFree (input->evaluator);
00456   xmlFree (input->simulator);
00457   input->nexperiments = input->ninputs = input->nvariables = 0;
00458 #if DEBUG
00459   fprintf (stderr, "input_free: end\n");
00460 #endif
00461 }
00462
00470 int
00471 input_open (char *filename)
```

```
00472 {
00473   char buffer2[64];
00474   xmlDoc *doc;
00475   xmlNode *node, *child;
00476   xmlChar *buffer;
00477   char *msg;
00478   int error_code;
00479   unsigned int i;
00480
00481 #if DEBUG
00482   fprintf (stderr, "input_new: start\n");
00483 #endif
00484
00485   // Resetting input data
00486   input_new ();
00487
00488   // Parsing the input file
00489   doc = xmlParseFile (filename);
00490   if (!doc)
00491     {
00492       msg = gettext ("Unable to parse the input file");
00493       goto exit_on_error;
00494     }
00495
00496   // Getting the root node
00497   node = xmlDocGetRootElement (doc);
00498   if (xmlStrcmp (node->name, XML_CALIBRATE))
00499     {
00500       msg = gettext ("Bad root XML node");
00501       goto exit_on_error;
00502     }
00503
00504   // Opening simulator program name
00505   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00506   if (!input->simulator)
00507     {
00508       msg = gettext ("Bad simulator program");
00509       goto exit_on_error;
00510     }
00511
00512   // Opening evaluator program name
00513   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00514
00515   // Obtaining pseudo-random numbers generator seed
00516   if (!xmlHasProp (node, XML_SEED))
00517     input->seed = DEFAULT_RANDOM_SEED;
00518   else
00519     {
00520       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00521       if (error_code)
00522         {
00523           msg = gettext ("Bad pseudo-random numbers generator seed");
00524           goto exit_on_error;
00525         }
00526     }
00527
00528   // Opening algorithm
00529   buffer = xmlGetProp (node, XML_ALGORITHM);
00530   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00531     {
00532       input->algorithm = ALGORITHM_MONTE_CARLO;
00533
00534       // Obtaining simulations number
00535       input->nsimulations
00536         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00537       if (error_code)
00538         {
00539           msg = gettext ("Bad simulations number");
00540           goto exit_on_error;
00541         }
00542     }
00543   else if (!xmlStrcmp (buffer, XML_SWEEP))
00544     input->algorithm = ALGORITHM_SWEEP;
00545   else if (!xmlStrcmp (buffer, XML_GENETIC))
00546     {
00547       input->algorithm = ALGORITHM_GENETIC;
00548
00549       // Obtaining population
00550       if (xmlHasProp (node, XML_NPOPULATION))
00551         {
00552           input->nsimulations
00553             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00554           if (error_code || input->nsimulations < 3)
00555             {
00556               msg = gettext ("Invalid population number");
00557               goto exit_on_error;
00558             }
```

```
00559         }
00560       else
00561         {
00562           msg = gettext ("No population number");
00563           goto exit_on_error;
00564         }
00565
00566       // Obtaining generations
00567       if (xmlHasProp (node, XML_NGENERATIONS))
00568         {
00569           input->niterations
00570             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00571           if (error_code || !input->niterations)
00572             {
00573               msg = gettext ("Invalid generations number");
00574               goto exit_on_error;
00575             }
00576         }
00577       else
00578         {
00579           msg = gettext ("No generations number");
00580           goto exit_on_error;
00581         }
00582
00583       // Obtaining mutation probability
00584       if (xmlHasProp (node, XML_MUTATION))
00585         {
00586           input->mutation_ratio
00587             = xml_node_get_float (node, XML_MUTATION, &error_code);
00588           if (error_code || input->mutation_ratio < 0.
00589               || input->mutation_ratio >= 1.)
00590             {
00591               msg = gettext ("Invalid mutation probability");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No mutation probability");
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining reproduction probability
00602       if (xmlHasProp (node, XML_REPRODUCTION))
00603         {
00604           input->reproduction_ratio
00605             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00606           if (error_code || input->reproduction_ratio < 0.
00607               || input->reproduction_ratio >= 1.0)
00608             {
00609               msg = gettext ("Invalid reproduction probability");
00610               goto exit_on_error;
00611             }
00612         }
00613       else
00614         {
00615           msg = gettext ("No reproduction probability");
00616           goto exit_on_error;
00617         }
00618
00619       // Obtaining adaptation probability
00620       if (xmlHasProp (node, XML_ADAPTATION))
00621         {
00622           input->adaptation_ratio
00623             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00624           if (error_code || input->adaptation_ratio < 0.
00625               || input->adaptation_ratio >= 1.)
00626             {
00627               msg = gettext ("Invalid adaptation probability");
00628               goto exit_on_error;
00629             }
00630         }
00631       else
00632         {
00633           msg = gettext ("No adaptation probability");
00634           goto exit_on_error;
00635         }
00636
00637       // Checking survivals
00638       i = input->mutation_ratio * input->nsimulations;
00639       i += input->reproduction_ratio * input->nsimulations;
00640       i += input->adaptation_ratio * input->nsimulations;
00641       if (i > input->nsimulations - 2)
00642         {
00643           msg = gettext
00644             ("No enough survival entities to reproduce the population");
00645           goto exit_on_error;
```

```
00646         }
00647     }
00648   else
00649     {
00650       msg = gettext ("Unknown algorithm");
00651       goto exit_on_error;
00652     }
00653
00654   if (input->algorithm == ALGORITHM_MONTE_CARLO
00655       || input->algorithm == ALGORITHM_SWEEP)
00656     {
00657
00658       // Obtaining iterations number
00659       input->niterations
00660         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00661       if (error_code == 1)
00662         input->niterations = 1;
00663       else if (error_code)
00664         {
00665           msg = gettext ("Bad iterations number");
00666           goto exit_on_error;
00667         }
00668
00669       // Obtaining best number
00670       if (xmlHasProp (node, XML_NBEST))
00671         {
00672           input->nbest = xml_node_get_uint (node,
    XML_NBEST, &error_code);
00673           if (error_code || !input->nbest)
00674             {
00675               msg = gettext ("Invalid best number");
00676               goto exit_on_error;
00677             }
00678         }
00679       else
00680         input->nbest = 1;
00681
00682       // Obtaining tolerance
00683       if (xmlHasProp (node, XML_TOLERANCE))
00684         {
00685           input->tolerance
00686             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00687           if (error_code || input->tolerance < 0.)
00688             {
00689               msg = gettext ("Invalid tolerance");
00690               goto exit_on_error;
00691             }
00692         }
00693       else
00694         input->tolerance = 0.;
00695     }
00696
00697   // Reading the experimental data
00698   for (child = node->children; child; child = child->next)
00699     {
00700       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00701         break;
00702 #if DEBUG
00703       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00704 #endif
00705       if (xmlHasProp (child, XML_NAME))
00706         {
00707           input->experiment
00708             = g_realloc (input->experiment,
00709                          (1 + input->nexperiments) * sizeof (char *));
00710           input->experiment[input->nexperiments]
00711             = (char *) xmlGetProp (child, XML_NAME);
00712         }
00713       else
00714         {
00715           msg = gettext ("No experiment file name");
00716           goto exit_on_error;
00717         }
00718 #if DEBUG
00719       fprintf (stderr, "input_new: experiment=%s\n",
00720                input->experiment[input->nexperiments]);
00721 #endif
00722       input->weight = g_realloc (input->weight,
00723                                  (1 + input->nexperiments) * sizeof (double));
00724       if (xmlHasProp (child, XML_WEIGHT))
00725         {
00726           input->weight[input->nexperiments]
00727             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00728           if (error_code)
00729             {
00730               msg = gettext ("Bad weight");
00731               goto exit_on_error;
```

```
00732             }
00733         }
00734       else
00735         input->weight[input->nexperiments] = 1.;
00736 #if DEBUG
00737       fprintf (stderr, "input_new: weight=%lg\n",
00738                input->weight[input->nexperiments]);
00739 #endif
00740       if (!input->nexperiments)
00741         input->ninputs = 0;
00742 #if DEBUG
00743       fprintf (stderr, "input_new: template[0]\n");
00744 #endif
00745       if (xmlHasProp (child, XML_TEMPLATE1))
00746         {
00747           input->template[0]
00748             = (char **) g_realloc (input->template[0],
00749                                    (1 + input->nexperiments) * sizeof (char *));
00750           input->template[0][input->nexperiments]
00751             = (char *) xmlGetProp (child, template[0]);
00752 #if DEBUG
00753           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00754                    input->nexperiments,
00755                    input->template[0][input->nexperiments]);
00756 #endif
00757           if (!input->nexperiments)
00758             ++input->ninputs;
00759 #if DEBUG
00760           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00761 #endif
00762         }
00763       else
00764         {
00765           msg = gettext ("No experiment template");
00766           goto exit_on_error;
00767         }
00768       for (i = 1; i < MAX_NINPUTS; ++i)
00769         {
00770 #if DEBUG
00771           fprintf (stderr, "input_new: template%u\n", i + 1);
00772 #endif
00773           if (xmlHasProp (child, template[i]))
00774             {
00775               if (input->nexperiments && input->ninputs < 2)
00776                 {
00777                   snprintf (buffer2, 64,
00778                             gettext ("Experiment %u: bad templates number"),
00779                             input->nexperiments + 1);
00780                   msg = buffer2;
00781                   goto exit_on_error;
00782                 }
00783               input->template[i] = (char **)
00784                 g_realloc (input->template[i],
00785                            (1 + input->nexperiments) * sizeof (char *));
00786               input->template[i][input->nexperiments]
00787                 = (char *) xmlGetProp (child, template[i]);
00788 #if DEBUG
00789               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00790                        input->nexperiments, i + 1,
00791                        input->template[i][input->nexperiments]);
00792 #endif
00793               if (!input->nexperiments)
00794                 ++input->ninputs;
00795 #if DEBUG
00796               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00797 #endif
00798             }
00799           else if (input->nexperiments && input->ninputs > 1)
00800             {
00801               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00802                         input->nexperiments + 1, i + 1);
00803               msg = buffer2;
00804               goto exit_on_error;
00805             }
00806           else
00807             break;
00808         }
00809       ++input->nexperiments;
00810 #if DEBUG
00811       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00812 #endif
00813     }
00814   if (!input->nexperiments)
00815     {
00816       msg = gettext ("No calibration experiments");
00817       goto exit_on_error;
00818     }
```

```
00819
00820   // Reading the variables data
00821   for (; child; child = child->next)
00822     {
00823       if (xmlStrcmp (child->name, XML_VARIABLE))
00824         {
00825           msg = gettext ("Bad XML node");
00826           goto exit_on_error;
00827         }
00828       if (xmlHasProp (child, XML_NAME))
00829         {
00830           input->label = g_realloc
00831             (input->label, (1 + input->nvariables) * sizeof (char *));
00832           input->label[input->nvariables]
00833             = (char *) xmlGetProp (child, XML_NAME);
00834         }
00835       else
00836         {
00837           msg = gettext ("No variable name");
00838           goto exit_on_error;
00839         }
00840       if (xmlHasProp (child, XML_MINIMUM))
00841         {
00842           input->rangemin = g_realloc
00843             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00844           input->rangeminabs = g_realloc
00845             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00846           input->rangemin[input->nvariables]
00847             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00848           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00849             {
00850               input->rangeminabs[input->nvariables]
00851                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00852             }
00853           else
00854             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00855         }
00856       else
00857         {
00858           msg = gettext ("No minimum range");
00859           goto exit_on_error;
00860         }
00861       if (xmlHasProp (child, XML_MAXIMUM))
00862         {
00863           input->rangemax = g_realloc
00864             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00865           input->rangemaxabs = g_realloc
00866             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00867           input->rangemax[input->nvariables]
00868             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00869           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00870             input->rangemaxabs[input->nvariables]
00871               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00872           else
00873             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00874         }
00875       else
00876         {
00877           msg = gettext ("No maximum range");
00878           goto exit_on_error;
00879         }
00880       input->precision = g_realloc
00881         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00882       if (xmlHasProp (child, XML_PRECISION))
00883         input->precision[input->nvariables]
00884           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00885       else
00886         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00887       if (input->algorithm == ALGORITHM_SWEEP)
00888         {
00889           if (xmlHasProp (child, XML_NSWEEPS))
00890             {
00891               input->nsweeps = (unsigned int *)
00892                 g_realloc (input->nsweeps,
00893                            (1 + input->nvariables) * sizeof (unsigned int));
00894               input->nsweeps[input->nvariables]
00895                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00896             }
00897           else
00898             {
00899               msg = gettext ("No sweeps number");
00900               goto exit_on_error;
00901             }
00902 #if DEBUG
```

```
00903           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00904                    input->nsweeps[input->nvariables], input->
    nsimulations);
00905 #endif
00906         }
00907       if (input->algorithm == ALGORITHM_GENETIC)
00908         {
00909           // Obtaining bits representing each variable
00910           if (xmlHasProp (child, XML_NBITS))
00911             {
00912               input->nbits = (unsigned int *)
00913                 g_realloc (input->nbits,
00914                            (1 + input->nvariables) * sizeof (unsigned int));
00915               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00916               if (error_code || !i)
00917                 {
00918                   msg = gettext ("Invalid bit number");
00919                   goto exit_on_error;
00920                 }
00921               input->nbits[input->nvariables] = i;
00922             }
00923           else
00924             {
00925               msg = gettext ("No bits number");
00926               goto exit_on_error;
00927             }
00928         }
00929       ++input->nvariables;
00930     }
00931   if (!input->nvariables)
00932     {
00933       msg = gettext ("No calibration variables");
00934       goto exit_on_error;
00935     }
00936
00937   // Getting the working directory
00938   input->directory = g_path_get_dirname (filename);
00939   input->name = g_path_get_basename (filename);
00940
00941   // Closing the XML document
00942   xmlFreeDoc (doc);
00943
00944 #if DEBUG
00945   fprintf (stderr, "input_new: end\n");
00946 #endif
00947   return 1;
00948
00949 exit_on_error:
00950   show_error (msg);
00951   input_free ();
00952 #if DEBUG
00953   fprintf (stderr, "input_new: end\n");
00954 #endif
00955   return 0;
00956 }
00957
00969 void
00970 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
00971 {
00972   unsigned int i;
00973   char buffer[32], value[32], *buffer2, *buffer3, *content;
00974   FILE *file;
00975   gsize length;
00976   GRegex *regex;
00977
00978 #if DEBUG
00979   fprintf (stderr, "calibrate_input: start\n");
00980 #endif
00981
00982   // Checking the file
00983   if (!template)
00984     goto calibrate_input_end;
00985
00986   // Opening template
00987   content = g_mapped_file_get_contents (template);
00988   length = g_mapped_file_get_length (template);
00989 #if DEBUG
00990   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
00991            content);
00992 #endif
00993   file = fopen (input, "w");
00994
00995   // Parsing template
00996   for (i = 0; i < calibrate->nvariables; ++i)
00997     {
00998 #if DEBUG
00999       fprintf (stderr, "calibrate_input: variable=%u\n", i);
```

```
01000 #endif
01001       snprintf (buffer, 32, "@variable%u@", i + 1);
01002       regex = g_regex_new (buffer, 0, 0, NULL);
01003       if (i == 0)
01004         {
01005           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01006                                              calibrate->label[i], 0, NULL);
01007 #if DEBUG
01008           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01009 #endif
01010         }
01011       else
01012         {
01013           length = strlen (buffer3);
01014           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01015                                              calibrate->label[i], 0, NULL);
01016           g_free (buffer3);
01017         }
01018       g_regex_unref (regex);
01019       length = strlen (buffer2);
01020       snprintf (buffer, 32, "@value%u@", i + 1);
01021       regex = g_regex_new (buffer, 0, 0, NULL);
01022       snprintf (value, 32, format[calibrate->precision[i]],
01023                 calibrate->value[simulation * calibrate->nvariables + i]);
01024
01025 #if DEBUG
01026       fprintf (stderr, "calibrate_input: value=%s\n", value);
01027 #endif
01028       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01029                                          0, NULL);
01030       g_free (buffer2);
01031       g_regex_unref (regex);
01032     }
01033
01034   // Saving input file
01035   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01036   g_free (buffer3);
01037   fclose (file);
01038
01039 calibrate_input_end:
01040 #if DEBUG
01041   fprintf (stderr, "calibrate_input: end\n");
01042 #endif
01043   return;
01044 }
01045
01056 double
01057 calibrate_parse (unsigned int simulation, unsigned int experiment)
01058 {
01059   unsigned int i;
01060   double e;
01061   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01062     *buffer3, *buffer4;
01063   FILE *file_result;
01064
01065 #if DEBUG
01066   fprintf (stderr, "calibrate_parse: start\n");
01067   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01068            experiment);
01069 #endif
01070
01071   // Opening input files
01072   for (i = 0; i < calibrate->ninputs; ++i)
01073     {
01074       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01075 #if DEBUG
01076       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01077 #endif
01078       calibrate_input (simulation, &input[i][0],
01079                        calibrate->file[i][experiment]);
01080     }
01081   for (; i < MAX_NINPUTS; ++i)
01082     strcpy (&input[i][0], "");
01083 #if DEBUG
01084   fprintf (stderr, "calibrate_parse: parsing end\n");
01085 #endif
01086
01087   // Performing the simulation
01088   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01089   buffer2 = g_path_get_dirname (calibrate->simulator);
01090   buffer3 = g_path_get_basename (calibrate->simulator);
01091   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01092   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01093             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01094             input[6], input[7], output);
01095   g_free (buffer4);
01096   g_free (buffer3);
```

```
01097   g_free (buffer2);
01098 #if DEBUG
01099   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01100 #endif
01101   system (buffer);
01102
01103   // Checking the objective value function
01104   if (calibrate->evaluator)
01105     {
01106       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01107       buffer2 = g_path_get_dirname (calibrate->evaluator);
01108       buffer3 = g_path_get_basename (calibrate->evaluator);
01109       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01110       snprintf (buffer, 512, "\"%s\" %s %s %s",
01111                 buffer4, output, calibrate->experiment[experiment], result);
01112       g_free (buffer4);
01113       g_free (buffer3);
01114       g_free (buffer2);
01115 #if DEBUG
01116       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01117 #endif
01118       system (buffer);
01119       file_result = fopen (result, "r");
01120       e = atof (fgets (buffer, 512, file_result));
01121       fclose (file_result);
01122     }
01123   else
01124     {
01125       strcpy (result, "");
01126       file_result = fopen (output, "r");
01127       e = atof (fgets (buffer, 512, file_result));
01128       fclose (file_result);
01129     }
01130
01131   // Removing files
01132 #if !DEBUG
01133   for (i = 0; i < calibrate->ninputs; ++i)
01134     {
01135       if (calibrate->file[i][0])
01136         {
01137           snprintf (buffer, 512, RM " %s", &input[i][0]);
01138           system (buffer);
01139         }
01140     }
01141   snprintf (buffer, 512, RM " %s %s", output, result);
01142   system (buffer);
01143 #endif
01144
01145 #if DEBUG
01146   fprintf (stderr, "calibrate_parse: end\n");
01147 #endif
01148
01149   // Returning the objective function
01150   return e * calibrate->weight[experiment];
01151 }
01152
01157 void
01158 calibrate_print ()
01159 {
01160   unsigned int i;
01161   char buffer[512];
01162 #if HAVE_MPI
01163   if (!calibrate->mpi_rank)
01164     {
01165 #endif
01166       printf ("THE BEST IS\n");
01167       fprintf (calibrate->file_result, "THE BEST IS\n");
01168       printf ("error=%.15le\n", calibrate->error_old[0]);
01169       fprintf (calibrate->file_result, "error=%.15le\n",
01170                calibrate->error_old[0]);
01171       for (i = 0; i < calibrate->nvariables; ++i)
01172         {
01173           snprintf (buffer, 512, "%s=%s\n",
01174                     calibrate->label[i], format[calibrate->precision[i]]);
01175           printf (buffer, calibrate->value_old[i]);
01176           fprintf (calibrate->file_result, buffer, calibrate->
      value_old[i]);
01177         }
01178       fflush (calibrate->file_result);
01179 #if HAVE_MPI
01180     }
01181 #endif
01182 }
01183
01192 void
01193 calibrate_save_variables (unsigned int simulation, double error)
01194 {
```

```
01195   unsigned int i;
01196   char buffer[64];
01197 #if DEBUG
01198   fprintf (stderr, "calibrate_save_variables: start\n");
01199 #endif
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
01202       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01203       fprintf (calibrate->file_variables, buffer,
01204                calibrate->value[simulation * calibrate->nvariables + i]);
01205     }
01206   fprintf (calibrate->file_variables, "%.14le\n", error);
01207 #if DEBUG
01208   fprintf (stderr, "calibrate_save_variables: end\n");
01209 #endif
01210 }
01211
01220 void
01221 calibrate_best_thread (unsigned int simulation, double value)
01222 {
01223   unsigned int i, j;
01224   double e;
01225 #if DEBUG
01226   fprintf (stderr, "calibrate_best_thread: start\n");
01227 #endif
01228   if (calibrate->nsaveds < calibrate->nbest
01229       || value < calibrate->error_best[calibrate->nsaveds - 1])
01230     {
01231       g_mutex_lock (mutex);
01232       if (calibrate->nsaveds < calibrate->nbest)
01233         ++calibrate->nsaveds;
01234       calibrate->error_best[calibrate->nsaveds - 1] = value;
01235       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01236       for (i = calibrate->nsaveds; --i;)
01237         {
01238           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01239             {
01240               j = calibrate->simulation_best[i];
01241               e = calibrate->error_best[i];
01242               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01243               calibrate->error_best[i] = calibrate->error_best[i - 1];
01244               calibrate->simulation_best[i - 1] = j;
01245               calibrate->error_best[i - 1] = e;
01246             }
01247           else
01248             break;
01249         }
01250       g_mutex_unlock (mutex);
01251     }
01252 #if DEBUG
01253   fprintf (stderr, "calibrate_best_thread: end\n");
01254 #endif
01255 }
01256
01265 void
01266 calibrate_best_sequential (unsigned int simulation, double value)
01267 {
01268   unsigned int i, j;
01269   double e;
01270 #if DEBUG
01271   fprintf (stderr, "calibrate_best_sequential: start\n");
01272 #endif
01273   if (calibrate->nsaveds < calibrate->nbest
01274       || value < calibrate->error_best[calibrate->nsaveds - 1])
01275     {
01276       if (calibrate->nsaveds < calibrate->nbest)
01277         ++calibrate->nsaveds;
01278       calibrate->error_best[calibrate->nsaveds - 1] = value;
01279       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01280       for (i = calibrate->nsaveds; --i;)
01281         {
01282           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01283             {
01284               j = calibrate->simulation_best[i];
01285               e = calibrate->error_best[i];
01286               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01287               calibrate->error_best[i] = calibrate->error_best[i - 1];
01288               calibrate->simulation_best[i - 1] = j;
01289               calibrate->error_best[i - 1] = e;
01290             }
01291           else
01292             break;
01293         }
01294     }
01295 #if DEBUG
```

```
01296   fprintf (stderr, "calibrate_best_sequential: end\n");
01297 #endif
01298 }
01299
01307 void *
01308 calibrate_thread (ParallelData * data)
01309 {
01310   unsigned int i, j, thread;
01311   double e;
01312 #if DEBUG
01313   fprintf (stderr, "calibrate_thread: start\n");
01314 #endif
01315   thread = data->thread;
01316 #if DEBUG
01317   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01318            calibrate->thread[thread], calibrate->thread[thread + 1]);
01319 #endif
01320   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01321     {
01322       e = 0.;
01323       for (j = 0; j < calibrate->nexperiments; ++j)
01324         e += calibrate_parse (i, j);
01325       calibrate_best_thread (i, e);
01326       g_mutex_lock (mutex);
01327       calibrate_save_variables (i, e);
01328       g_mutex_unlock (mutex);
01329 #if DEBUG
01330       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01331 #endif
01332     }
01333 #if DEBUG
01334   fprintf (stderr, "calibrate_thread: end\n");
01335 #endif
01336   g_thread_exit (NULL);
01337   return NULL;
01338 }
01339
01344 void
01345 calibrate_sequential ()
01346 {
01347   unsigned int i, j;
01348   double e;
01349 #if DEBUG
01350   fprintf (stderr, "calibrate_sequential: start\n");
01351   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01352            calibrate->nstart, calibrate->nend);
01353 #endif
01354   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01355     {
01356       e = 0.;
01357       for (j = 0; j < calibrate->nexperiments; ++j)
01358         e += calibrate_parse (i, j);
01359       calibrate_best_sequential (i, e);
01360       calibrate_save_variables (i, e);
01361 #if DEBUG
01362       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01363 #endif
01364     }
01365 #if DEBUG
01366   fprintf (stderr, "calibrate_sequential: end\n");
01367 #endif
01368 }
01369
01381 void
01382 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01383                  double *error_best)
01384 {
01385   unsigned int i, j, k, s[calibrate->nbest];
01386   double e[calibrate->nbest];
01387 #if DEBUG
01388   fprintf (stderr, "calibrate_merge: start\n");
01389 #endif
01390   i = j = k = 0;
01391   do
01392     {
01393       if (i == calibrate->nsaveds)
01394         {
01395           s[k] = simulation_best[j];
01396           e[k] = error_best[j];
01397           ++j;
01398           ++k;
01399           if (j == nsaveds)
01400             break;
01401         }
01402       else if (j == nsaveds)
01403         {
01404           s[k] = calibrate->simulation_best[i];
```

```
01405           e[k] = calibrate->error_best[i];
01406           ++i;
01407           ++k;
01408           if (i == calibrate->nsaveds)
01409             break;
01410         }
01411       else if (calibrate->error_best[i] > error_best[j])
01412         {
01413           s[k] = simulation_best[j];
01414           e[k] = error_best[j];
01415           ++j;
01416           ++k;
01417         }
01418       else
01419         {
01420           s[k] = calibrate->simulation_best[i];
01421           e[k] = calibrate->error_best[i];
01422           ++i;
01423           ++k;
01424         }
01425     }
01426   while (k < calibrate->nbest);
01427   calibrate->nsaveds = k;
01428   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01429   memcpy (calibrate->error_best, e, k * sizeof (double));
01430 #if DEBUG
01431   fprintf (stderr, "calibrate_merge: end\n");
01432 #endif
01433 }
01434
01439 #if HAVE_MPI
01440 void
01441 calibrate_synchronise ()
01442 {
01443   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01444   double error_best[calibrate->nbest];
01445   MPI_Status mpi_stat;
01446 #if DEBUG
01447   fprintf (stderr, "calibrate_synchronise: start\n");
01448 #endif
01449   if (calibrate->mpi_rank == 0)
01450     {
01451       for (i = 1; i < ntasks; ++i)
01452         {
01453           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01454           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01455                     MPI_COMM_WORLD, &mpi_stat);
01456           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01457                     MPI_COMM_WORLD, &mpi_stat);
01458           calibrate_merge (nsaveds, simulation_best, error_best);
01459         }
01460     }
01461   else
01462     {
01463       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01464       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01465                 MPI_COMM_WORLD);
01466       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01467                 MPI_COMM_WORLD);
01468     }
01469 #if DEBUG
01470   fprintf (stderr, "calibrate_synchronise: end\n");
01471 #endif
01472 }
01473 #endif
01474
01479 void
01480 calibrate_sweep ()
01481 {
01482   unsigned int i, j, k, l;
01483   double e;
01484   GThread *thread[nthreads];
01485   ParallelData data[nthreads];
01486 #if DEBUG
01487   fprintf (stderr, "calibrate_sweep: start\n");
01488 #endif
01489   for (i = 0; i < calibrate->nsimulations; ++i)
01490     {
01491       k = i;
01492       for (j = 0; j < calibrate->nvariables; ++j)
01493         {
01494           l = k % calibrate->nsweeps[j];
01495           k /= calibrate->nsweeps[j];
01496           e = calibrate->rangemin[j];
01497           if (calibrate->nsweeps[j] > 1)
01498             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01499               / (calibrate->nsweeps[j] - 1);
```

```
01500           calibrate->value[i * calibrate->nvariables + j] = e;
01501         }
01502     }
01503   calibrate->nsaveds = 0;
01504   if (nthreads <= 1)
01505     calibrate_sequential ();
01506   else
01507     {
01508       for (i = 0; i < nthreads; ++i)
01509         {
01510           data[i].thread = i;
01511           thread[i]
01512             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01513         }
01514       for (i = 0; i < nthreads; ++i)
01515         g_thread_join (thread[i]);
01516     }
01517 #if HAVE_MPI
01518   // Communicating tasks results
01519   calibrate_synchronise ();
01520 #endif
01521 #if DEBUG
01522   fprintf (stderr, "calibrate_sweep: end\n");
01523 #endif
01524 }
01525
01530 void
01531 calibrate_MonteCarlo ()
01532 {
01533   unsigned int i, j;
01534   GThread *thread[nthreads];
01535   ParallelData data[nthreads];
01536 #if DEBUG
01537   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01538 #endif
01539   for (i = 0; i < calibrate->nsimulations; ++i)
01540     for (j = 0; j < calibrate->nvariables; ++j)
01541       calibrate->value[i * calibrate->nvariables + j]
01542         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01543         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01544   calibrate->nsaveds = 0;
01545   if (nthreads <= 1)
01546     calibrate_sequential ();
01547   else
01548     {
01549       for (i = 0; i < nthreads; ++i)
01550         {
01551           data[i].thread = i;
01552           thread[i]
01553             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01554         }
01555       for (i = 0; i < nthreads; ++i)
01556         g_thread_join (thread[i]);
01557     }
01558 #if HAVE_MPI
01559   // Communicating tasks results
01560   calibrate_synchronise ();
01561 #endif
01562 #if DEBUG
01563   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01564 #endif
01565 }
01566
01574 double
01575 calibrate_genetic_objective (Entity * entity)
01576 {
01577   unsigned int j;
01578   double objective;
01579   char buffer[64];
01580 #if DEBUG
01581   fprintf (stderr, "calibrate_genetic_objective: start\n");
01582 #endif
01583   for (j = 0; j < calibrate->nvariables; ++j)
01584     {
01585       calibrate->value[entity->id * calibrate->nvariables + j]
01586         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01587     }
01588   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01589     objective += calibrate_parse (entity->id, j);
01590   g_mutex_lock (mutex);
01591   for (j = 0; j < calibrate->nvariables; ++j)
01592     {
01593       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01594       fprintf (calibrate->file_variables, buffer,
01595                genetic_get_variable (entity, calibrate->genetic_variable + j));
01596     }
01597   fprintf (calibrate->file_variables, "%.14le\n", objective);
```

```
01598   g_mutex_unlock (mutex);
01599 #if DEBUG
01600   fprintf (stderr, "calibrate_genetic_objective: end\n");
01601 #endif
01602   return objective;
01603 }
01604
01609 void
01610 calibrate_genetic ()
01611 {
01612   char *best_genome;
01613   double best_objective, *best_variable;
01614 #if DEBUG
01615   fprintf (stderr, "calibrate_genetic: start\n");
01616   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
01617            nthreads);
01618   fprintf (stderr,
01619            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
01620            calibrate->nvariables, calibrate->nsimulations,
01621            calibrate->niterations);
01622   fprintf (stderr,
01623            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01624            calibrate->mutation_ratio, calibrate->
    reproduction_ratio,
01625            calibrate->adaptation_ratio);
01626 #endif
01627   genetic_algorithm_default (calibrate->nvariables,
01628                              calibrate->genetic_variable,
01629                              calibrate->nsimulations,
01630                              calibrate->niterations,
01631                              calibrate->mutation_ratio,
01632                              calibrate->reproduction_ratio,
01633                              calibrate->adaptation_ratio,
01634                              &calibrate_genetic_objective,
01635                              &best_genome, &best_variable, &best_objective);
01636 #if DEBUG
01637   fprintf (stderr, "calibrate_genetic: the best\n");
01638 #endif
01639   calibrate->error_old = (double *) g_malloc (sizeof (double));
01640   calibrate->value_old
01641     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
01642   calibrate->error_old[0] = best_objective;
01643   memcpy (calibrate->value_old, best_variable,
01644           calibrate->nvariables * sizeof (double));
01645   g_free (best_genome);
01646   g_free (best_variable);
01647   calibrate_print ();
01648 #if DEBUG
01649   fprintf (stderr, "calibrate_genetic: end\n");
01650 #endif
01651 }
01652
01657 void
01658 calibrate_save_old ()
01659 {
01660   unsigned int i, j;
01661 #if DEBUG
01662   fprintf (stderr, "calibrate_save_old: start\n");
01663 #endif
01664   memcpy (calibrate->error_old, calibrate->error_best,
01665           calibrate->nbest * sizeof (double));
01666   for (i = 0; i < calibrate->nbest; ++i)
01667     {
01668       j = calibrate->simulation_best[i];
01669       memcpy (calibrate->value_old + i * calibrate->nvariables,
01670               calibrate->value + j * calibrate->nvariables,
01671               calibrate->nvariables * sizeof (double));
01672     }
01673 #if DEBUG
01674   for (i = 0; i < calibrate->nvariables; ++i)
01675     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
01676              i, calibrate->value_old[i]);
01677   fprintf (stderr, "calibrate_save_old: end\n");
01678 #endif
01679 }
01680
01686 void
01687 calibrate_merge_old ()
01688 {
01689   unsigned int i, j, k;
01690   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
    nbest],
01691     *enew, *eold;
01692 #if DEBUG
01693   fprintf (stderr, "calibrate_merge_old: start\n");
01694 #endif
01695   enew = calibrate->error_best;
```

```
01696   eold = calibrate->error_old;
01697   i = j = k = 0;
01698   do
01699     {
01700       if (*enew < *eold)
01701         {
01702           memcpy (v + k * calibrate->nvariables,
01703                   calibrate->value
01704                   + calibrate->simulation_best[i] * calibrate->
       nvariables,
01705                   calibrate->nvariables * sizeof (double));
01706           e[k] = *enew;
01707           ++k;
01708           ++enew;
01709           ++i;
01710         }
01711       else
01712         {
01713           memcpy (v + k * calibrate->nvariables,
01714                   calibrate->value_old + j * calibrate->nvariables,
01715                   calibrate->nvariables * sizeof (double));
01716           e[k] = *eold;
01717           ++k;
01718           ++eold;
01719           ++j;
01720         }
01721     }
01722   while (k < calibrate->nbest);
01723   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
01724   memcpy (calibrate->error_old, e, k * sizeof (double));
01725 #if DEBUG
01726   fprintf (stderr, "calibrate_merge_old: end\n");
01727 #endif
01728 }
01729
01735 void
01736 calibrate_refine ()
01737 {
01738   unsigned int i, j;
01739   double d;
01740 #if HAVE_MPI
01741   MPI_Status mpi_stat;
01742 #endif
01743 #if DEBUG
01744   fprintf (stderr, "calibrate_refine: start\n");
01745 #endif
01746 #if HAVE_MPI
01747   if (!calibrate->mpi_rank)
01748     {
01749 #endif
01750       for (j = 0; j < calibrate->nvariables; ++j)
01751         {
01752           calibrate->rangemin[j] = calibrate->rangemax[j]
01753             = calibrate->value_old[j];
01754         }
01755       for (i = 0; ++i < calibrate->nbest;)
01756         {
01757           for (j = 0; j < calibrate->nvariables; ++j)
01758             {
01759               calibrate->rangemin[j]
01760                 = fmin (calibrate->rangemin[j],
01761                         calibrate->value_old[i * calibrate->nvariables + j]);
01762               calibrate->rangemax[j]
01763                 = fmax (calibrate->rangemax[j],
01764                         calibrate->value_old[i * calibrate->nvariables + j]);
01765             }
01766         }
01767       for (j = 0; j < calibrate->nvariables; ++j)
01768         {
01769           d = 0.5 * calibrate->tolerance
01770             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01771           calibrate->rangemin[j] -= d;
01772           calibrate->rangemin[j]
01773             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
01774           calibrate->rangemax[j] += d;
01775           calibrate->rangemax[j]
01776             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
01777           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
01778                   calibrate->rangemin[j], calibrate->rangemax[j]);
01779           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
01780                    calibrate->label[j], calibrate->rangemin[j],
01781                    calibrate->rangemax[j]);
01782         }
01783 #if HAVE_MPI
01784       for (i = 1; i < ntasks; ++i)
01785         {
01786           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
```

```
01787                     1, MPI_COMM_WORLD);
01788           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
01789                     1, MPI_COMM_WORLD);
01790         }
01791     }
01792   else
01793     {
01794       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01795                 MPI_COMM_WORLD, &mpi_stat);
01796       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01797                 MPI_COMM_WORLD, &mpi_stat);
01798     }
01799 #endif
01800 #if DEBUG
01801   fprintf (stderr, "calibrate_refine: end\n");
01802 #endif
01803 }
01804
01809 void
01810 calibrate_iterate ()
01811 {
01812   unsigned int i;
01813 #if DEBUG
01814   fprintf (stderr, "calibrate_iterate: start\n");
01815 #endif
01816   calibrate->error_old
01817     = (double *) g_malloc (calibrate->nbest * sizeof (double));
01818   calibrate->value_old = (double *)
01819     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
01820   calibrate_step ();
01821   calibrate_save_old ();
01822   calibrate_refine ();
01823   calibrate_print ();
01824   for (i = 1; i < calibrate->niterations; ++i)
01825     {
01826       calibrate_step ();
01827       calibrate_merge_old ();
01828       calibrate_refine ();
01829       calibrate_print ();
01830     }
01831 #if DEBUG
01832   fprintf (stderr, "calibrate_iterate: end\n");
01833 #endif
01834 }
01835
01840 void
01841 calibrate_free ()
01842 {
01843   unsigned int i, j;
01844 #if DEBUG
01845   fprintf (stderr, "calibrate_free: start\n");
01846 #endif
01847   for (i = 0; i < calibrate->nexperiments; ++i)
01848     {
01849       for (j = 0; j < calibrate->ninputs; ++j)
01850         g_mapped_file_unref (calibrate->file[j][i]);
01851     }
01852   for (i = 0; i < calibrate->ninputs; ++i)
01853     g_free (calibrate->file[i]);
01854   g_free (calibrate->error_old);
01855   g_free (calibrate->value_old);
01856   g_free (calibrate->value);
01857   g_free (calibrate->genetic_variable);
01858 #if DEBUG
01859   fprintf (stderr, "calibrate_free: end\n");
01860 #endif
01861 }
01862
01867 void
01868 calibrate_new ()
01869 {
01870   unsigned int i, j, *nbits;
01871
01872 #if DEBUG
01873   fprintf (stderr, "calibrate_new: start\n");
01874 #endif
01875
01876   // Initing pseudo-random numbers generator
01877   gsl_rng_set (calibrate->rng, calibrate->seed);
01878
01879   // Replacing the working dir
01880   chdir (input->directory);
01881
01882   // Obtaining the simulator file
01883   calibrate->simulator = input->simulator;
01884
01885   // Obtaining the evaluator file
```

```
01886   calibrate->evaluator = input->evaluator;
01887
01888   // Obtaining the pseudo-random numbers generator seed
01889   calibrate->seed = input->seed;
01890
01891   // Reading the algorithm
01892   calibrate->algorithm = input->algorithm;
01893   switch (calibrate->algorithm)
01894     {
01895     case ALGORITHM_MONTE_CARLO:
01896       calibrate_step = calibrate_MonteCarlo;
01897       break;
01898     case ALGORITHM_SWEEP:
01899       calibrate_step = calibrate_sweep;
01900       break;
01901     default:
01902       calibrate_step = calibrate_genetic;
01903       calibrate->mutation_ratio = input->mutation_ratio;
01904       calibrate->reproduction_ratio = input->
      reproduction_ratio;
01905       calibrate->adaptation_ratio = input->adaptation_ratio;
01906     }
01907   calibrate->nsimulations = input->nsimulations;
01908   calibrate->niterations = input->niterations;
01909   calibrate->nbest = input->nbest;
01910   calibrate->tolerance = input->tolerance;
01911
01912   calibrate->simulation_best
01913     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
01914   calibrate->error_best
01915     = (double *) alloca (calibrate->nbest * sizeof (double));
01916
01917   // Reading the experimental data
01918 #if DEBUG
01919   fprintf (stderr, "calibrate_new: current directory=%s\n",
01920            g_get_current_dir ());
01921 #endif
01922   calibrate->nexperiments = input->nexperiments;
01923   calibrate->ninputs = input->ninputs;
01924   calibrate->experiment = input->experiment;
01925   calibrate->weight = input->weight;
01926   for (i = 0; i < input->ninputs; ++i)
01927     {
01928       calibrate->template[i] = input->template[i];
01929       calibrate->file[i]
01930         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
01931     }
01932   for (i = 0; i < input->nexperiments; ++i)
01933     {
01934 #if DEBUG
01935       fprintf (stderr, "calibrate_new: i=%u\n", i);
01936       fprintf (stderr, "calibrate_new: experiment=%s\n",
01937                calibrate->experiment[i]);
01938       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
01939 #endif
01940       for (j = 0; j < input->ninputs; ++j)
01941         {
01942 #if DEBUG
01943           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
01944           fprintf (stderr, "calibrate_new: experiment=%u template%u=%s\n",
01945                    i, j + 1, calibrate->template[j][i]);
01946 #endif
01947           calibrate->file[j][i]
01948             = g_mapped_file_new (input->template[j][i], 0, NULL);
01949         }
01950     }
01951
01952   // Reading the variables data
01953 #if DEBUG
01954   fprintf (stderr, "calibrate_new: reading variables\n");
01955 #endif
01956   calibrate->nvariables = input->nvariables;
01957   calibrate->label = input->label;
01958   calibrate->rangemin = input->rangemin;
01959   calibrate->rangeminabs = input->rangeminabs;
01960   calibrate->rangemax = input->rangemax;
01961   calibrate->rangemaxabs = input->rangemaxabs;
01962   calibrate->precision = input->precision;
01963   calibrate->nsweeps = input->nsweeps;
01964   nbits = input->nbits;
01965   if (input->algorithm == ALGORITHM_SWEEP)
01966     calibrate->nsimulations = 1;
01967   else if (input->algorithm == ALGORITHM_GENETIC)
01968     for (i = 0; i < input->nvariables; ++i)
01969       {
01970         if (calibrate->algorithm == ALGORITHM_SWEEP)
01971           {
```

```
01972             calibrate->nsimulations *= input->nsweeps[i];
01973 #if DEBUG
01974             fprintf (stderr, "calibrate_new: nsweeps=%u nsimulations=%u\n",
01975                      calibrate->nsweeps[i], calibrate->nsimulations);
01976 #endif
01977           }
01978       }
01979
01980   // Allocating values
01981 #if DEBUG
01982   fprintf (stderr, "calibrate_new: allocating variables\n");
01983   fprintf (stderr, "calibrate_new: nvariables=%u\n", calibrate->nvariables);
01984 #endif
01985   calibrate->genetic_variable = NULL;
01986   if (calibrate->algorithm == ALGORITHM_GENETIC)
01987     {
01988       calibrate->genetic_variable = (GeneticVariable *)
01989         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
01990       for (i = 0; i < calibrate->nvariables; ++i)
01991         {
01992 #if DEBUG
01993           fprintf (stderr, "calibrate_new: i=%u min=%lg max=%lg nbits=%u\n",
01994                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
01995 #endif
01996           calibrate->genetic_variable[i].minimum = calibrate->
      rangemin[i];
01997           calibrate->genetic_variable[i].maximum = calibrate->
      rangemax[i];
01998           calibrate->genetic_variable[i].nbits = nbits[i];
01999         }
02000     }
02001 #if DEBUG
02002   fprintf (stderr, "calibrate_new: nvariables=%u nsimulations=%u\n",
02003            calibrate->nvariables, calibrate->nsimulations);
02004 #endif
02005   calibrate->value = (double *) g_malloc (calibrate->nsimulations *
02006                                           calibrate->nvariables *
02007                                           sizeof (double));
02008
02009   // Calculating simulations to perform on each task
02010 #if HAVE_MPI
02011 #if DEBUG
02012   fprintf (stderr, "calibrate_new: rank=%u ntasks=%u\n",
02013            calibrate->mpi_rank, ntasks);
02014 #endif
02015   calibrate->nstart = calibrate->mpi_rank * calibrate->
      nsimulations / ntasks;
02016   calibrate->nend = (1 + calibrate->mpi_rank) * calibrate->
      nsimulations
02017     / ntasks;
02018 #else
02019   calibrate->nstart = 0;
02020   calibrate->nend = calibrate->nsimulations;
02021 #endif
02022 #if DEBUG
02023   fprintf (stderr, "calibrate_new: nstart=%u nend=%u\n", calibrate->nstart,
02024            calibrate->nend);
02025 #endif
02026
02027   // Calculating simulations to perform on each thread
02028   calibrate->thread
02029     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02030   for (i = 0; i <= nthreads; ++i)
02031     {
02032       calibrate->thread[i] = calibrate->nstart
02033         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02034 #if DEBUG
02035       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02036                calibrate->thread[i]);
02037 #endif
02038     }
02039
02040   // Opening result files
02041   calibrate->file_result = fopen ("result", "w");
02042   calibrate->file_variables = fopen ("variables", "w");
02043
02044   // Performing the algorithm
02045   switch (calibrate->algorithm)
02046     {
02047       // Genetic algorithm
02048     case ALGORITHM_GENETIC:
02049       calibrate_genetic ();
02050       break;
02051
02052       // Iterative algorithm
02053     default:
02054       calibrate_iterate ();
```

```
02055     }
02056
02057   // Closing result files
02058   fclose (calibrate->file_variables);
02059   fclose (calibrate->file_result);
02060
02061 #if DEBUG
02062   fprintf (stderr, "calibrate_new: end\n");
02063 #endif
02064 }
02065
02066 #if HAVE_GTK
02067
02074 void
02075 input_save (char *filename)
02076 {
02077   unsigned int i, j;
02078   char *buffer;
02079   xmlDoc *doc;
02080   xmlNode *node, *child;
02081   GFile *file, *file2;
02082
02083   // Getting the input file directory
02084   input->name = g_path_get_basename (filename);
02085   input->directory = g_path_get_dirname (filename);
02086   file = g_file_new_for_path (input->directory);
02087
02088   // Opening the input file
02089   doc = xmlNewDoc ((const xmlChar *) "1.0");
02090
02091   // Setting root XML node
02092   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02093   xmlDocSetRootElement (doc, node);
02094
02095   // Adding properties to the root XML node
02096   file2 = g_file_new_for_path (input->simulator);
02097   buffer = g_file_get_relative_path (file, file2);
02098   g_object_unref (file2);
02099   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02100   g_free (buffer);
02101   if (input->evaluator)
02102     {
02103       file2 = g_file_new_for_path (input->evaluator);
02104       buffer = g_file_get_relative_path (file, file2);
02105       g_object_unref (file2);
02106       if (xmlStrlen ((xmlChar *) buffer))
02107         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02108       g_free (buffer);
02109     }
02110   if (input->seed != DEFAULT_RANDOM_SEED)
02111     xml_node_set_uint (node, XML_SEED, input->seed);
02112
02113   // Setting the algorithm
02114   buffer = (char *) g_malloc (64);
02115   switch (input->algorithm)
02116     {
02117     case ALGORITHM_MONTE_CARLO:
02118       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02119       snprintf (buffer, 64, "%u", input->nsimulations);
02120       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02121       snprintf (buffer, 64, "%u", input->niterations);
02122       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02123       snprintf (buffer, 64, "%.3lg", input->tolerance);
02124       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%u", input->nbest);
02126       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02127       break;
02128     case ALGORITHM_SWEEP:
02129       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02130       snprintf (buffer, 64, "%u", input->niterations);
02131       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02132       snprintf (buffer, 64, "%.3lg", input->tolerance);
02133       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->nbest);
02135       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02136       break;
02137     default:
02138       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02139       snprintf (buffer, 64, "%u", input->nsimulations);
02140       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02141       snprintf (buffer, 64, "%u", input->niterations);
02142       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02143       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02144       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02145       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02146       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02147       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
```

```
02148       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02149       break;
02150     }
02151   g_free (buffer);
02152
02153   // Setting the experimental data
02154   for (i = 0; i < input->nexperiments; ++i)
02155     {
02156       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02157       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02158       if (input->weight[i] != 1.)
02159         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02160       for (j = 0; j < input->ninputs; ++j)
02161         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02162     }
02163
02164   // Setting the variables data
02165   for (i = 0; i < input->nvariables; ++i)
02166     {
02167       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02168       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02169       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02170       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02171         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02172       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02173       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02174         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02175       if (input->precision[i] != DEFAULT_PRECISION)
02176         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02177       if (input->algorithm == ALGORITHM_SWEEP)
02178         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02179       else if (input->algorithm == ALGORITHM_GENETIC)
02180         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02181     }
02182
02183   // Saving the XML file
02184   xmlSaveFormatFile (filename, doc, 1);
02185
02186   // Freeing memory
02187   xmlFreeDoc (doc);
02188 }
02189
02194 void
02195 options_new ()
02196 {
02197   options->label_processors
02198     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02199   options->spin_processors
02200     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02201   gtk_spin_button_set_value (options->spin_processors, (gdouble)
      nthreads);
02202   options->label_seed = (GtkLabel *)
02203     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02204   options->spin_seed = (GtkSpinButton *)
02205     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02206   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02207   options->grid = (GtkGrid *) gtk_grid_new ();
02208   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02209                    0, 0, 1, 1);
02210   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02211                    1, 0, 1, 1);
02212   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 1, 1, 1);
02213   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 1, 1, 1);
02214   gtk_widget_show_all (GTK_WIDGET (options->grid));
02215   options->dialog = (GtkDialog *)
02216     gtk_dialog_new_with_buttons (gettext ("Options"),
02217                                  window->window,
02218                                  GTK_DIALOG_MODAL,
02219                                  gettext ("_OK"), GTK_RESPONSE_OK,
02220                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02221                                  NULL);
02222   gtk_container_add
02223     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02224      GTK_WIDGET (options->grid));
02225   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02226     {
02227       nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02228       input->seed
02229         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
```

```
02230     }
02231   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02232 }
02233
02238 void
02239 running_new ()
02240 {
02241 #if DEBUG
02242   fprintf (stderr, "running_new: start\n");
02243 #endif
02244   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02245   running->dialog = (GtkDialog *)
02246     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02247                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02248   gtk_container_add
02249     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02250      GTK_WIDGET (running->label));
02251   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02252 #if DEBUG
02253   fprintf (stderr, "running_new: end\n");
02254 #endif
02255 }
02256
02262 int
02263 window_save ()
02264 {
02265   char *buffer;
02266   GtkFileChooserDialog *dlg;
02267
02268 #if DEBUG
02269   fprintf (stderr, "window_save: start\n");
02270 #endif
02271
02272   // Opening the saving dialog
02273   dlg = (GtkFileChooserDialog *)
02274     gtk_file_chooser_dialog_new (gettext ("Save file"),
02275                                  window->window,
02276                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02277                                  gettext ("_Cancel"),
02278                                  GTK_RESPONSE_CANCEL,
02279                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02280   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02281
02282   // If OK response then saving
02283   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02284     {
02285
02286       // Adding properties to the root XML node
02287       input->simulator = gtk_file_chooser_get_filename
02288         (GTK_FILE_CHOOSER (window->button_simulator));
02289       if (gtk_toggle_button_get_active
02290           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02291         input->evaluator = gtk_file_chooser_get_filename
02292           (GTK_FILE_CHOOSER (window->button_evaluator));
02293       else
02294         input->evaluator = NULL;
02295
02296       // Setting the algorithm
02297       switch (window_get_algorithm ())
02298         {
02299         case ALGORITHM_MONTE_CARLO:
02300           input->algorithm = ALGORITHM_MONTE_CARLO;
02301           input->nsimulations
02302             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02303           input->niterations
02304             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02305           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02306           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02307           break;
02308         case ALGORITHM_SWEEP:
02309           input->algorithm = ALGORITHM_SWEEP;
02310           input->niterations
02311             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02312           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02313           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02314           break;
02315         default:
02316           input->algorithm = ALGORITHM_GENETIC;
02317           input->nsimulations
02318             = gtk_spin_button_get_value_as_int (window->spin_population);
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_generations);
02321           input->mutation_ratio
```

```
02322             = gtk_spin_button_get_value (window->spin_mutation);
02323           input->reproduction_ratio
02324             = gtk_spin_button_get_value (window->spin_reproduction);
02325           input->adaptation_ratio
02326             = gtk_spin_button_get_value (window->spin_adaptation);
02327           break;
02328         }
02329
02330       // Saving the XML file
02331       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02332       input_save (buffer);
02333
02334       // Closing and freeing memory
02335       g_free (buffer);
02336       gtk_widget_destroy (GTK_WIDGET (dlg));
02337 #if DEBUG
02338       fprintf (stderr, "window_save: end\n");
02339 #endif
02340       return 1;
02341     }
02342
02343   // Closing and freeing memory
02344   gtk_widget_destroy (GTK_WIDGET (dlg));
02345 #if DEBUG
02346   fprintf (stderr, "window_save: end\n");
02347 #endif
02348   return 0;
02349 }
02350
02355 void
02356 window_run ()
02357 {
02358   unsigned int i;
02359   char *msg, *msg2, buffer[64], buffer2[64];
02360 #if DEBUG
02361   fprintf (stderr, "window_run: start\n");
02362 #endif
02363   if (!window_save ())
02364     {
02365 #if DEBUG
02366       fprintf (stderr, "window_run: end\n");
02367 #endif
02368       return;
02369     }
02370   running_new ();
02371   while (gtk_events_pending ())
02372     gtk_main_iteration ();
02373   calibrate_new ();
02374   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02375   snprintf (buffer, 64, "error=%.15le\n", calibrate->error_old[0]);
02376   msg2 = g_strdup (buffer);
02377   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02378     {
02379       snprintf (buffer, 64, "%s=%s\n",
02380                 calibrate->label[i], format[calibrate->precision[i]]);
02381       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02382       msg = g_strconcat (msg2, buffer2, NULL);
02383       g_free (msg2);
02384     }
02385   show_message (gettext ("Best result"), msg2, INFO_TYPE);
02386   g_free (msg2);
02387   calibrate_free ();
02388 #if DEBUG
02389   fprintf (stderr, "window_run: end\n");
02390 #endif
02391 }
02392
02397 void
02398 window_help ()
02399 {
02400   char *buffer, *buffer2;
02401   buffer2 = g_build_filename (window->application_directory, "manuals",
02402                               gettext ("user-manual.pdf"), NULL);
02403   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02404   g_free (buffer2);
02405   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02406   g_free (buffer);
02407 }
02408
02413 void
02414 window_about ()
02415 {
02416   gchar *authors[] = {
02417     "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02418     "Borja Latorre Garcés (borja.latorre@csic.es)",
02419     NULL
02420   };
```

```
02421   gtk_show_about_dialog (window->window,
02422                          "program_name",
02423                          "Calibrator",
02424                          "comments",
02425                          gettext ("A software to make calibrations of "
02426                                   "empirical parameters"),
02427                          "authors", authors,
02428                          "translator-credits",
02429                          "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02430                          "version", "1.1.28", "copyright",
02431                          "Copyright 2012-2015 Javier Burguete Tolosa",
02432                          "logo", window->logo,
02433                          "website-label", gettext ("Website"),
02434                          "website",
02435                          "https://github.com/jburguete/calibrator", NULL);
02436 }
02437
02443 int
02444 window_get_algorithm ()
02445 {
02446   unsigned int i;
02447   for (i = 0; i < NALGORITHMS; ++i)
02448     if (gtk_toggle_button_get_active
02449         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02450       break;
02451   return i;
02452 }
02453
02458 void
02459 window_update ()
02460 {
02461   unsigned int i;
02462   gtk_widget_set_sensitive
02463     (GTK_WIDGET (window->button_evaluator),
02464      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02465                                    (window->check_evaluator)));
02466   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02467   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02468   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02469   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02470   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02471   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
02472   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02473   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02474   gtk_widget_hide (GTK_WIDGET (window->label_population));
02475   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02476   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02477   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02478   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02479   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
02480   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02481   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
02482   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
02483   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
02484   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
02485   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
02486   gtk_widget_hide (GTK_WIDGET (window->label_bits));
02487   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
02488   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
02489   switch (window_get_algorithm ())
02490     {
02491     case ALGORITHM_MONTE_CARLO:
02492       gtk_widget_show (GTK_WIDGET (window->label_simulations));
02493       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
02494       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02495       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02496       if (i > 1)
02497         {
02498           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02499           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02500           gtk_widget_show (GTK_WIDGET (window->label_bests));
02501           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02502         }
02503       break;
02504     case ALGORITHM_SWEEP:
02505       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02506       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02507       gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02508       gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02509       if (i > 1)
02510         {
02511           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02512           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02513           gtk_widget_show (GTK_WIDGET (window->label_bests));
02514           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02515         }
02516       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
```

```
02517       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
02518       break;
02519     default:
02520       gtk_widget_show (GTK_WIDGET (window->label_population));
02521       gtk_widget_show (GTK_WIDGET (window->spin_population));
02522       gtk_widget_show (GTK_WIDGET (window->label_generations));
02523       gtk_widget_show (GTK_WIDGET (window->spin_generations));
02524       gtk_widget_show (GTK_WIDGET (window->label_mutation));
02525       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
02526       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
02527       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
02528       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
02529       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
02530       gtk_widget_show (GTK_WIDGET (window->label_bits));
02531       gtk_widget_show (GTK_WIDGET (window->spin_bits));
02532     }
02533   gtk_widget_set_sensitive
02534     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
02535   gtk_widget_set_sensitive
02536     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
02537   for (i = 0; i < input->ninputs; ++i)
02538     {
02539       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02540       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02541       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
02542       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
02543       g_signal_handler_block
02544         (window->check_template[i], window->id_template[i]);
02545       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
02546       gtk_toggle_button_set_active
02547         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
02548       g_signal_handler_unblock
02549         (window->button_template[i], window->id_input[i]);
02550       g_signal_handler_unblock
02551         (window->check_template[i], window->id_template[i]);
02552     }
02553   if (i > 0)
02554     {
02555       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
02556       gtk_widget_set_sensitive
02557         (GTK_WIDGET (window->button_template[i - 1]),
02558          gtk_toggle_button_get_active
02559          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
02560     }
02561   if (i < MAX_NINPUTS)
02562     {
02563       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02564       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02565       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
02566       gtk_widget_set_sensitive
02567         (GTK_WIDGET (window->button_template[i]),
02568          gtk_toggle_button_get_active
02569          GTK_TOGGLE_BUTTON (window->check_template[i]));
02570       g_signal_handler_block
02571         (window->check_template[i], window->id_template[i]);
02572       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
02573       gtk_toggle_button_set_active
02574         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02575       g_signal_handler_unblock
02576         (window->button_template[i], window->id_input[i]);
02577       g_signal_handler_unblock
02578         (window->check_template[i], window->id_template[i]);
02579     }
02580   while (++i < MAX_NINPUTS)
02581     {
02582       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02583       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02584     }
02585   gtk_widget_set_sensitive
02586     (GTK_WIDGET (window->spin_minabs),
02587      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02588   gtk_widget_set_sensitive
02589     (GTK_WIDGET (window->spin_maxabs),
02590      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02591 }
02592
02597 void
02598 window_set_algorithm ()
02599 {
02600   unsigned int i;
02601   i = window_get_algorithm ();
02602   switch (i)
02603     {
```

```
02604     case ALGORITHM_SWEEP:
02605       input->nsweeps = (unsigned int *) g_realloc
02606         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02607       break;
02608     case ALGORITHM_GENETIC:
02609       input->nbits = (unsigned int *) g_realloc
02610         (input->nbits, input->nvariables * sizeof (unsigned int));
02611     }
02612   window_update ();
02613 }
02614
02619 void
02620 window_set_experiment ()
02621 {
02622   unsigned int i, j;
02623   char *buffer1, *buffer2;
02624 #if DEBUG
02625   fprintf (stderr, "window_set_experiment: start\n");
02626 #endif
02627   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02628   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02629   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02630   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02631   g_free (buffer1);
02632   g_signal_handler_block
02633     (window->button_experiment, window->id_experiment_name);
02634   gtk_file_chooser_set_filename
02635     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
02636   g_signal_handler_unblock
02637     (window->button_experiment, window->id_experiment_name);
02638   g_free (buffer2);
02639   for (j = 0; j < input->ninputs; ++j)
02640     {
02641       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02642       buffer2
02643         = g_build_filename (input->directory, input->template[j][i], NULL);
02644       gtk_file_chooser_set_filename
02645         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02646       g_free (buffer2);
02647       g_signal_handler_unblock
02648         (window->button_template[j], window->id_input[j]);
02649     }
02650 #if DEBUG
02651   fprintf (stderr, "window_set_experiment: end\n");
02652 #endif
02653 }
02654
02659 void
02660 window_remove_experiment ()
02661 {
02662   unsigned int i, j;
02663   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02664   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02665   gtk_combo_box_text_remove (window->combo_experiment, i);
02666   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02667   xmlFree (input->experiment[i]);
02668   --input->nexperiments;
02669   for (j = i; j < input->nexperiments; ++j)
02670     {
02671       input->experiment[j] = input->experiment[j + 1];
02672       input->weight[j] = input->weight[j + 1];
02673     }
02674   j = input->nexperiments - 1;
02675   if (i > j)
02676     i = j;
02677   for (j = 0; j < input->ninputs; ++j)
02678     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02679   g_signal_handler_block
02680     (window->button_experiment, window->id_experiment_name);
02681   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02682   g_signal_handler_unblock
02683     (window->button_experiment, window->id_experiment_name);
02684   for (j = 0; j < input->ninputs; ++j)
02685     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02686   window_update ();
02687 }
02688
02693 void
02694 window_add_experiment ()
02695 {
02696   unsigned int i, j;
02697   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
```

```
02698   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02699   gtk_combo_box_text_insert_text
02700     (window->combo_experiment, i, input->experiment[i]);
02701   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02702   input->experiment = (char **) g_realloc
02703     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02704   input->weight = (double *) g_realloc
02705     (input->weight, (input->nexperiments + 1) * sizeof (double));
02706   for (j = input->nexperiments - 1; j > i; --j)
02707     {
02708       input->experiment[j + 1] = input->experiment[j];
02709       input->weight[j + 1] = input->weight[j];
02710     }
02711   input->experiment[j + 1]
02712     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02713   input->weight[j + 1] = input->weight[j];
02714   ++input->nexperiments;
02715   for (j = 0; j < input->ninputs; ++j)
02716     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02717   g_signal_handler_block
02718     (window->button_experiment, window->id_experiment_name);
02719   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02720   g_signal_handler_unblock
02721     (window->button_experiment, window->id_experiment_name);
02722   for (j = 0; j < input->ninputs; ++j)
02723     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02724   window_update ();
02725 }
02726
02731 void
02732 window_name_experiment ()
02733 {
02734   unsigned int i;
02735   char *buffer;
02736   GFile *file1, *file2;
02737 #if DEBUG
02738   fprintf (stderr, "window_name_experiment: start\n");
02739 #endif
02740   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02741   file1
02742     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02743   file2 = g_file_new_for_path (input->directory);
02744   buffer = g_file_get_relative_path (file2, file1);
02745   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02746   gtk_combo_box_text_remove (window->combo_experiment, i);
02747   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02748   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02749   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02750   g_free (buffer);
02751   g_object_unref (file2);
02752   g_object_unref (file1);
02753 #if DEBUG
02754   fprintf (stderr, "window_name_experiment: end\n");
02755 #endif
02756 }
02757
02762 void
02763 window_weight_experiment ()
02764 {
02765   unsigned int i;
02766 #if DEBUG
02767   fprintf (stderr, "window_weight_experiment: start\n");
02768 #endif
02769   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02770   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02771 #if DEBUG
02772   fprintf (stderr, "window_weight_experiment: end\n");
02773 #endif
02774 }
02775
02781 void
02782 window_inputs_experiment ()
02783 {
02784   unsigned int j;
02785 #if DEBUG
02786   fprintf (stderr, "window_inputs_experiment: start\n");
02787 #endif
02788   j = input->ninputs - 1;
02789   if (j
02790       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02791                                         (window->check_template[j])))
```

```
02792     --input->ninputs;
02793   if (input->ninputs < MAX_NINPUTS
02794       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02795                                        (window->check_template[j])))
02796     {
02797       ++input->ninputs;
02798       for (j = 0; j < input->ninputs; ++j)
02799         {
02800           input->template[j] = (char **)
02801             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02802         }
02803     }
02804   window_update ();
02805 #if DEBUG
02806   fprintf (stderr, "window_inputs_experiment: end\n");
02807 #endif
02808 }
02809
02817 void
02818 window_template_experiment (void *data)
02819 {
02820   unsigned int i, j;
02821   char *buffer;
02822   GFile *file1, *file2;
02823 #if DEBUG
02824   fprintf (stderr, "window_template_experiment: start\n");
02825 #endif
02826   i = (size_t) data;
02827   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02828   file1
02829     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02830   file2 = g_file_new_for_path (input->directory);
02831   buffer = g_file_get_relative_path (file2, file1);
02832   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02833   g_free (buffer);
02834   g_object_unref (file2);
02835   g_object_unref (file1);
02836 #if DEBUG
02837   fprintf (stderr, "window_template_experiment: end\n");
02838 #endif
02839 }
02840
02845 void
02846 window_set_variable ()
02847 {
02848   unsigned int i;
02849 #if DEBUG
02850   fprintf (stderr, "window_set_variable: start\n");
02851 #endif
02852   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02853   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02854   gtk_entry_set_text (window->entry_variable, input->label[i]);
02855   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02856   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02857   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
02858   if (input->rangeminabs[i] != -G_MAXDOUBLE)
02859     {
02860       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
02861       gtk_toggle_button_set_active
02862         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
02863     }
02864   else
02865     {
02866       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
02867       gtk_toggle_button_set_active
02868         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
02869     }
02870   if (input->rangemaxabs[i] != G_MAXDOUBLE)
02871     {
02872       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
02873       gtk_toggle_button_set_active
02874         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
02875     }
02876   else
02877     {
02878       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
02879       gtk_toggle_button_set_active
02880         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
02881     }
02882   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
02883   switch (input->algorithm)
02884     {
```

```
02885     case ALGORITHM_SWEEP:
02886       gtk_spin_button_set_value (window->spin_sweeps,
02887                                  (gdouble) input->nsweeps[i]);
02888       break;
02889     case ALGORITHM_GENETIC:
02890       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02891       break;
02892     }
02893   window_update ();
02894 #if DEBUG
02895   fprintf (stderr, "window_set_variable: end\n");
02896 #endif
02897 }
02898
02903 void
02904 window_remove_variable ()
02905 {
02906   unsigned int i, j;
02907   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02908   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02909   gtk_combo_box_text_remove (window->combo_variable, i);
02910   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02911   xmlFree (input->label[i]);
02912   --input->nvariables;
02913   for (j = i; j < input->nvariables; ++j)
02914     {
02915       input->label[j] = input->label[j + 1];
02916       input->rangemin[j] = input->rangemin[j + 1];
02917       input->rangemax[j] = input->rangemax[j + 1];
02918       input->rangeminabs[j] = input->rangeminabs[j + 1];
02919       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
02920       input->precision[j] = input->precision[j + 1];
02921       switch (window_get_algorithm ())
02922         {
02923         case ALGORITHM_SWEEP:
02924           input->nsweeps[j] = input->nsweeps[j + 1];
02925           break;
02926         case ALGORITHM_GENETIC:
02927           input->nbits[j] = input->nbits[j + 1];
02928         }
02929     }
02930   j = input->nvariables - 1;
02931   if (i > j)
02932     i = j;
02933   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02934   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
02935   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02936   window_update ();
02937 }
02938
02943 void
02944 window_add_variable ()
02945 {
02946   unsigned int i, j;
02947 #if DEBUG
02948   fprintf (stderr, "window_add_variable: start\n");
02949 #endif
02950   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02951   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02952   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
02953   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02954   input->label = (char **) g_realloc
02955     (input->label, (input->nvariables + 1) * sizeof (char *));
02956   input->rangemin = (double *) g_realloc
02957     (input->rangemin, (input->nvariables + 1) * sizeof (double));
02958   input->rangemax = (double *) g_realloc
02959     (input->rangemax, (input->nvariables + 1) * sizeof (double));
02960   input->rangeminabs = (double *) g_realloc
02961     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
02962   input->rangemaxabs = (double *) g_realloc
02963     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
02964   input->precision = (unsigned int *) g_realloc
02965     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
02966   for (j = input->nvariables - 1; j > i; --j)
02967     {
02968       input->label[j + 1] = input->label[j];
02969       input->rangemin[j + 1] = input->rangemin[j];
02970       input->rangemax[j + 1] = input->rangemax[j];
02971       input->rangeminabs[j + 1] = input->rangeminabs[j];
```

```
02972       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02973       input->precision[j + 1] = input->precision[j];
02974     }
02975   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
02976   input->rangemin[j + 1] = input->rangemin[j];
02977   input->rangemax[j + 1] = input->rangemax[j];
02978   input->rangeminabs[j + 1] = input->rangeminabs[j];
02979   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02980   input->precision[j + 1] = input->precision[j];
02981   switch (window_get_algorithm ())
02982     {
02983     case ALGORITHM_SWEEP:
02984       input->nsweeps = (unsigned int *) g_realloc
02985         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
02986       for (j = input->nvariables - 1; j > i; --j)
02987         input->nsweeps[j + 1] = input->nsweeps[j];
02988       input->nsweeps[j + 1] = input->nsweeps[j];
02989       break;
02990     case ALGORITHM_GENETIC:
02991       input->nbits = (unsigned int *) g_realloc
02992         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
02993       for (j = input->nvariables - 1; j > i; --j)
02994         input->nbits[j + 1] = input->nbits[j];
02995       input->nbits[j + 1] = input->nbits[j];
02996     }
02997   ++input->nvariables;
02998   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
02999   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03000   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
03001   window_update ();
03002 #if DEBUG
03003   fprintf (stderr, "window_add_variable: end\n");
03004 #endif
03005 }
03006
03011 void
03012 window_label_variable ()
03013 {
03014   unsigned int i;
03015   const char *buffer;
03016 #if DEBUG
03017   fprintf (stderr, "window_label_variable: start\n");
03018 #endif
03019   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03020   buffer = gtk_entry_get_text (window->entry_variable);
03021   g_signal_handler_block (window->combo_variable, window->
    id_variable);
03022   gtk_combo_box_text_remove (window->combo_variable, i);
03023   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03024   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03025   g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
03026 #if DEBUG
03027   fprintf (stderr, "window_label_variable: end\n");
03028 #endif
03029 }
03030
03035 void
03036 window_precision_variable ()
03037 {
03038   unsigned int i;
03039 #if DEBUG
03040   fprintf (stderr, "window_precision_variable: start\n");
03041 #endif
03042   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03043   input->precision[i]
03044     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03045   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03046   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03047   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03048   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03049 #if DEBUG
03050   fprintf (stderr, "window_precision_variable: end\n");
03051 #endif
03052 }
03053
03058 void
03059 window_rangemin_variable ()
03060 {
03061   unsigned int i;
03062 #if DEBUG
03063   fprintf (stderr, "window_rangemin_variable: start\n");
03064 #endif
03065   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03066   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
```

```
03067 #if DEBUG
03068   fprintf (stderr, "window_rangemin_variable: end\n");
03069 #endif
03070 }
03071
03076 void
03077 window_rangemax_variable ()
03078 {
03079   unsigned int i;
03080 #if DEBUG
03081   fprintf (stderr, "window_rangemax_variable: start\n");
03082 #endif
03083   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03084   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03085 #if DEBUG
03086   fprintf (stderr, "window_rangemax_variable: end\n");
03087 #endif
03088 }
03089
03094 void
03095 window_rangeminabs_variable ()
03096 {
03097   unsigned int i;
03098 #if DEBUG
03099   fprintf (stderr, "window_rangeminabs_variable: start\n");
03100 #endif
03101   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03102   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03103 #if DEBUG
03104   fprintf (stderr, "window_rangeminabs_variable: end\n");
03105 #endif
03106 }
03107
03112 void
03113 window_rangemaxabs_variable ()
03114 {
03115   unsigned int i;
03116 #if DEBUG
03117   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03118 #endif
03119   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03120   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03121 #if DEBUG
03122   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03123 #endif
03124 }
03125
03130 void
03131 window_update_variable ()
03132 {
03133   unsigned int i;
03134 #if DEBUG
03135   fprintf (stderr, "window_update_variable: start\n");
03136 #endif
03137   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03138   switch (window_get_algorithm ())
03139     {
03140     case ALGORITHM_SWEEP:
03141       input->nsweeps[i]
03142         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03143       break;
03144     case ALGORITHM_GENETIC:
03145       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03146     }
03147 #if DEBUG
03148   fprintf (stderr, "window_update_variable: end\n");
03149 #endif
03150 }
03151
03159 int
03160 window_read (char *filename)
03161 {
03162   unsigned int i;
03163   char *buffer, *directory, *name;
03164 #if DEBUG
03165   fprintf (stderr, "window_read: start\n");
03166 #endif
03167   directory = name = NULL;
03168   if (input->directory) directory = g_strdup (input->directory);
03169   if (input->name) name = g_strdup (input->name);
03170   input_free ();
03171   if (!input_open (filename))
03172     {
03173 #if DEBUG
03174           fprintf (stderr, "window_read: error reading input file\n");
```

```
03175 #endif
03176       buffer = g_build_filename (directory, name, NULL);
03177       if (!input_open (buffer))
03178         {
03179 #if DEBUG
03180           fprintf (stderr, "window_read: error reading backup file\n");
03181 #endif
03182           g_free (buffer);
03183           g_free (name);
03184           g_free (directory);
03185 #if DEBUG
03186           fprintf (stderr, "window_read: end\n");
03187 #endif
03188           return 0;
03189         }
03190       g_free (buffer);
03191     }
03192   g_free (name);
03193   g_free (directory);
03194   buffer = g_build_filename (input->directory, input->simulator, NULL);
03195   puts (buffer);
03196   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03197                                  (window->button_simulator), buffer);
03198   g_free (buffer);
03199   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03200                                 (size_t) input->evaluator);
03201   if (input->evaluator)
03202     {
03203       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03204       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03205                                      (window->button_evaluator), buffer);
03206       g_free (buffer);
03207     }
03208   gtk_toggle_button_set_active
03209     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03210   switch (input->algorithm)
03211     {
03212     case ALGORITHM_MONTE_CARLO:
03213       gtk_spin_button_set_value (window->spin_simulations,
03214                                  (gdouble) input->nsimulations);
03215     case ALGORITHM_SWEEP:
03216       gtk_spin_button_set_value (window->spin_iterations,
03217                                  (gdouble) input->niterations);
03218       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03219       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03220       break;
03221     default:
03222       gtk_spin_button_set_value (window->spin_population,
03223                                  (gdouble) input->nsimulations);
03224       gtk_spin_button_set_value (window->spin_generations,
03225                                  (gdouble) input->niterations);
03226       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03227       gtk_spin_button_set_value (window->spin_reproduction,
03228                                  input->reproduction_ratio);
03229       gtk_spin_button_set_value (window->spin_adaptation,
03230                                  input->adaptation_ratio);
03231     }
03232   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03233   g_signal_handler_block (window->button_experiment,
03234                           window->id_experiment_name);
03235   gtk_combo_box_text_remove_all (window->combo_experiment);
03236   for (i = 0; i < input->nexperiments; ++i)
03237     gtk_combo_box_text_append_text (window->combo_experiment,
03238                                     input->experiment[i]);
03239   g_signal_handler_unblock
03240     (window->button_experiment, window->id_experiment_name);
03241   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03242   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03243   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03244   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03245   gtk_combo_box_text_remove_all (window->combo_variable);
03246   for (i = 0; i < input->nvariables; ++i)
03247     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03248   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03249   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03250   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
```

```
03251   window_set_variable ();
03252   window_update ();
03253 #if DEBUG
03254   fprintf (stderr, "window_read: end\n");
03255 #endif
03256   return 1;
03257 }
03258
03263 void
03264 window_open ()
03265 {
03266   char *buffer;
03267   GtkFileChooserDialog *dlg;
03268   dlg = (GtkFileChooserDialog *)
03269     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03270                                  window->window,
03271                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03272                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03273                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03274   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03275     {
03276       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03277       if (!window_read (buffer))
03278         gtk_main_quit ();
03279       g_free (buffer);
03280     }
03281   gtk_widget_destroy (GTK_WIDGET (dlg));
03282 }
03283
03288 void
03289 window_new ()
03290 {
03291   unsigned int i;
03292   char *buffer, *buffer2, buffer3[64];
03293   GtkViewport *viewport;
03294   char *label_algorithm[NALGORITHMS] = {
03295     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03296   };
03297
03298   // Creating the window
03299   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
03300
03301   // Finish when closing the window
03302   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
03303
03304   // Setting the window title
03305   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03306
03307   // Creating the open button
03308   window->button_open = (GtkToolButton *) gtk_tool_button_new
03309     (gtk_image_new_from_icon_name ("document-open",
03310                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03311      gettext ("Open"));
03312   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03313
03314   // Creating the save button
03315   window->button_save = (GtkToolButton *) gtk_tool_button_new
03316     (gtk_image_new_from_icon_name ("document-save",
03317                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03318      gettext ("Save"));
03319   g_signal_connect (window->button_save, "clicked", (void (*))
    window_save,
03320                     NULL);
03321
03322   // Creating the run button
03323   window->button_run = (GtkToolButton *) gtk_tool_button_new
03324     (gtk_image_new_from_icon_name ("system-run",
03325                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03326      gettext ("Run"));
03327   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03328
03329   // Creating the options button
03330   window->button_options = (GtkToolButton *) gtk_tool_button_new
03331     (gtk_image_new_from_icon_name ("preferences-system",
03332                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03333      gettext ("Options"));
03334   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03335
03336   // Creating the help button
03337   window->button_help = (GtkToolButton *) gtk_tool_button_new
03338     (gtk_image_new_from_icon_name ("help-browser",
03339                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03340      gettext ("Help"));
03341   g_signal_connect (window->button_help, "clicked", window_help, NULL);
03342
03343   // Creating the about button
03344   window->button_about = (GtkToolButton *) gtk_tool_button_new
```

```
03345     (gtk_image_new_from_icon_name ("help-about",
03346                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03347      gettext ("About"));
03348   g_signal_connect (window->button_about, "clicked", window_about, NULL);
03349
03350   // Creating the exit button
03351   window->button_exit = (GtkToolButton *) gtk_tool_button_new
03352     (gtk_image_new_from_icon_name ("application-exit",
03353                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03354      gettext ("Exit"));
03355   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
03356
03357   // Creating the buttons bar
03358   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
03359   gtk_toolbar_insert
03360     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
03361   gtk_toolbar_insert
03362     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
03363   gtk_toolbar_insert
03364     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
03365   gtk_toolbar_insert
03366     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
03367   gtk_toolbar_insert
03368     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
03369   gtk_toolbar_insert
03370     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
03371   gtk_toolbar_insert
03372     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
03373   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
03374
03375   // Creating the simulator program label and entry
03376   window->label_simulator
03377     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
03378   window->button_simulator = (GtkFileChooserButton *)
03379     gtk_file_chooser_button_new (gettext ("Simulator program"),
03380                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03381   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
03382                                gettext ("Simulator program executable file"));
03383
03384   // Creating the evaluator program label and entry
03385   window->check_evaluator = (GtkCheckButton *)
03386     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
03387   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
03388   window->button_evaluator = (GtkFileChooserButton *)
03389     gtk_file_chooser_button_new (gettext ("Evaluator program"),
03390                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03391   gtk_widget_set_tooltip_text
03392     (GTK_WIDGET (window->button_evaluator),
03393      gettext ("Optional evaluator program executable file"));
03394
03395   // Creating the algorithm properties
03396   window->label_simulations = (GtkLabel *) gtk_label_new
03397     (gettext ("Simulations number"));
03398   window->spin_simulations
03399     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03400   window->label_iterations = (GtkLabel *)
03401     gtk_label_new (gettext ("Iterations number"));
03402   window->spin_iterations
03403     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03404   g_signal_connect
03405     (window->spin_iterations, "value-changed", window_update, NULL);
03406   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
03407   window->spin_tolerance
03408     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03409   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
03410   window->spin_bests
03411     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03412   window->label_population
03413     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
03414   window->spin_population
03415     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03416   window->label_generations
03417     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
03418   window->spin_generations
03419     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03420   window->label_mutation
03421     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
03422   window->spin_mutation
03423     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03424   window->label_reproduction
03425     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
03426   window->spin_reproduction
03427     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03428   window->label_adaptation
03429     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
03430   window->spin_adaptation
```

```
03431     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03432
03433   // Creating the array of algorithms
03434   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
03435   window->button_algorithm[0] = (GtkRadioButton *)
03436     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
03437   gtk_grid_attach (window->grid_algorithm,
03438                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
03439   g_signal_connect (window->button_algorithm[0], "clicked",
03440                     window_set_algorithm, NULL);
03441   for (i = 0; ++i < NALGORITHMS;)
03442     {
03443       window->button_algorithm[i] = (GtkRadioButton *)
03444         gtk_radio_button_new_with_mnemonic
03445         (gtk_radio_button_get_group (window->button_algorithm[0]),
03446          label_algorithm[i]);
03447       gtk_grid_attach (window->grid_algorithm,
03448                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
03449       g_signal_connect (window->button_algorithm[i], "clicked",
03450                         window_set_algorithm, NULL);
03451     }
03452   gtk_grid_attach (window->grid_algorithm,
03453                    GTK_WIDGET (window->label_simulations), 0,
03454                    NALGORITHMS, 1, 1);
03455   gtk_grid_attach (window->grid_algorithm,
03456                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
03457   gtk_grid_attach (window->grid_algorithm,
03458                    GTK_WIDGET (window->label_iterations), 0,
03459                    NALGORITHMS + 1, 1, 1);
03460   gtk_grid_attach (window->grid_algorithm,
03461                    GTK_WIDGET (window->spin_iterations), 1,
03462                    NALGORITHMS + 1, 1, 1);
03463   gtk_grid_attach (window->grid_algorithm,
03464                    GTK_WIDGET (window->label_tolerance), 0,
03465                    NALGORITHMS + 2, 1, 1);
03466   gtk_grid_attach (window->grid_algorithm,
03467                    GTK_WIDGET (window->spin_tolerance), 1,
03468                    NALGORITHMS + 2, 1, 1);
03469   gtk_grid_attach (window->grid_algorithm,
03470                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
03471   gtk_grid_attach (window->grid_algorithm,
03472                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
03473   gtk_grid_attach (window->grid_algorithm,
03474                    GTK_WIDGET (window->label_population), 0,
03475                    NALGORITHMS + 4, 1, 1);
03476   gtk_grid_attach (window->grid_algorithm,
03477                    GTK_WIDGET (window->spin_population), 1,
03478                    NALGORITHMS + 4, 1, 1);
03479   gtk_grid_attach (window->grid_algorithm,
03480                    GTK_WIDGET (window->label_generations), 0,
03481                    NALGORITHMS + 5, 1, 1);
03482   gtk_grid_attach (window->grid_algorithm,
03483                    GTK_WIDGET (window->spin_generations), 1,
03484                    NALGORITHMS + 5, 1, 1);
03485   gtk_grid_attach (window->grid_algorithm,
03486                    GTK_WIDGET (window->label_mutation), 0,
03487                    NALGORITHMS + 6, 1, 1);
03488   gtk_grid_attach (window->grid_algorithm,
03489                    GTK_WIDGET (window->spin_mutation), 1,
03490                    NALGORITHMS + 6, 1, 1);
03491   gtk_grid_attach (window->grid_algorithm,
03492                    GTK_WIDGET (window->label_reproduction), 0,
03493                    NALGORITHMS + 7, 1, 1);
03494   gtk_grid_attach (window->grid_algorithm,
03495                    GTK_WIDGET (window->spin_reproduction), 1,
03496                    NALGORITHMS + 7, 1, 1);
03497   gtk_grid_attach (window->grid_algorithm,
03498                    GTK_WIDGET (window->label_adaptation), 0,
03499                    NALGORITHMS + 8, 1, 1);
03500   gtk_grid_attach (window->grid_algorithm,
03501                    GTK_WIDGET (window->spin_adaptation), 1,
03502                    NALGORITHMS + 8, 1, 1);
03503   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
03504   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
03505                      GTK_WIDGET (window->grid_algorithm));
03506
03507   // Creating the variable widgets
03508   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
03509   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_variable),
03510                                gettext ("Variables selector"));
03511   window->id_variable = g_signal_connect
03512     (window->combo_variable, "changed", window_set_variable, NULL);
03513   window->button_add_variable
03514     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03515                                                    GTK_ICON_SIZE_BUTTON);
03516   g_signal_connect
03517     (window->button_add_variable, "clicked",
```

```
      window_add_variable, NULL);
03518   gtk_widget_set_tooltip_text (GTK_WIDGET
03519                                (window->button_add_variable),
03520                                gettext ("Add variable"));
03521   window->button_remove_variable
03522     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03523                                                    GTK_ICON_SIZE_BUTTON);
03524   g_signal_connect
03525     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
03526   gtk_widget_set_tooltip_text (GTK_WIDGET
03527                                (window->button_remove_variable),
03528                                gettext ("Remove variable"));
03529   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
03530   window->entry_variable = (GtkEntry *) gtk_entry_new ();
03531   window->id_variable_label = g_signal_connect
03532     (window->entry_variable, "changed", window_label_variable, NULL);
03533   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
03534   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
03535     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03536   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03537   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
03538   window->scrolled_min
03539     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03540   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
03541                      GTK_WIDGET (viewport));
03542   g_signal_connect (window->spin_min, "value-changed",
03543                     window_rangemin_variable, NULL);
03544   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
03545   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
03546     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03547   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03548   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
03549   window->scrolled_max
03550     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03551   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
03552                      GTK_WIDGET (viewport));
03553   g_signal_connect (window->spin_max, "value-changed",
03554                     window_rangemax_variable, NULL);
03555   window->check_minabs = (GtkCheckButton *)
03556     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
03557   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
03558   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03559     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03560   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03561   gtk_container_add (GTK_CONTAINER (viewport),
03562                      GTK_WIDGET (window->spin_minabs));
03563   window->scrolled_minabs
03564     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03565   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
03566                      GTK_WIDGET (viewport));
03567   g_signal_connect (window->spin_minabs, "value-changed",
03568                     window_rangeminabs_variable, NULL);
03569   window->check_maxabs = (GtkCheckButton *)
03570     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
03571   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
03572   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03573     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03574   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03575   gtk_container_add (GTK_CONTAINER (viewport),
03576                      GTK_WIDGET (window->spin_maxabs));
03577   window->scrolled_maxabs
03578     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03579   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
03580                      GTK_WIDGET (viewport));
03581   g_signal_connect (window->spin_maxabs, "value-changed",
03582                     window_rangemaxabs_variable, NULL);
03583   window->label_precision
03584     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
03585   window->spin_precision = (GtkSpinButton *)
03586     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
03587   g_signal_connect (window->spin_precision, "value-changed",
03588                     window_precision_variable, NULL);
03589   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
03590   window->spin_sweeps
03591     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03592   g_signal_connect
03593     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
03594   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
03595   window->spin_bits
03596     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03597   g_signal_connect
03598     (window->spin_bits, "value-changed", window_update_variable, NULL);
03599   window->grid_variable = (GtkGrid *) gtk_grid_new ();
03600   gtk_grid_attach (window->grid_variable,
03601                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
03602   gtk_grid_attach (window->grid_variable,
```

```
03603                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
03604   gtk_grid_attach (window->grid_variable,
03605                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
03606   gtk_grid_attach (window->grid_variable,
03607                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
03608   gtk_grid_attach (window->grid_variable,
03609                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
03610   gtk_grid_attach (window->grid_variable,
03611                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
03612   gtk_grid_attach (window->grid_variable,
03613                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
03614   gtk_grid_attach (window->grid_variable,
03615                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
03616   gtk_grid_attach (window->grid_variable,
03617                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
03618   gtk_grid_attach (window->grid_variable,
03619                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
03620   gtk_grid_attach (window->grid_variable,
03621                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
03622   gtk_grid_attach (window->grid_variable,
03623                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
03624   gtk_grid_attach (window->grid_variable,
03625                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
03626   gtk_grid_attach (window->grid_variable,
03627                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
03628   gtk_grid_attach (window->grid_variable,
03629                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
03630   gtk_grid_attach (window->grid_variable,
03631                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
03632   gtk_grid_attach (window->grid_variable,
03633                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
03634   gtk_grid_attach (window->grid_variable,
03635                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
03636   gtk_grid_attach (window->grid_variable,
03637                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
03638   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
03639   gtk_container_add (GTK_CONTAINER (window->frame_variable),
03640                      GTK_WIDGET (window->grid_variable));
03641
03642   // Creating the experiment widgets
03643   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03644   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03645                                gettext ("Experiment selector"));
03646   window->id_experiment = g_signal_connect
03647     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
03648   window->button_add_experiment
03649     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03650                                                    GTK_ICON_SIZE_BUTTON);
03651   g_signal_connect
03652     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
03653   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03654                                gettext ("Add experiment"));
03655   window->button_remove_experiment
03656     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03657                                                    GTK_ICON_SIZE_BUTTON);
03658   g_signal_connect (window->button_remove_experiment, "clicked",
03659                     window_remove_experiment, NULL);
03660   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03661                                gettext ("Remove experiment"));
03662   window->label_experiment
03663     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
03664   window->button_experiment = (GtkFileChooserButton *)
03665     gtk_file_chooser_button_new (gettext ("Experimental data file"),
03666                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03667   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03668                                gettext ("Experimental data file"));
03669   window->id_experiment_name
03670     = g_signal_connect (window->button_experiment, "selection-changed",
03671                         window_name_experiment, NULL);
03672   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
03673   window->spin_weight
03674     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03675   gtk_widget_set_tooltip_text
03676     (GTK_WIDGET (window->spin_weight),
03677      gettext ("Weight factor to build the objective function"));
03678   g_signal_connect
03679     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
03680   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03681   gtk_grid_attach (window->grid_experiment,
03682                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03683   gtk_grid_attach (window->grid_experiment,
03684                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03685   gtk_grid_attach (window->grid_experiment,
03686                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
```

```
03687   gtk_grid_attach (window->grid_experiment,
03688                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03689   gtk_grid_attach (window->grid_experiment,
03690                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03691   gtk_grid_attach (window->grid_experiment,
03692                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03693   gtk_grid_attach (window->grid_experiment,
03694                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03695   for (i = 0; i < MAX_NINPUTS; ++i)
03696     {
03697       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03698       window->check_template[i] = (GtkCheckButton *)
03699         gtk_check_button_new_with_label (buffer3);
03700       window->id_template[i]
03701         = g_signal_connect (window->check_template[i], "toggled",
03702                             window_inputs_experiment, NULL);
03703       gtk_grid_attach (window->grid_experiment,
03704                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03705       window->button_template[i] = (GtkFileChooserButton *)
03706         gtk_file_chooser_button_new (gettext ("Input template"),
03707                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03708       gtk_widget_set_tooltip_text
03709         (GTK_WIDGET (window->button_template[i]),
03710          gettext ("Experimental input template file"));
03711       window->id_input[i]
03712         = g_signal_connect_swapped (window->button_template[i],
03713                                     "selection-changed",
03714                                     (void (*)) window_template_experiment,
03715                                     (void *) (size_t) i);
03716       gtk_grid_attach (window->grid_experiment,
03717                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03718     }
03719   window->frame_experiment
03720     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03721   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
03722                      GTK_WIDGET (window->grid_experiment));
03723
03724   // Creating the grid and attaching the widgets to the grid
03725   window->grid = (GtkGrid *) gtk_grid_new ();
03726   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 6, 1);
03727   gtk_grid_attach (window->grid,
03728                    GTK_WIDGET (window->label_simulator), 0, 1, 1, 1);
03729   gtk_grid_attach (window->grid,
03730                    GTK_WIDGET (window->button_simulator), 1, 1, 1, 1);
03731   gtk_grid_attach (window->grid,
03732                    GTK_WIDGET (window->check_evaluator), 2, 1, 1, 1);
03733   gtk_grid_attach (window->grid,
03734                    GTK_WIDGET (window->button_evaluator), 3, 1, 1, 1);
03735   gtk_grid_attach (window->grid,
03736                    GTK_WIDGET (window->frame_algorithm), 0, 2, 2, 1);
03737   gtk_grid_attach (window->grid,
03738                    GTK_WIDGET (window->frame_variable), 2, 2, 2, 1);
03739   gtk_grid_attach (window->grid,
03740                    GTK_WIDGET (window->frame_experiment), 4, 2, 2, 1);
03741   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
03742
03743   // Setting the window logo
03744   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03745   gtk_window_set_icon (window->window, window->logo);
03746
03747   // Showing the window
03748   gtk_widget_show_all (GTK_WIDGET (window->window));
03749
03750   // In Windows the default scrolled size is wrong
03751 #ifdef G_OS_WIN32
03752   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03753   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03754   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03755   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03756 #endif
03757
03758   // Reading initial example
03759   input_new ();
03760   buffer2 = g_get_current_dir ();
03761   buffer = g_build_filename (buffer2, "tests", "test1", INPUT_FILE, NULL);
03762   g_free (buffer2);
03763   window_read (buffer);
03764   g_free (buffer);
03765 }
03766
03767 #endif
03768
03774 int
03775 cores_number ()
03776 {
03777 #ifdef G_OS_WIN32
```

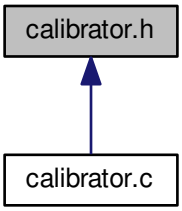
```
03778   SYSTEM_INFO sysinfo;
03779   GetSystemInfo (&sysinfo);
03780   return sysinfo.dwNumberOfProcessors;
03781 #else
03782   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03783 #endif
03784 }
03785
03795 int
03796 main (int argn, char **argc)
03797 {
03798   // Starting pseudo-random numbers generator
03799   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03800   calibrate->seed = DEFAULT_RANDOM_SEED;
03801
03802   // Allowing spaces in the XML data file
03803   xmlKeepBlanksDefault (0);
03804
03805   // Starting MPI
03806 #if HAVE_MPI
03807   MPI_Init (&argn, &argc);
03808   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03809   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03810   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03811 #else
03812   ntasks = 1;
03813 #endif
03814
03815 #if HAVE_GTK
03816
03817   // Getting threads number
03818   nthreads = cores_number ();
03819
03820   // Setting local language and international floating point numbers notation
03821   setlocale (LC_ALL, "");
03822   setlocale (LC_NUMERIC, "C");
03823   window->application_directory = g_get_current_dir ();
03824   bindtextdomain (PROGRAM_INTERFACE,
03825                   g_build_filename (window->application_directory,
03826                                     LOCALE_DIR, NULL));
03827   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03828   textdomain (PROGRAM_INTERFACE);
03829
03830   // Initing GTK+
03831   gtk_disable_setlocale ();
03832   gtk_init (&argn, &argc);
03833
03834   // Opening the main window
03835   window_new ();
03836   gtk_main ();
03837
03838   // Freeing memory
03839   gtk_widget_destroy (GTK_WIDGET (window->window));
03840   g_free (window->application_directory);
03841
03842 #else
03843
03844   // Checking syntax
03845   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03846     {
03847       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03848       return 1;
03849     }
03850
03851   // Getting threads number
03852   if (argn == 2)
03853     nthreads = cores_number ();
03854   else
03855     nthreads = atoi (argc[2]);
03856   printf ("nthreads=%u\n", nthreads);
03857
03858   // Making calibration
03859   input_new ();
03860   if (input_open (argc[argn - 1]))
03861     calibrate_new ();
03862
03863   // Freeing memory
03864   calibrate_free ();
03865
03866 #endif
03867
03868   // Closing MPI
03869 #if HAVE_MPI
03870   MPI_Finalize ();
03871 #endif
03872
03873   // Freeing memory
```

```
03874   gsl_rng_free (calibrate->rng);
03875
03876   // Closing
03877   return 0;
03878 }
```

5.3 calibrator.h File Reference

Header file of the calibrator.

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



Data Structures

- struct Input

  Struct to define the calibration input file.
- struct Calibrate

  Struct to define the calibration data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

  Enum to define the algorithms.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to calibrate with the Monte-Carlo algorithm.
- double calibrate_genetic_objective (Entity *entity)

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_new ()

  Function to open and perform a calibration.

5.3.1 Detailed Description

Header file of the calibrator.

Authors

Javier Burguete.

Copyright

Definition in file calibrator.h.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

Definition at line 43 of file calibrator.h.

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

5.3.3 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1266 of file calibrator.c.

```
01267 {
01268   unsigned int i, j;
01269   double e;
01270 #if DEBUG
01271   fprintf (stderr, "calibrate_best_sequential: start\n");
01272 #endif
01273   if (calibrate->nsaveds < calibrate->nbest
01274       || value < calibrate->error_best[calibrate->nsaveds - 1])
01275     {
01276       if (calibrate->nsaveds < calibrate->nbest)
01277         ++calibrate->nsaveds;
01278       calibrate->error_best[calibrate->nsaveds - 1] = value;
01279       calibrate->simulation_best[calibrate->
     nsaveds - 1] = simulation;
01280       for (i = calibrate->nsaveds; --i;)
01281         {
01282           if (calibrate->error_best[i] < calibrate->
```

```
      error_best[i - 1])
01283             {
01284               j = calibrate->simulation_best[i];
01285               e = calibrate->error_best[i];
01286               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01287               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01288               calibrate->simulation_best[i - 1] = j;
01289               calibrate->error_best[i - 1] = e;
01290             }
01291           else
01292             break;
01293         }
01294     }
01295 #if DEBUG
01296   fprintf (stderr, "calibrate_best_sequential: end\n");
01297 #endif
01298 }
```

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 1221 of file calibrator.c.

```
01222 {
01223   unsigned int i, j;
01224   double e;
01225 #if DEBUG
01226   fprintf (stderr, "calibrate_best_thread: start\n");
01227 #endif
01228   if (calibrate->nsaveds < calibrate->nbest
01229       || value < calibrate->error_best[calibrate->nsaveds - 1])
01230     {
01231       g_mutex_lock (mutex);
01232       if (calibrate->nsaveds < calibrate->nbest)
01233         ++calibrate->nsaveds;
01234       calibrate->error_best[calibrate->nsaveds - 1] = value;
01235       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01236       for (i = calibrate->nsaveds; --i;)
01237         {
01238           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01239             {
01240               j = calibrate->simulation_best[i];
01241               e = calibrate->error_best[i];
01242               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01243               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01244               calibrate->simulation_best[i - 1] = j;
01245               calibrate->error_best[i - 1] = e;
01246             }
01247           else
01248             break;
01249         }
01250       g_mutex_unlock (mutex);
01251     }
01252 #if DEBUG
01253   fprintf (stderr, "calibrate_best_thread: end\n");
01254 #endif
01255 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

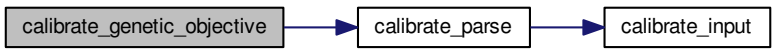
Returns

objective function value.

Definition at line 1575 of file calibrator.c.

```
01576 {
01577   unsigned int j;
01578   double objective;
01579   char buffer[64];
01580 #if DEBUG
01581   fprintf (stderr, "calibrate_genetic_objective: start\n");
01582 #endif
01583   for (j = 0; j < calibrate->nvariables; ++j)
01584     {
01585       calibrate->value[entity->id * calibrate->nvariables + j]
01586         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01587     }
01588   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01589     objective += calibrate_parse (entity->id, j);
01590   g_mutex_lock (mutex);
01591   for (j = 0; j < calibrate->nvariables; ++j)
01592     {
01593       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01594       fprintf (calibrate->file_variables, buffer,
01595                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01596     }
01597   fprintf (calibrate->file_variables, "%.14le\n", objective);
01598   g_mutex_unlock (mutex);
01599 #if DEBUG
01600   fprintf (stderr, "calibrate_genetic_objective: end\n");
01601 #endif
01602   return objective;
01603 }
```

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 970 of file calibrator.c.

```
00971 {
00972   unsigned int i;
00973   char buffer[32], value[32], *buffer2, *buffer3, *content;
00974   FILE *file;
00975   gsize length;
00976   GRegex *regex;
00977
```

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

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 1382 of file calibrator.c.

```
01384 {
01385   unsigned int i, j, k, s[calibrate->nbest];
01386   double e[calibrate->nbest];
01387 #if DEBUG
01388   fprintf (stderr, "calibrate_merge: start\n");
01389 #endif
01390   i = j = k = 0;
01391   do
01392     {
01393       if (i == calibrate->nsaveds)
01394         {
01395           s[k] = simulation_best[j];
01396           e[k] = error_best[j];
01397           ++j;
01398           ++k;
01399           if (j == nsaveds)
01400             break;
01401         }
01402       else if (j == nsaveds)
01403         {
01404           s[k] = calibrate->simulation_best[i];
01405           e[k] = calibrate->error_best[i];
01406           ++i;
01407           ++k;
01408           if (i == calibrate->nsaveds)
01409             break;
01410         }
01411       else if (calibrate->error_best[i] > error_best[j])
01412         {
01413           s[k] = simulation_best[j];
01414           e[k] = error_best[j];
01415           ++j;
01416           ++k;
01417         }
01418       else
01419         {
01420           s[k] = calibrate->simulation_best[i];
01421           e[k] = calibrate->error_best[i];
01422           ++i;
01423           ++k;
01424         }
01425     }
01426   while (k < calibrate->nbest);
01427   calibrate->nsaveds = k;
01428   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01429   memcpy (calibrate->error_best, e, k * sizeof (double));
01430 #if DEBUG
01431   fprintf (stderr, "calibrate_merge: end\n");
01432 #endif
01433 }
```

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 1057 of file calibrator.c.

```
01058 {
01059   unsigned int i;
01060   double e;
01061   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
```

```
01062     *buffer3, *buffer4;
01063   FILE *file_result;
01064
01065 #if DEBUG
01066   fprintf (stderr, "calibrate_parse: start\n");
01067   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01068            experiment);
01069 #endif
01070
01071   // Opening input files
01072   for (i = 0; i < calibrate->ninputs; ++i)
01073     {
01074       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01075 #if DEBUG
01076       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01077 #endif
01078       calibrate_input (simulation, &input[i][0],
01079                        calibrate->file[i][experiment]);
01080     }
01081   for (; i < MAX_NINPUTS; ++i)
01082     strcpy (&input[i][0], "");
01083 #if DEBUG
01084   fprintf (stderr, "calibrate_parse: parsing end\n");
01085 #endif
01086
01087   // Performing the simulation
01088   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01089   buffer2 = g_path_get_dirname (calibrate->simulator);
01090   buffer3 = g_path_get_basename (calibrate->simulator);
01091   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01092   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01093             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01094             input[6], input[7], output);
01095   g_free (buffer4);
01096   g_free (buffer3);
01097   g_free (buffer2);
01098 #if DEBUG
01099   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01100 #endif
01101   system (buffer);
01102
01103   // Checking the objective value function
01104   if (calibrate->evaluator)
01105     {
01106       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01107       buffer2 = g_path_get_dirname (calibrate->evaluator);
01108       buffer3 = g_path_get_basename (calibrate->evaluator);
01109       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01110       snprintf (buffer, 512, "\"%s\" %s %s %s",
01111                 buffer4, output, calibrate->experiment[experiment], result);
01112       g_free (buffer4);
01113       g_free (buffer3);
01114       g_free (buffer2);
01115 #if DEBUG
01116       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01117 #endif
01118       system (buffer);
01119       file_result = fopen (result, "r");
01120       e = atof (fgets (buffer, 512, file_result));
01121       fclose (file_result);
01122     }
01123   else
01124     {
01125       strcpy (result, "");
01126       file_result = fopen (output, "r");
01127       e = atof (fgets (buffer, 512, file_result));
01128       fclose (file_result);
01129     }
01130
01131   // Removing files
01132 #if !DEBUG
01133   for (i = 0; i < calibrate->ninputs; ++i)
01134     {
01135       if (calibrate->file[i][0])
01136         {
01137           snprintf (buffer, 512, RM " %s", &input[i][0]);
01138           system (buffer);
01139         }
01140     }
01141   snprintf (buffer, 512, RM " %s %s", output, result);
01142   system (buffer);
01143 #endif
01144
01145 #if DEBUG
01146   fprintf (stderr, "calibrate_parse: end\n");
01147 #endif
01148
```

```
01149   // Returning the objective function
01150   return e * calibrate->weight[experiment];
01151 }
```

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 1193 of file calibrator.c.

```
01194 {
01195   unsigned int i;
01196   char buffer[64];
01197 #if DEBUG
01198   fprintf (stderr, "calibrate_save_variables: start\n");
01199 #endif
01200   for (i = 0; i < calibrate->nvariables; ++i)
01201     {
01202       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01203       fprintf (calibrate->file_variables, buffer,
01204                calibrate->value[simulation * calibrate->
      nvariables + i]);
01205     }
01206   fprintf (calibrate->file_variables, "%.14le\n", error);
01207 #if DEBUG
01208   fprintf (stderr, "calibrate_save_variables: end\n");
01209 #endif
01210 }
```

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

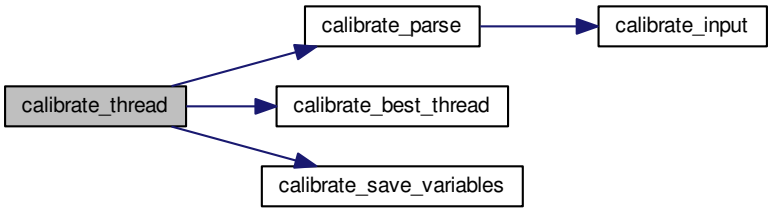
NULL

Definition at line 1308 of file calibrator.c.

```
01309 {
01310   unsigned int i, j, thread;
01311   double e;
01312 #if DEBUG
```

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

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 471 of file calibrator.c.

```
00472 {
00473   char buffer2[64];
00474   xmlDoc *doc;
00475   xmlNode *node, *child;
00476   xmlChar *buffer;
00477   char *msg;
00478   int error_code;
00479   unsigned int i;
00480
00481 #if DEBUG
00482   fprintf (stderr, "input_new: start\n");
00483 #endif
00484
```

```
00485   // Resetting input data
00486   input_new ();
00487
00488   // Parsing the input file
00489   doc = xmlParseFile (filename);
00490   if (!doc)
00491     {
00492       msg = gettext ("Unable to parse the input file");
00493       goto exit_on_error;
00494     }
00495
00496   // Getting the root node
00497   node = xmlDocGetRootElement (doc);
00498   if (xmlStrcmp (node->name, XML_CALIBRATE))
00499     {
00500       msg = gettext ("Bad root XML node");
00501       goto exit_on_error;
00502     }
00503
00504   // Opening simulator program name
00505   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00506   if (!input->simulator)
00507     {
00508       msg = gettext ("Bad simulator program");
00509       goto exit_on_error;
00510     }
00511
00512   // Opening evaluator program name
00513   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00514
00515   // Obtaining pseudo-random numbers generator seed
00516   if (!xmlHasProp (node, XML_SEED))
00517     input->seed = DEFAULT_RANDOM_SEED;
00518   else
00519     {
00520       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00521       if (error_code)
00522         {
00523           msg = gettext ("Bad pseudo-random numbers generator seed");
00524           goto exit_on_error;
00525         }
00526     }
00527
00528   // Opening algorithm
00529   buffer = xmlGetProp (node, XML_ALGORITHM);
00530   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00531     {
00532       input->algorithm = ALGORITHM_MONTE_CARLO;
00533
00534       // Obtaining simulations number
00535       input->nsimulations
00536         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00537       if (error_code)
00538         {
00539           msg = gettext ("Bad simulations number");
00540           goto exit_on_error;
00541         }
00542     }
00543   else if (!xmlStrcmp (buffer, XML_SWEEP))
00544     input->algorithm = ALGORITHM_SWEEP;
00545   else if (!xmlStrcmp (buffer, XML_GENETIC))
00546     {
00547       input->algorithm = ALGORITHM_GENETIC;
00548
00549       // Obtaining population
00550       if (xmlHasProp (node, XML_NPOPULATION))
00551         {
00552           input->nsimulations
00553             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00554           if (error_code || input->nsimulations < 3)
00555             {
00556               msg = gettext ("Invalid population number");
00557               goto exit_on_error;
00558             }
00559         }
00560       else
00561         {
00562           msg = gettext ("No population number");
00563           goto exit_on_error;
00564         }
00565
00566       // Obtaining generations
00567       if (xmlHasProp (node, XML_NGENERATIONS))
00568         {
00569           input->niterations
00570             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00571           if (error_code || !input->niterations)
```

```
00572             {
00573               msg = gettext ("Invalid generations number");
00574               goto exit_on_error;
00575             }
00576         }
00577       else
00578         {
00579           msg = gettext ("No generations number");
00580           goto exit_on_error;
00581         }
00582
00583       // Obtaining mutation probability
00584       if (xmlHasProp (node, XML_MUTATION))
00585         {
00586           input->mutation_ratio
00587             = xml_node_get_float (node, XML_MUTATION, &error_code);
00588           if (error_code || input->mutation_ratio < 0.
00589               || input->mutation_ratio >= 1.)
00590             {
00591               msg = gettext ("Invalid mutation probability");
00592               goto exit_on_error;
00593             }
00594         }
00595       else
00596         {
00597           msg = gettext ("No mutation probability");
00598           goto exit_on_error;
00599         }
00600
00601       // Obtaining reproduction probability
00602       if (xmlHasProp (node, XML_REPRODUCTION))
00603         {
00604           input->reproduction_ratio
00605             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00606           if (error_code || input->reproduction_ratio < 0.
00607               || input->reproduction_ratio >= 1.0)
00608             {
00609               msg = gettext ("Invalid reproduction probability");
00610               goto exit_on_error;
00611             }
00612         }
00613       else
00614         {
00615           msg = gettext ("No reproduction probability");
00616           goto exit_on_error;
00617         }
00618
00619       // Obtaining adaptation probability
00620       if (xmlHasProp (node, XML_ADAPTATION))
00621         {
00622           input->adaptation_ratio
00623             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00624           if (error_code || input->adaptation_ratio < 0.
00625               || input->adaptation_ratio >= 1.)
00626             {
00627               msg = gettext ("Invalid adaptation probability");
00628               goto exit_on_error;
00629             }
00630         }
00631       else
00632         {
00633           msg = gettext ("No adaptation probability");
00634           goto exit_on_error;
00635         }
00636
00637       // Checking survivals
00638       i = input->mutation_ratio * input->nsimulations;
00639       i += input->reproduction_ratio * input->
     nsimulations;
00640       i += input->adaptation_ratio * input->
     nsimulations;
00641       if (i > input->nsimulations - 2)
00642         {
00643           msg = gettext
00644             ("No enough survival entities to reproduce the population");
00645           goto exit_on_error;
00646         }
00647     }
00648   else
00649     {
00650       msg = gettext ("Unknown algorithm");
00651       goto exit_on_error;
00652     }
00653
00654   if (input->algorithm == ALGORITHM_MONTE_CARLO
00655       || input->algorithm == ALGORITHM_SWEEP)
00656     {
```

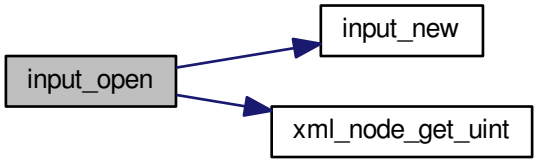
```
00657
00658       // Obtaining iterations number
00659       input->niterations
00660         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00661       if (error_code == 1)
00662         input->niterations = 1;
00663       else if (error_code)
00664         {
00665           msg = gettext ("Bad iterations number");
00666           goto exit_on_error;
00667         }
00668
00669       // Obtaining best number
00670       if (xmlHasProp (node, XML_NBEST))
00671         {
00672           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00673           if (error_code || !input->nbest)
00674             {
00675               msg = gettext ("Invalid best number");
00676               goto exit_on_error;
00677             }
00678         }
00679       else
00680         input->nbest = 1;
00681
00682       // Obtaining tolerance
00683       if (xmlHasProp (node, XML_TOLERANCE))
00684         {
00685           input->tolerance
00686             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00687           if (error_code || input->tolerance < 0.)
00688             {
00689               msg = gettext ("Invalid tolerance");
00690               goto exit_on_error;
00691             }
00692         }
00693       else
00694         input->tolerance = 0.;
00695     }
00696
00697   // Reading the experimental data
00698   for (child = node->children; child; child = child->next)
00699     {
00700       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00701         break;
00702 #if DEBUG
00703       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00704 #endif
00705       if (xmlHasProp (child, XML_NAME))
00706         {
00707           input->experiment
00708             = g_realloc (input->experiment,
00709                          (1 + input->nexperiments) * sizeof (char *));
00710           input->experiment[input->nexperiments]
00711             = (char *) xmlGetProp (child, XML_NAME);
00712         }
00713       else
00714         {
00715           msg = gettext ("No experiment file name");
00716           goto exit_on_error;
00717         }
00718 #if DEBUG
00719       fprintf (stderr, "input_new: experiment=%s\n",
00720                input->experiment[input->nexperiments]);
00721 #endif
00722       input->weight = g_realloc (input->weight,
00723                                  (1 + input->nexperiments) * sizeof (double));
00724       if (xmlHasProp (child, XML_WEIGHT))
00725         {
00726           input->weight[input->nexperiments]
00727             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00728           if (error_code)
00729             {
00730               msg = gettext ("Bad weight");
00731               goto exit_on_error;
00732             }
00733         }
00734       else
00735         input->weight[input->nexperiments] = 1.;
00736 #if DEBUG
00737       fprintf (stderr, "input_new: weight=%lg\n",
00738                input->weight[input->nexperiments]);
00739 #endif
00740       if (!input->nexperiments)
00741         input->ninputs = 0;
00742 #if DEBUG
```

```
00743       fprintf (stderr, "input_new: template[0]\n");
00744 #endif
00745       if (xmlHasProp (child, XML_TEMPLATE1))
00746         {
00747           input->template[0]
00748             = (char **) g_realloc (input->template[0],
00749                                    (1 + input->nexperiments) * sizeof (char *));
00750           input->template[0][input->nexperiments]
00751             = (char *) xmlGetProp (child, template[0]);
00752 #if DEBUG
00753           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00754                    input->nexperiments,
00755                    input->template[0][input->nexperiments]);
00756 #endif
00757           if (!input->nexperiments)
00758             ++input->ninputs;
00759 #if DEBUG
00760           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00761 #endif
00762         }
00763       else
00764         {
00765           msg = gettext ("No experiment template");
00766           goto exit_on_error;
00767         }
00768       for (i = 1; i < MAX_NINPUTS; ++i)
00769         {
00770 #if DEBUG
00771           fprintf (stderr, "input_new: template%u\n", i + 1);
00772 #endif
00773           if (xmlHasProp (child, template[i]))
00774             {
00775               if (input->nexperiments && input->ninputs < 2)
00776                 {
00777                   snprintf (buffer2, 64,
00778                             gettext ("Experiment %u: bad templates number"),
00779                             input->nexperiments + 1);
00780                   msg = buffer2;
00781                   goto exit_on_error;
00782                 }
00783               input->template[i] = (char **)
00784                 g_realloc (input->template[i],
00785                            (1 + input->nexperiments) * sizeof (char *));
00786               input->template[i][input->nexperiments]
00787                 = (char *) xmlGetProp (child, template[i]);
00788 #if DEBUG
00789               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00790                        input->nexperiments, i + 1,
00791                        input->template[i][input->nexperiments]);
00792 #endif
00793               if (!input->nexperiments)
00794                 ++input->ninputs;
00795 #if DEBUG
00796               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00797 #endif
00798             }
00799           else if (input->nexperiments && input->ninputs > 1)
00800             {
00801               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00802                         input->nexperiments + 1, i + 1);
00803               msg = buffer2;
00804               goto exit_on_error;
00805             }
00806           else
00807             break;
00808         }
00809       ++input->nexperiments;
00810 #if DEBUG
00811       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00812 #endif
00813     }
00814   if (!input->nexperiments)
00815     {
00816       msg = gettext ("No calibration experiments");
00817       goto exit_on_error;
00818     }
00819
00820   // Reading the variables data
00821   for (; child; child = child->next)
00822     {
00823       if (xmlStrcmp (child->name, XML_VARIABLE))
00824         {
00825           msg = gettext ("Bad XML node");
00826           goto exit_on_error;
00827         }
00828       if (xmlHasProp (child, XML_NAME))
00829         {
```

```
00830           input->label = g_realloc
00831             (input->label, (1 + input->nvariables) * sizeof (char *));
00832           input->label[input->nvariables]
00833             = (char *) xmlGetProp (child, XML_NAME);
00834         }
00835       else
00836         {
00837           msg = gettext ("No variable name");
00838           goto exit_on_error;
00839         }
00840       if (xmlHasProp (child, XML_MINIMUM))
00841         {
00842           input->rangemin = g_realloc
00843             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00844           input->rangeminabs = g_realloc
00845             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00846           input->rangemin[input->nvariables]
00847             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00848           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00849             {
00850               input->rangeminabs[input->nvariables]
00851                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00852             }
00853           else
00854             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00855         }
00856       else
00857         {
00858           msg = gettext ("No minimum range");
00859           goto exit_on_error;
00860         }
00861       if (xmlHasProp (child, XML_MAXIMUM))
00862         {
00863           input->rangemax = g_realloc
00864             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00865           input->rangemaxabs = g_realloc
00866             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00867           input->rangemax[input->nvariables]
00868             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00869           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00870             input->rangemaxabs[input->nvariables]
00871               = xml_node_get_float (child,
    XML_ABSOLUTE_MAXIMUM, &error_code);
00872           else
00873             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00874         }
00875       else
00876         {
00877           msg = gettext ("No maximum range");
00878           goto exit_on_error;
00879         }
00880       input->precision = g_realloc
00881         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00882       if (xmlHasProp (child, XML_PRECISION))
00883         input->precision[input->nvariables]
00884           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00885       else
00886         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00887       if (input->algorithm == ALGORITHM_SWEEP)
00888         {
00889           if (xmlHasProp (child, XML_NSWEEPS))
00890             {
00891               input->nsweeps = (unsigned int *)
00892                 g_realloc (input->nsweeps,
00893                            (1 + input->nvariables) * sizeof (unsigned int));
00894               input->nsweeps[input->nvariables]
00895                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00896             }
00897           else
00898             {
00899               msg = gettext ("No sweeps number");
00900               goto exit_on_error;
00901             }
00902 #if DEBUG
00903           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00904                    input->nsweeps[input->nvariables],
    input->nsimulations);
00905 #endif
00906         }
00907       if (input->algorithm == ALGORITHM_GENETIC)
00908         {
00909           // Obtaining bits representing each variable
00910           if (xmlHasProp (child, XML_NBITS))
00911             {
00912               input->nbits = (unsigned int *)
```

```
00913                 g_realloc (input->nbits,
00914                            (1 + input->nvariables) * sizeof (unsigned int));
00915               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00916               if (error_code || !i)
00917                 {
00918                   msg = gettext ("Invalid bit number");
00919                   goto exit_on_error;
00920                 }
00921               input->nbits[input->nvariables] = i;
00922             }
00923           else
00924             {
00925               msg = gettext ("No bits number");
00926               goto exit_on_error;
00927             }
00928         }
00929       ++input->nvariables;
00930     }
00931   if (!input->nvariables)
00932     {
00933       msg = gettext ("No calibration variables");
00934       goto exit_on_error;
00935     }
00936
00937   // Getting the working directory
00938   input->directory = g_path_get_dirname (filename);
00939   input->name = g_path_get_basename (filename);
00940
00941   // Closing the XML document
00942   xmlFreeDoc (doc);
00943
00944 #if DEBUG
00945   fprintf (stderr, "input_new: end\n");
00946 #endif
00947   return 1;
00948
00949 exit_on_error:
00950   show_error (msg);
00951   input_free ();
00952 #if DEBUG
00953   fprintf (stderr, "input_new: end\n");
00954 #endif
00955   return 0;
00956 }
```

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.

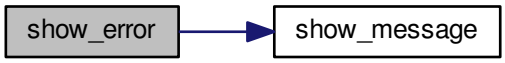
Parameters

msg	Error message.

Definition at line 245 of file calibrator.c.

```
00246 {
00247   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00248 }
```

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 215 of file calibrator.c.

```
00216 {
00217 #if HAVE_GTK
00218   GtkMessageDialog *dlg;
00219
00220   // Creating the dialog
00221   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00222     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00223
00224   // Setting the dialog title
00225   gtk_window_set_title (GTK_WINDOW (dlg), title);
00226
00227   // Showing the dialog and waiting response
00228   gtk_dialog_run (GTK_DIALOG (dlg));
00229
00230   // Closing and freeing memory
00231   gtk_widget_destroy (GTK_WIDGET (dlg));
00232
00233 #else
00234   printf ("%s: %s\n", title, msg);
00235 #endif
00236 }
```

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 324 of file calibrator.c.

```
00325 {
```

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

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 262 of file calibrator.c.

```
00263 {
00264   int i = 0;
00265   xmlChar *buffer;
00266   buffer = xmlGetProp (node, prop);
00267   if (!buffer)
00268     *error_code = 1;
00269   else
00270     {
00271       if (sscanf ((char *) buffer, "%d", &i) != 1)
00272         *error_code = 2;
00273       else
00274         *error_code = 0;
00275       xmlFree (buffer);
00276     }
00277   return i;
00278 }
```

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 293 of file calibrator.c.

```
00294 {
00295   unsigned int i = 0;
00296   xmlChar *buffer;
00297   buffer = xmlGetProp (node, prop);
00298   if (!buffer)
00299     *error_code = 1;
00300   else
00301     {
00302       if (sscanf ((char *) buffer, "%u", &i) != 1)
00303         *error_code = 2;
00304       else
00305         *error_code = 0;
00306       xmlFree (buffer);
00307     }
00308   return i;
00309 }
```

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 391 of file calibrator.c.

```
00392 {
00393   xmlChar buffer[64];
00394   snprintf ((char *) buffer, 64, "%.14lg", value);
00395   xmlSetProp (node, prop, buffer);
00396 }
```

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 353 of file calibrator.c.

```
00354 {
00355   xmlChar buffer[64];
00356   snprintf ((char *) buffer, 64, "%d", value);
00357   xmlSetProp (node, prop, buffer);
00358 }
```

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 372 of file calibrator.c.

```
00373 {
00374   xmlChar buffer[64];
00375   snprintf ((char *) buffer, 64, "%u", value);
00376   xmlSetProp (node, prop, buffer);
00377 }
```

5.4 calibrator.h

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef CALIBRATOR__H
00037 #define CALIBRATOR__H 1
00038
00043 enum Algorithm
00044 {
00045   ALGORITHM_MONTE_CARLO = 0,
00046   ALGORITHM_SWEEP = 1,
00047   ALGORITHM_GENETIC = 2
00048 };
00049
00054 typedef struct
00055 {
00056   char *simulator;
00057   char *evaluator;
00059   char **experiment;
00060   char **template[MAX_NINPUTS];
00061   char **label;
00062   char *directory;
00063   char *name;
00064   double *rangemin;
00065   double *rangemax;
00066   double *rangeminabs;
00067   double *rangemaxabs;
00068   double *weight;
00069   double tolerance;
00070   double mutation_ratio;
00071   double reproduction_ratio;
00072   double adaptation_ratio;
00073   unsigned long int seed;
00075   unsigned int nvariables;
00076   unsigned int nexperiments;
00077   unsigned int ninputs;
00078   unsigned int nsimulations;
00079   unsigned int algorithm;
00080   unsigned int *precision;
00081   unsigned int *nsweeps;
00082   unsigned int *nbits;
00084   unsigned int niterations;
00085   unsigned int nbest;
```

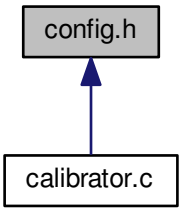
```
00086 } Input;
00087
00092 typedef struct
00093 {
00094   char *simulator;
00095   char *evaluator;
00097   char **experiment;
00098   char **template[MAX_NINPUTS];
00099   char **label;
00100   unsigned int nvariables;
00101   unsigned int nexperiments;
00102   unsigned int ninputs;
00103   unsigned int nsimulations;
00104   unsigned int algorithm;
00105   unsigned int *precision;
00106   unsigned int *nsweeps;
00107   unsigned int nstart;
00108   unsigned int nend;
00109   unsigned int *thread;
00111   unsigned int niterations;
00112   unsigned int nbest;
00113   unsigned int nsaveds;
00114   unsigned int *simulation_best;
00115   unsigned long int seed;
00117   double *value;
00118   double *rangemin;
00119   double *rangemax;
00120   double *rangeminabs;
00121   double *rangemaxabs;
00122   double *error_best;
00123   double *weight;
00124   double *value_old;
00126   double *error_old;
00128   double tolerance;
00129   double mutation_ratio;
00130   double reproduction_ratio;
00131   double adaptation_ratio;
00132   FILE *file_result;
00133   FILE *file_variables;
00134   gsl_rng *rng;
00135   GMappedFile **file[MAX_NINPUTS];
00136   GeneticVariable *genetic_variable;
00138 #if HAVE_MPI
00139   int mpi_rank;
00140 #endif
00141 } Calibrate;
00142
00147 typedef struct
00148 {
00149   unsigned int thread;
00150 } ParallelData;
00151
00152 // Public functions
00153 void show_message (char *title, char *msg, int type);
00154 void show_error (char *msg);
00155 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00156 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00157                                 int *error_code);
00158 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00159                            int *error_code);
00160 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00161 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00162                         unsigned int value);
00163 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00164 void input_new ();
00165 void input_free ();
00166 int input_open (char *filename);
00167 void calibrate_input (unsigned int simulation, char *input,
00168                       GMappedFile * template);
00169 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00170 void calibrate_print ();
00171 void calibrate_save_variables (unsigned int simulation, double error);
00172 void calibrate_best_thread (unsigned int simulation, double value);
00173 void calibrate_best_sequential (unsigned int simulation, double value);
00174 void *calibrate_thread (ParallelData * data);
00175 void calibrate_sequential ();
00176 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00177                       double *error_best);
00178 #if HAVE_MPI
00179 void calibrate_synchronise ();
00180 #endif
00181 void calibrate_sweep ();
00182 void calibrate_MonteCarlo ();
00183 double calibrate_genetic_objective (Entity * entity);
00184 void calibrate_genetic ();
00185 void calibrate_save_old ();
00186 void calibrate_merge_old ();
```

```
00187 void calibrate_refine ();
00188 void calibrate_iterate ();
00189 void calibrate_new ();
00190
00191 #endif
```

5.5 config.h File Reference

Configuration header file.

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



Macros

- #define MAX_NINPUTS 8

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

  Number of algorithms.
- #define NPRECISIONS 15

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

  Default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

  Default pseudo-random numbers seed.
- #define LOCALE_DIR "locales"

  Locales directory.
- #define PROGRAM_INTERFACE "calibrator"

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

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

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

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

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

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

evaluator XML label.

- #define XML_EXPERIMENT (const xmlChar*)"experiment"

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  seed XML label.
- #define XML_SWEEP (const xmlChar*)"sweep"

  sweep XML label.
- #define XML_TEMPLATE1 (const xmlChar*)"template1"

  template1 XML label.
- #define XML_TEMPLATE2 (const xmlChar*)"template2"

  template2 XML label.
- #define XML_TEMPLATE3 (const xmlChar*)"template3"

  template3 XML label.
- #define XML_TEMPLATE4 (const xmlChar*)"template4"

  template4 XML label.
- #define XML_TEMPLATE5 (const xmlChar*)"template5"

  template5 XML label.
- #define XML_TEMPLATE6 (const xmlChar*)"template6"

  template6 XML label.

- #define XML_TEMPLATE7 (const xmlChar*)"template7"

  template7 XML label.
- #define XML_TEMPLATE8 (const xmlChar*)"template8"

  template8 XML label.
- #define XML_TOLERANCE (const xmlChar*)"tolerance"

  tolerance XML label.
- #define XML_VARIABLE (const xmlChar*)"variable"

  variable XML label.
- #define XML_WEIGHT (const xmlChar*)"weight"

  weight XML label.

5.5.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2014, all rights reserved.

Definition in file config.h.

5.6 config.h

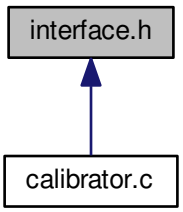
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
00045 #define NPRECISIONS 15
00046
00047 // Default choices
00048
```

```
00049 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00050 #define DEFAULT_RANDOM_SEED 7007
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "calibrator"
00056
00057 // XML labels
00058
00059 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00060 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00062 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00064 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00066 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00068 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00070 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00072 #define XML_GENETIC (const xmlChar*)"genetic"
00074 #define XML_MINIMUM (const xmlChar*)"minimum"
00075 #define XML_MAXIMUM (const xmlChar*)"maximum"
00076 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00077 #define XML_MUTATION (const xmlChar*)"mutation"
00079 #define XML_NAME (const xmlChar*)"name"
00080 #define XML_NBEST (const xmlChar*)"nbest"
00081 #define XML_NBITS (const xmlChar*)"nbits"
00082 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00083 #define XML_NITERATIONS (const xmlChar*)"niterations"
00085 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00087 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00089 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00091 #define XML_PRECISION (const xmlChar*)"precision"
00092 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00094 #define XML_SIMULATOR (const xmlChar*)"simulator"
00096 #define XML_SEED (const xmlChar*)"seed"
00098 #define XML_SWEEP (const xmlChar*)"sweep"
00099 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00100 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00102 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00104 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00106 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00108 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00110 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00112 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00114 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00116 #define XML_VARIABLE (const xmlChar*)"variable"
00118 #define XML_WEIGHT (const xmlChar*)"weight"
00119
00120 #endif
```

5.7 interface.h File Reference

Header file of the interface.

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



Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- void input_save (char *filename)

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

  Function to get the algorithm number.
- void window_update ()

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

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

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

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

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

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

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

  Function to update the experiment input templates number in the main window.

- void window_template_experiment (void *data)

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

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

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

  Function to add a variable in the main window.
- void window_label_variable ()

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

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

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

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

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

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

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

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

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

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

  Function to obtain the cores number.

5.7.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.7.2 Function Documentation

5.7.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 3775 of file calibrator.c.

```
03776 {
03777 #ifdef G_OS_WIN32
03778   SYSTEM_INFO sysinfo;
03779   GetSystemInfo (&sysinfo);
03780   return sysinfo.dwNumberOfProcessors;
03781 #else
03782   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03783 #endif
03784 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

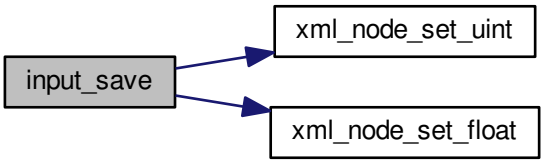
filename	Input file name.

Definition at line 2075 of file calibrator.c.

```
02076 {
02077   unsigned int i, j;
02078   char *buffer;
02079   xmlDoc *doc;
02080   xmlNode *node, *child;
02081   GFile *file, *file2;
02082
02083   // Getting the input file directory
02084   input->name = g_path_get_basename (filename);
02085   input->directory = g_path_get_dirname (filename);
02086   file = g_file_new_for_path (input->directory);
02087
02088   // Opening the input file
02089   doc = xmlNewDoc ((const xmlChar *) "1.0");
02090
02091   // Setting root XML node
02092   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02093   xmlDocSetRootElement (doc, node);
02094
02095   // Adding properties to the root XML node
02096   file2 = g_file_new_for_path (input->simulator);
02097   buffer = g_file_get_relative_path (file, file2);
02098   g_object_unref (file2);
02099   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02100   g_free (buffer);
02101   if (input->evaluator)
02102     {
02103       file2 = g_file_new_for_path (input->evaluator);
02104       buffer = g_file_get_relative_path (file, file2);
02105       g_object_unref (file2);
02106       if (xmlStrlen ((xmlChar *) buffer))
02107         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02108       g_free (buffer);
02109     }
02110   if (input->seed != DEFAULT_RANDOM_SEED)
02111     xml_node_set_uint (node, XML_SEED, input->seed);
02112
02113   // Setting the algorithm
02114   buffer = (char *) g_malloc (64);
02115   switch (input->algorithm)
02116     {
02117     case ALGORITHM_MONTE_CARLO:
02118       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02119       snprintf (buffer, 64, "%u", input->nsimulations);
02120       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02121       snprintf (buffer, 64, "%u", input->niterations);
02122       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02123       snprintf (buffer, 64, "%.3lg", input->tolerance);
02124       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02125       snprintf (buffer, 64, "%u", input->nbest);
02126       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02127       break;
02128     case ALGORITHM_SWEEP:
```

```
02129       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02130       snprintf (buffer, 64, "%u", input->niterations);
02131       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02132       snprintf (buffer, 64, "%.3lg", input->tolerance);
02133       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02134       snprintf (buffer, 64, "%u", input->nbest);
02135       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02136       break;
02137     default:
02138       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02139       snprintf (buffer, 64, "%u", input->nsimulations);
02140       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02141       snprintf (buffer, 64, "%u", input->niterations);
02142       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02143       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02144       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02145       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02146       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02147       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02148       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02149       break;
02150     }
02151   g_free (buffer);
02152
02153   // Setting the experimental data
02154   for (i = 0; i < input->nexperiments; ++i)
02155     {
02156       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02157       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02158       if (input->weight[i] != 1.)
02159         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02160       for (j = 0; j < input->ninputs; ++j)
02161         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02162     }
02163
02164   // Setting the variables data
02165   for (i = 0; i < input->nvariables; ++i)
02166     {
02167       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02168       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02169       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02170       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02171         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02172       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02173       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02174         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02175       if (input->precision[i] != DEFAULT_PRECISION)
02176         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02177       if (input->algorithm == ALGORITHM_SWEEP)
02178         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02179       else if (input->algorithm == ALGORITHM_GENETIC)
02180         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02181     }
02182
02183   // Saving the XML file
02184   xmlSaveFormatFile (filename, doc, 1);
02185
02186   // Freeing memory
02187   xmlFreeDoc (doc);
02188 }
```

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 2444 of file calibrator.c.

```
02445 {
02446   unsigned int i;
02447   for (i = 0; i < NALGORITHMS; ++i)
02448     if (gtk_toggle_button_get_active
02449         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02450       break;
02451   return i;
02452 }
```

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.

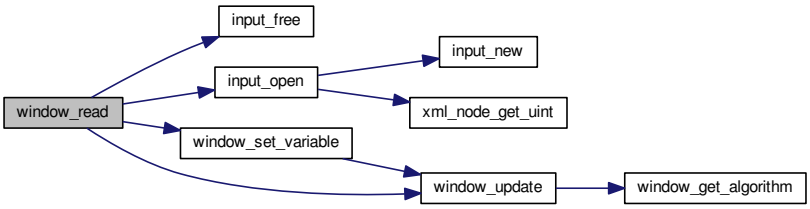
Definition at line 3160 of file calibrator.c.

```
03161 {
03162   unsigned int i;
03163   char *buffer, *directory, *name;
03164 #if DEBUG
03165   fprintf (stderr, "window_read: start\n");
03166 #endif
03167   directory = name = NULL;
03168   if (input->directory) directory = g_strdup (input->
      directory);
03169   if (input->name) name = g_strdup (input->name);
03170   input_free ();
03171   if (!input_open (filename))
03172     {
03173 #if DEBUG
```

```
03174           fprintf (stderr, "window_read: error reading input file\n");
03175 #endif
03176       buffer = g_build_filename (directory, name, NULL);
03177       if (!input_open (buffer))
03178         {
03179 #if DEBUG
03180           fprintf (stderr, "window_read: error reading backup file\n");
03181 #endif
03182           g_free (buffer);
03183           g_free (name);
03184           g_free (directory);
03185 #if DEBUG
03186           fprintf (stderr, "window_read: end\n");
03187 #endif
03188           return 0;
03189         }
03190       g_free (buffer);
03191     }
03192   g_free (name);
03193   g_free (directory);
03194   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03195   puts (buffer);
03196   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03197                                  (window->button_simulator), buffer);
03198   g_free (buffer);
03199   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03200                                 (size_t) input->evaluator);
03201   if (input->evaluator)
03202     {
03203       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03204       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03205                                      (window->button_evaluator), buffer);
03206       g_free (buffer);
03207     }
03208   gtk_toggle_button_set_active
03209     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03210   switch (input->algorithm)
03211     {
03212     case ALGORITHM_MONTE_CARLO:
03213       gtk_spin_button_set_value (window->spin_simulations,
03214                                  (gdouble) input->nsimulations);
03215     case ALGORITHM_SWEEP:
03216       gtk_spin_button_set_value (window->spin_iterations,
03217                                  (gdouble) input->niterations);
03218       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03219       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03220       break;
03221     default:
03222       gtk_spin_button_set_value (window->spin_population,
03223                                  (gdouble) input->nsimulations);
03224       gtk_spin_button_set_value (window->spin_generations,
03225                                  (gdouble) input->niterations);
03226       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03227       gtk_spin_button_set_value (window->spin_reproduction,
03228                                  input->reproduction_ratio);
03229       gtk_spin_button_set_value (window->spin_adaptation,
03230                                  input->adaptation_ratio);
03231     }
03232   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03233   g_signal_handler_block (window->button_experiment,
03234                           window->id_experiment_name);
03235   gtk_combo_box_text_remove_all (window->combo_experiment);
03236   for (i = 0; i < input->nexperiments; ++i)
03237     gtk_combo_box_text_append_text (window->combo_experiment,
03238                                     input->experiment[i]);
03239   g_signal_handler_unblock
03240     (window->button_experiment, window->
      id_experiment_name);
03241   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03242   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03243   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03244   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03245   gtk_combo_box_text_remove_all (window->combo_variable);
03246   for (i = 0; i < input->nvariables; ++i)
03247     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03248   g_signal_handler_unblock (window->entry_variable, window->
```

```
      id_variable_label);
03249   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03250   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03251   window_set_variable ();
03252   window_update ();
03253 #if DEBUG
03254   fprintf (stderr, "window_read: end\n");
03255 #endif
03256   return 1;
03257 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

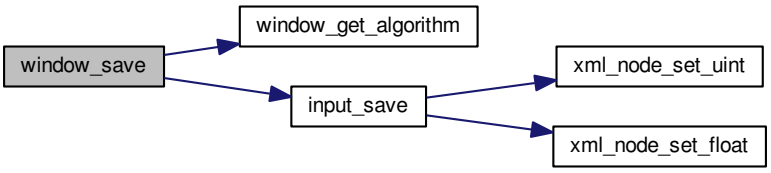
1 on OK, 0 on Cancel.

Definition at line 2263 of file calibrator.c.

```
02264 {
02265   char *buffer;
02266   GtkFileChooserDialog *dlg;
02267
02268 #if DEBUG
02269   fprintf (stderr, "window_save: start\n");
02270 #endif
02271
02272   // Opening the saving dialog
02273   dlg = (GtkFileChooserDialog *)
02274     gtk_file_chooser_dialog_new (gettext ("Save file"),
02275                                  window->window,
02276                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02277                                  gettext ("_Cancel"),
02278                                  GTK_RESPONSE_CANCEL,
02279                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02280   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02281
02282   // If OK response then saving
02283   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02284     {
02285
02286       // Adding properties to the root XML node
02287       input->simulator = gtk_file_chooser_get_filename
02288         (GTK_FILE_CHOOSER (window->button_simulator));
02289       if (gtk_toggle_button_get_active
02290           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02291         input->evaluator = gtk_file_chooser_get_filename
02292           (GTK_FILE_CHOOSER (window->button_evaluator));
02293       else
02294         input->evaluator = NULL;
02295
02296       // Setting the algorithm
02297       switch (window_get_algorithm ())
02298         {
```

```
02299         case ALGORITHM_MONTE_CARLO:
02300           input->algorithm = ALGORITHM_MONTE_CARLO;
02301           input->nsimulations
02302             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02303           input->niterations
02304             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02305           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02306           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02307           break;
02308         case ALGORITHM_SWEEP:
02309           input->algorithm = ALGORITHM_SWEEP;
02310           input->niterations
02311             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02312           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02313           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02314           break;
02315         default:
02316           input->algorithm = ALGORITHM_GENETIC;
02317           input->nsimulations
02318             = gtk_spin_button_get_value_as_int (window->spin_population);
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_generations);
02321           input->mutation_ratio
02322             = gtk_spin_button_get_value (window->spin_mutation);
02323           input->reproduction_ratio
02324             = gtk_spin_button_get_value (window->spin_reproduction);
02325           input->adaptation_ratio
02326             = gtk_spin_button_get_value (window->spin_adaptation);
02327           break;
02328         }
02329
02330       // Saving the XML file
02331       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02332       input_save (buffer);
02333
02334       // Closing and freeing memory
02335       g_free (buffer);
02336       gtk_widget_destroy (GTK_WIDGET (dlg));
02337 #if DEBUG
02338       fprintf (stderr, "window_save: end\n");
02339 #endif
02340       return 1;
02341     }
02342
02343   // Closing and freeing memory
02344   gtk_widget_destroy (GTK_WIDGET (dlg));
02345 #if DEBUG
02346   fprintf (stderr, "window_save: end\n");
02347 #endif
02348   return 0;
02349 }
```

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 2818 of file calibrator.c.

```
02819 {
02820   unsigned int i, j;
02821   char *buffer;
02822   GFile *file1, *file2;
02823 #if DEBUG
02824   fprintf (stderr, "window_template_experiment: start\n");
02825 #endif
02826   i = (size_t) data;
02827   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02828   file1
02829     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02830   file2 = g_file_new_for_path (input->directory);
02831   buffer = g_file_get_relative_path (file2, file1);
02832   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02833   g_free (buffer);
02834   g_object_unref (file2);
02835   g_object_unref (file1);
02836 #if DEBUG
02837   fprintf (stderr, "window_template_experiment: end\n");
02838 #endif
02839 }
```

5.8 interface.h

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef INTERFACE__H
00037 #define INTERFACE__H 1
00038
00039 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00040
00046 typedef struct
00047 {
00048   char *template[MAX_NINPUTS];
00049   char *name;
00050   double weight;
00052 } Experiment;
00053
00058 typedef struct
00059 {
00060   char *label;
00061   double rangemin;
00062   double rangemax;
00063   double rangeminabs;
00064   double rangemaxabs;
00065   unsigned int precision;
00066   unsigned int nsweeps;
00067   unsigned int nbits;
00068 } Variable;
```

```
00069
00074 typedef struct
00075 {
00076   GtkDialog *dialog;
00077   GtkGrid *grid;
00078   GtkLabel *label_processors;
00079   GtkSpinButton *spin_processors;
00080   GtkLabel *label_seed;
00082   GtkSpinButton *spin_seed;
00084 } Options;
00085
00090 typedef struct
00091 {
00092   GtkDialog *dialog;
00093   GtkLabel *label;
00094 } Running;
00095
00100 typedef struct
00101 {
00102   GtkWindow *window;
00103   GtkGrid *grid;
00104   GtkToolbar *bar_buttons;
00105   GtkToolButton *button_open;
00106   GtkToolButton *button_save;
00107   GtkToolButton *button_run;
00108   GtkToolButton *button_options;
00109   GtkToolButton *button_help;
00110   GtkToolButton *button_about;
00111   GtkToolButton *button_exit;
00112   GtkLabel *label_simulator;
00113   GtkFileChooserButton *button_simulator;
00115   GtkCheckButton *check_evaluator;
00116   GtkFileChooserButton *button_evaluator;
00118   GtkFrame *frame_algorithm;
00119   GtkGrid *grid_algorithm;
00120   GtkRadioButton *button_algorithm[NALGORITHMS];
00122   GtkLabel *label_simulations;
00123   GtkSpinButton *spin_simulations;
00125   GtkLabel *label_iterations;
00126   GtkSpinButton *spin_iterations;
00128   GtkLabel *label_tolerance;
00129   GtkSpinButton *spin_tolerance;
00130   GtkLabel *label_bests;
00131   GtkSpinButton *spin_bests;
00132   GtkLabel *label_population;
00133   GtkSpinButton *spin_population;
00135   GtkLabel *label_generations;
00136   GtkSpinButton *spin_generations;
00138   GtkLabel *label_mutation;
00139   GtkSpinButton *spin_mutation;
00140   GtkLabel *label_reproduction;
00141   GtkSpinButton *spin_reproduction;
00143   GtkLabel *label_adaptation;
00144   GtkSpinButton *spin_adaptation;
00146   GtkFrame *frame_variable;
00147   GtkGrid *grid_variable;
00148   GtkComboBoxText *combo_variable;
00150   GtkButton *button_add_variable;
00151   GtkButton *button_remove_variable;
00152   GtkLabel *label_variable;
00153   GtkEntry *entry_variable;
00154   GtkLabel *label_min;
00155   GtkSpinButton *spin_min;
00156   GtkScrolledWindow *scrolled_min;
00157   GtkLabel *label_max;
00158   GtkSpinButton *spin_max;
00159   GtkScrolledWindow *scrolled_max;
00160   GtkCheckButton *check_minabs;
00161   GtkSpinButton *spin_minabs;
00162   GtkScrolledWindow *scrolled_minabs;
00163   GtkCheckButton *check_maxabs;
00164   GtkSpinButton *spin_maxabs;
00165   GtkScrolledWindow *scrolled_maxabs;
00166   GtkLabel *label_precision;
00167   GtkSpinButton *spin_precision;
00168   GtkLabel *label_sweeps;
00169   GtkSpinButton *spin_sweeps;
00170   GtkLabel *label_bits;
00171   GtkSpinButton *spin_bits;
00172   GtkFrame *frame_experiment;
00173   GtkGrid *grid_experiment;
00174   GtkComboBoxText *combo_experiment;
00175   GtkButton *button_add_experiment;
00176   GtkButton *button_remove_experiment;
00177   GtkLabel *label_experiment;
00178   GtkFileChooserButton *button_experiment;
00180   GtkLabel *label_weight;
```

```
00181   GtkSpinButton *spin_weight;
00182   GtkCheckButton *check_template[MAX_NINPUTS];
00184   GtkFileChooserButton *button_template[MAX_NINPUTS];
00186   GdkPixbuf *logo;
00187   Experiment *experiment;
00188   Variable *variable;
00189   char *application_directory;
00190   gulong id_experiment;
00191   gulong id_experiment_name;
00192   gulong id_variable;
00193   gulong id_variable_label;
00194   gulong id_template[MAX_NINPUTS];
00196   gulong id_input[MAX_NINPUTS];
00198   unsigned int nexperiments;
00199   unsigned int nvariables;
00200 } Window;
00201
00202 // Public functions
00203 void input_save (char *filename);
00204 void options_new ();
00205 void running_new ();
00206 int window_save ();
00207 void window_run ();
00208 void window_help ();
00209 int window_get_algorithm ();
00210 void window_update ();
00211 void window_set_algorithm ();
00212 void window_set_experiment ();
00213 void window_remove_experiment ();
00214 void window_add_experiment ();
00215 void window_name_experiment ();
00216 void window_weight_experiment ();
00217 void window_inputs_experiment ();
00218 void window_template_experiment (void *data);
00219 void window_set_variable ();
00220 void window_remove_variable ();
00221 void window_add_variable ();
00222 void window_label_variable ();
00223 void window_precision_variable ();
00224 void window_rangemin_variable ();
00225 void window_rangemax_variable ();
00226 void window_rangeminabs_variable ();
00227 void window_rangemaxabs_variable ();
00228 void window_update_variable ();
00229 int window_read (char *filename);
00230 void window_open ();
00231 void window_new ();
00232 int cores_number ();
00233
00234 #endif
```

Index