Calibrator

1.1.27

Generated by Doxygen 1.8.8

Thu Oct 29 2015 03:43:27

Contents

Chapter 1

CALIBRATOR (1.1.27 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.

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/1.1.27
   $ 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.

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 131 of file calibrator.h.

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

- calibrator.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

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

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 49 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <calibrator.h>`

Data Fields

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

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

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 60 of file calibrator.h.

4.3.2 Field Documentation

4.3.2.1 Input::nbits

Parameters

Array	of bits numbers of the genetic algorithm.

Definition at line 122 of file calibrator.h.

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

- calibrator.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkLabel * label_processors

  Processors number GtkLabel.
- GtkLabel * label_seed

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

  Processors number GtkSpinButton.
- GtkSpinButton * spin_seed

  Pseudo-random numbers generator seed GtkSpinButton.
- GtkGrid * grid

  main GtkGrid.
- GtkDialog * dialog

  main GtkDialog.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 96 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <calibrator.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 234 of file calibrator.h.

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

- calibrator.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

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

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 122 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label
  Variable label.
- double rangemin
  Minimum value.
- double rangemax
  Maximum value.
- double rangeminabs
  Minimum allowed value.
- double rangemaxabs
  Maximum allowed value.
- unsigned int precision
  Precision digits.

- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 67 of file interface.h.

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

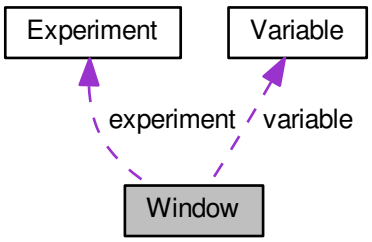
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

- GtkToolButton * button_open

  Open GtkToolButton.
- GtkToolButton * button_save

  Save GtkToolButton.
- GtkToolButton * button_run

  Run GtkToolButton.
- GtkToolButton * button_options

  Options GtkToolButton.
- GtkToolButton * button_help

  Help GtkToolButton.
- GtkToolButton * button_about

  Help GtkToolButton.
- GtkToolButton * button_exit

Exit GtkToolButton.

- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

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

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

  Evaluator program GtkCheckButton.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

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

  Simulator program GtkLabel.
- GtkLabel * label_simulations

  GtkLabel to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkLabel * label_bits

  Bits number GtkLabel.

- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkLabel * label_weight

  Weight GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkComboBoxText * combo_variable

  Variable GtkComboBoxEntry.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkFileChooserButton * button_experiment

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

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

  Minimum GtkSpinButton.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkSpinButton * spin_weight

Weight GtkSpinButton.

- GtkToolbar * bar_buttons

  GtkToolbar to store the main buttons.
- GtkGrid * grid

  Main GtkGrid.
- GtkGrid * grid_algorithm

  GtkGrid to set the algorithm.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkFrame * frame_experiment

  Experiment GtkFrame.
- GdkPixbuf * logo

  Logo GdkPixbuf.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkWindow * window

  Main GtkWindow.
- GtkApplication * application

  Main GtkApplication.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- gulong id_experiment

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

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

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

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

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

  Array of identifiers (gulong) of the button_template signal.
- unsigned int nexperiments

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 138 of file interface.h.

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

- interface.h

Chapter 5

File Documentation

5.1 calibrator.c File Reference

Source file of the calibrator.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "calibrator.h"
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "interface.h"
```

Include dependency graph for calibrator.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG 0

  Macro to debug.
- #define ERROR_TYPE GTK_MESSAGE_ERROR

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

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

  Macro to define the initial input file.

- #define RM "rm"

  Macro to define the shell remove command.

Functions

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

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

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

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

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

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

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

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

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

  Function to create a new Input struct.
- int input_open (char *filename)

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

  Function to free the memory of the input file data.
- void calibrate_input (unsigned int simulation, char *input, GMappedFile *template)

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

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

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

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

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

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

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

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

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

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

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

Function to calibrate with the Monte-Carlo algorithm.

- double calibrate_genetic_objective (Entity *entity)

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_free ()

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

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

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

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

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

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

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

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

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

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

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

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

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

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

- void window_set_variable ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Main function.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- char * current_directory

  Application directory.
- GMutex mutex [1]

  Mutex struct.
- void(* calibrate_step)()

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

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

  Calibration data.
- const xmlChar * template [MAX_NINPUTS]

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

Array of C-strings with variable formats.

- const double precision [NPRECISIONS]

  Array of variable precisions.

- const char * logo []

  Logo pixmap.

- Options options [1]

  Options struct to define the options dialog.

- Running running [1]

  Running struct to define the running dialog.

- Window window [1]

  Window struct to define the main interface window.

5.1.1 Detailed Description

Source file of the calibrator.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2015, all rights reserved.

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 1280 of file calibrator.c.

```
01281 {
01282   unsigned int i, j;
01283   double e;
01284 #if DEBUG
01285   fprintf (stderr, "calibrate_best_sequential: start\n");
01286 #endif
01287   if (calibrate->nsaveds < calibrate->nbest
01288       || value < calibrate->error_best[calibrate->nsaveds - 1])
01289     {
01290       if (calibrate->nsaveds < calibrate->nbest)
01291         ++calibrate->nsaveds;
01292       calibrate->error_best[calibrate->nsaveds - 1] = value;
01293       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01294       for (i = calibrate->nsaveds; --i;)
01295         {
01296           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01297             {
01298               j = calibrate->simulation_best[i];
01299               e = calibrate->error_best[i];
01300               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01301               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
```

```
01302               calibrate->simulation_best[i - 1] = j;
01303               calibrate->error_best[i - 1] = e;
01304             }
01305           else
01306             break;
01307         }
01308     }
01309 #if DEBUG
01310   fprintf (stderr, "calibrate_best_sequential: end\n");
01311 #endif
01312 }
```

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 1235 of file calibrator.c.

```
01236 {
01237   unsigned int i, j;
01238   double e;
01239 #if DEBUG
01240   fprintf (stderr, "calibrate_best_thread: start\n");
01241 #endif
01242   if (calibrate->nsaveds < calibrate->nbest
01243       || value < calibrate->error_best[calibrate->nsaveds - 1])
01244     {
01245       g_mutex_lock (mutex);
01246       if (calibrate->nsaveds < calibrate->nbest)
01247         ++calibrate->nsaveds;
01248       calibrate->error_best[calibrate->nsaveds - 1] = value;
01249       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01250       for (i = calibrate->nsaveds; --i;)
01251         {
01252           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01253             {
01254               j = calibrate->simulation_best[i];
01255               e = calibrate->error_best[i];
01256               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01257               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01258               calibrate->simulation_best[i - 1] = j;
01259               calibrate->error_best[i - 1] = e;
01260             }
01261           else
01262             break;
01263         }
01264       g_mutex_unlock (mutex);
01265     }
01266 #if DEBUG
01267   fprintf (stderr, "calibrate_best_thread: end\n");
01268 #endif
01269 }
```

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

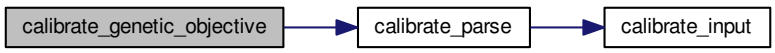
Returns

objective function value.

Definition at line 1589 of file calibrator.c.

```
01590 {
01591   unsigned int j;
01592   double objective;
01593   char buffer[64];
01594 #if DEBUG
01595   fprintf (stderr, "calibrate_genetic_objective: start\n");
01596 #endif
01597   for (j = 0; j < calibrate->nvariables; ++j)
01598     {
01599       calibrate->value[entity->id * calibrate->nvariables + j]
01600         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01601     }
01602   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01603     objective += calibrate_parse (entity->id, j);
01604   g_mutex_lock (mutex);
01605   for (j = 0; j < calibrate->nvariables; ++j)
01606     {
01607       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01608       fprintf (calibrate->file_variables, buffer,
01609                genetic_get_variable (entity, calibrate->
     genetic_variable + j));
01610     }
01611   fprintf (calibrate->file_variables, "%.14le\n", objective);
01612   g_mutex_unlock (mutex);
01613 #if DEBUG
01614   fprintf (stderr, "calibrate_genetic_objective: end\n");
01615 #endif
01616   return objective;
01617 }
```

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 984 of file calibrator.c.

```
00985 {
00986   unsigned int i;
00987   char buffer[32], value[32], *buffer2, *buffer3, *content;
00988   FILE *file;
00989   gsize length;
00990   GRegex *regex;
00991
00992 #if DEBUG
00993   fprintf (stderr, "calibrate_input: start\n");
00994 #endif
00995
00996   // Checking the file
00997   if (!template)
```

```
00998     goto calibrate_input_end;
00999
01000   // Opening template
01001   content = g_mapped_file_get_contents (template);
01002   length = g_mapped_file_get_length (template);
01003 #if DEBUG
01004   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01005            content);
01006 #endif
01007   file = fopen (input, "w");
01008
01009   // Parsing template
01010   for (i = 0; i < calibrate->nvariables; ++i)
01011     {
01012 #if DEBUG
01013       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01014 #endif
01015       snprintf (buffer, 32, "@variable%u@", i + 1);
01016       regex = g_regex_new (buffer, 0, 0, NULL);
01017       if (i == 0)
01018         {
01019           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01020                                              calibrate->label[i], 0, NULL);
01021 #if DEBUG
01022           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01023 #endif
01024         }
01025       else
01026         {
01027           length = strlen (buffer3);
01028           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01029                                              calibrate->label[i], 0, NULL);
01030           g_free (buffer3);
01031         }
01032       g_regex_unref (regex);
01033       length = strlen (buffer2);
01034       snprintf (buffer, 32, "@value%u@", i + 1);
01035       regex = g_regex_new (buffer, 0, 0, NULL);
01036       snprintf (value, 32, format[calibrate->precision[i]],
01037                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01038
01039 #if DEBUG
01040       fprintf (stderr, "calibrate_input: value=%s\n", value);
01041 #endif
01042       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01043                                          0, NULL);
01044       g_free (buffer2);
01045       g_regex_unref (regex);
01046     }
01047
01048   // Saving input file
01049   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01050   g_free (buffer3);
01051   fclose (file);
01052
01053 calibrate_input_end:
01054 #if DEBUG
01055   fprintf (stderr, "calibrate_input: end\n");
01056 #endif
01057   return;
01058 }
```

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 1396 of file calibrator.c.

```
01398 {
01399   unsigned int i, j, k, s[calibrate->nbest];
01400   double e[calibrate->nbest];
01401 #if DEBUG
01402   fprintf (stderr, "calibrate_merge: start\n");
```

```
01403 #endif
01404   i = j = k = 0;
01405   do
01406     {
01407       if (i == calibrate->nsaveds)
01408         {
01409           s[k] = simulation_best[j];
01410           e[k] = error_best[j];
01411           ++j;
01412           ++k;
01413           if (j == nsaveds)
01414             break;
01415         }
01416       else if (j == nsaveds)
01417         {
01418           s[k] = calibrate->simulation_best[i];
01419           e[k] = calibrate->error_best[i];
01420           ++i;
01421           ++k;
01422           if (i == calibrate->nsaveds)
01423             break;
01424         }
01425       else if (calibrate->error_best[i] > error_best[j])
01426         {
01427           s[k] = simulation_best[j];
01428           e[k] = error_best[j];
01429           ++j;
01430           ++k;
01431         }
01432       else
01433         {
01434           s[k] = calibrate->simulation_best[i];
01435           e[k] = calibrate->error_best[i];
01436           ++i;
01437           ++k;
01438         }
01439     }
01440   while (k < calibrate->nbest);
01441   calibrate->nsaveds = k;
01442   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01443   memcpy (calibrate->error_best, e, k * sizeof (double));
01444 #if DEBUG
01445   fprintf (stderr, "calibrate_merge: end\n");
01446 #endif
01447 }
```

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 1071 of file calibrator.c.

```
01072 {
01073   unsigned int i;
01074   double e;
01075   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01076     *buffer3, *buffer4;
01077   FILE *file_result;
01078
01079 #if DEBUG
01080   fprintf (stderr, "calibrate_parse: start\n");
01081   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01082            experiment);
01083 #endif
01084
01085   // Opening input files
01086   for (i = 0; i < calibrate->ninputs; ++i)
01087     {
01088       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
```

```
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01091 #endif
01092       calibrate_input (simulation, &input[i][0],
01093                        calibrate->file[i][experiment]);
01094     }
01095   for (; i < MAX_NINPUTS; ++i)
01096     strcpy (&input[i][0], "");
01097 #if DEBUG
01098   fprintf (stderr, "calibrate_parse: parsing end\n");
01099 #endif
01100
01101   // Performing the simulation
01102   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01103   buffer2 = g_path_get_dirname (calibrate->simulator);
01104   buffer3 = g_path_get_basename (calibrate->simulator);
01105   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01106   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01107             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01108             input[6], input[7], output);
01109   g_free (buffer4);
01110   g_free (buffer3);
01111   g_free (buffer2);
01112 #if DEBUG
01113   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01114 #endif
01115   system (buffer);
01116
01117   // Checking the objective value function
01118   if (calibrate->evaluator)
01119     {
01120       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01121       buffer2 = g_path_get_dirname (calibrate->evaluator);
01122       buffer3 = g_path_get_basename (calibrate->evaluator);
01123       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01124       snprintf (buffer, 512, "\"%s\" %s %s %s",
01125                 buffer4, output, calibrate->experiment[experiment], result);
01126       g_free (buffer4);
01127       g_free (buffer3);
01128       g_free (buffer2);
01129 #if DEBUG
01130       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01131 #endif
01132       system (buffer);
01133       file_result = fopen (result, "r");
01134       e = atof (fgets (buffer, 512, file_result));
01135       fclose (file_result);
01136     }
01137   else
01138     {
01139       strcpy (result, "");
01140       file_result = fopen (output, "r");
01141       e = atof (fgets (buffer, 512, file_result));
01142       fclose (file_result);
01143     }
01144
01145   // Removing files
01146 #if !DEBUG
01147   for (i = 0; i < calibrate->ninputs; ++i)
01148     {
01149       if (calibrate->file[i][0])
01150         {
01151           snprintf (buffer, 512, RM " %s", &input[i][0]);
01152           system (buffer);
01153         }
01154     }
01155   snprintf (buffer, 512, RM " %s %s", output, result);
01156   system (buffer);
01157 #endif
01158
01159 #if DEBUG
01160   fprintf (stderr, "calibrate_parse: end\n");
01161 #endif
01162
01163   // Returning the objective function
01164   return e * calibrate->weight[experiment];
01165 }
```

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 1207 of file calibrator.c.

```
01208 {
01209   unsigned int i;
01210   char buffer[64];
01211 #if DEBUG
01212   fprintf (stderr, "calibrate_save_variables: start\n");
01213 #endif
01214   for (i = 0; i < calibrate->nvariables; ++i)
01215     {
01216       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01217       fprintf (calibrate->file_variables, buffer,
01218                calibrate->value[simulation * calibrate->
      nvariables + i]);
01219     }
01220   fprintf (calibrate->file_variables, "%.14le\n", error);
01221 #if DEBUG
01222   fprintf (stderr, "calibrate_save_variables: end\n");
01223 #endif
01224 }
```

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

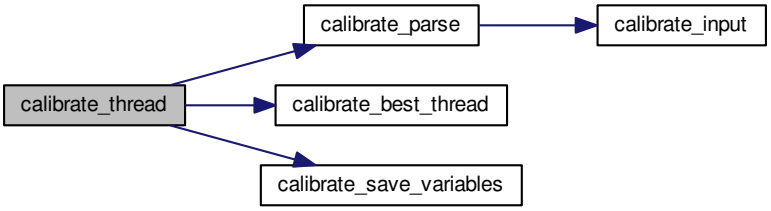
NULL

Definition at line 1322 of file calibrator.c.

```
01323 {
01324   unsigned int i, j, thread;
01325   double e;
01326 #if DEBUG
01327   fprintf (stderr, "calibrate_thread: start\n");
01328 #endif
01329   thread = data->thread;
01330 #if DEBUG
01331   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01332            calibrate->thread[thread], calibrate->thread[thread + 1]);
```

```
01333 #endif
01334   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01335     {
01336       e = 0.;
01337       for (j = 0; j < calibrate->nexperiments; ++j)
01338         e += calibrate_parse (i, j);
01339       calibrate_best_thread (i, e);
01340       g_mutex_lock (mutex);
01341       calibrate_save_variables (i, e);
01342       g_mutex_unlock (mutex);
01343 #if DEBUG
01344       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01345 #endif
01346     }
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_thread: end\n");
01349 #endif
01350   g_thread_exit (NULL);
01351   return NULL;
01352 }
```

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 3763 of file calibrator.c.

```
03764 {
03765 #ifdef G_OS_WIN32
03766   SYSTEM_INFO sysinfo;
03767   GetSystemInfo (&sysinfo);
03768   return sysinfo.dwNumberOfProcessors;
03769 #else
03770   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03771 #endif
03772 }
```

5.1.2.10 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 458 of file calibrator.c.

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

```
00532     {
00533       input->algorithm = ALGORITHM_GENETIC;
00534
00535       // Obtaining population
00536       if (xmlHasProp (node, XML_NPOPULATION))
00537         {
00538           input->nsimulations
00539             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00540           if (error_code || input->nsimulations < 3)
00541             {
00542               show_error (gettext ("Invalid population number"));
00543               return 0;
00544             }
00545         }
00546       else
00547         {
00548           show_error (gettext ("No population number"));
00549           return 0;
00550         }
00551
00552       // Obtaining generations
00553       if (xmlHasProp (node, XML_NGENERATIONS))
00554         {
00555           input->niterations
00556             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00557           if (error_code || !input->niterations)
00558             {
00559               show_error (gettext ("Invalid generation number"));
00560               return 0;
00561             }
00562         }
00563       else
00564         {
00565           show_error (gettext ("No generation number"));
00566           return 0;
00567         }
00568
00569       // Obtaining mutation probability
00570       if (xmlHasProp (node, XML_MUTATION))
00571         {
00572           input->mutation_ratio
00573             = xml_node_get_float (node, XML_MUTATION, &error_code);
00574           if (error_code || input->mutation_ratio < 0.
00575               || input->mutation_ratio >= 1.)
00576             {
00577               show_error (gettext ("Invalid mutation probability"));
00578               return 0;
00579             }
00580         }
00581       else
00582         {
00583           show_error (gettext ("No mutation probability"));
00584           return 0;
00585         }
00586
00587       // Obtaining reproduction probability
00588       if (xmlHasProp (node, XML_REPRODUCTION))
00589         {
00590           input->reproduction_ratio
00591             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00592           if (error_code || input->reproduction_ratio < 0.
00593               || input->reproduction_ratio >= 1.0)
00594             {
00595               show_error (gettext ("Invalid reproduction probability"));
00596               return 0;
00597             }
00598         }
00599       else
00600         {
00601           show_error (gettext ("No reproduction probability"));
00602           return 0;
00603         }
00604
00605       // Obtaining adaptation probability
00606       if (xmlHasProp (node, XML_ADAPTATION))
00607         {
00608           input->adaptation_ratio
00609             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00610           if (error_code || input->adaptation_ratio < 0.
00611               || input->adaptation_ratio >= 1.)
00612             {
00613               show_error (gettext ("Invalid adaptation probability"));
00614               return 0;
00615             }
00616         }
00617       else
00618         {
```

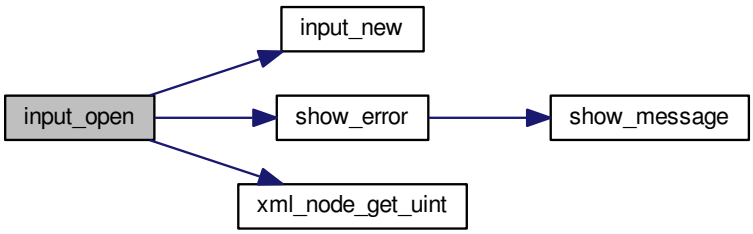
```
00619           show_error (gettext ("No adaptation probability"));
00620           return 0;
00621         }
00622
00623       // Checking survivals
00624       i = input->mutation_ratio * input->nsimulations;
00625       i += input->reproduction_ratio * input->
      nsimulations;
00626       i += input->adaptation_ratio * input->
      nsimulations;
00627       if (i > input->nsimulations - 2)
00628         {
00629           show_error
00630             (gettext
00631              ("No enough survival entities to reproduce the population"));
00632           return 0;
00633         }
00634     }
00635   else
00636     {
00637       show_error (gettext ("Unknown algorithm"));
00638       return 0;
00639     }
00640
00641   if (input->algorithm == ALGORITHM_MONTE_CARLO
00642       || input->algorithm == ALGORITHM_SWEEP)
00643     {
00644
00645       // Obtaining iterations number
00646       input->niterations
00647         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00648       if (error_code == 1)
00649         input->niterations = 1;
00650       else if (error_code)
00651         {
00652           show_error (gettext ("Bad iterations number"));
00653           return 0;
00654         }
00655
00656       // Obtaining best number
00657       if (xmlHasProp (node, XML_NBEST))
00658         {
00659           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00660           if (error_code || !input->nbest)
00661             {
00662               show_error (gettext ("Invalid best number"));
00663               return 0;
00664             }
00665         }
00666       else
00667         input->nbest = 1;
00668
00669       // Obtaining tolerance
00670       if (xmlHasProp (node, XML_TOLERANCE))
00671         {
00672           input->tolerance
00673             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00674           if (error_code || input->tolerance < 0.)
00675             {
00676               show_error (gettext ("Invalid tolerance"));
00677               return 0;
00678             }
00679         }
00680       else
00681         input->tolerance = 0.;
00682     }
00683
00684   // Reading the experimental data
00685   for (child = node->children; child; child = child->next)
00686     {
00687       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00688         break;
00689 #if DEBUG
00690       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00691 #endif
00692       if (xmlHasProp (child, XML_NAME))
00693         {
00694           input->experiment
00695             = g_realloc (input->experiment,
00696                          (1 + input->nexperiments) * sizeof (char *));
00697           input->experiment[input->nexperiments]
00698             = (char *) xmlGetProp (child, XML_NAME);
00699         }
00700       else
00701         {
00702           show_error (gettext ("No experiment file name"));
```

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

```
00790 #if DEBUG
00791       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00792 #endif
00793     }
00794   if (!input->nexperiments)
00795     {
00796       show_error (gettext ("No calibration experiments"));
00797       return 0;
00798     }
00799
00800   // Reading the variables data
00801   for (; child; child = child->next)
00802     {
00803       if (xmlStrcmp (child->name, XML_VARIABLE))
00804         {
00805           show_error (gettext ("Bad XML node"));
00806           return 0;
00807         }
00808       if (xmlHasProp (child, XML_NAME))
00809         {
00810           input->label = g_realloc
00811             (input->label, (1 + input->nvariables) * sizeof (char *));
00812           input->label[input->nvariables]
00813             = (char *) xmlGetProp (child, XML_NAME);
00814         }
00815       else
00816         {
00817           show_error (gettext ("No variable name"));
00818           return 0;
00819         }
00820       if (xmlHasProp (child, XML_MINIMUM))
00821         {
00822           input->rangemin = g_realloc
00823             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00824           input->rangeminabs = g_realloc
00825             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00826           input->rangemin[input->nvariables]
00827             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00828           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00829             {
00830               input->rangeminabs[input->nvariables]
00831                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00832             }
00833           else
00834             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00835         }
00836       else
00837         {
00838           show_error (gettext ("No minimum range"));
00839           return 0;
00840         }
00841       if (xmlHasProp (child, XML_MAXIMUM))
00842         {
00843           input->rangemax = g_realloc
00844             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00845           input->rangemaxabs = g_realloc
00846             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00847           input->rangemax[input->nvariables]
00848             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00849           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00850             input->rangemaxabs[input->nvariables]
00851               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00852           else
00853             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00854         }
00855       else
00856         {
00857           show_error (gettext ("No maximum range"));
00858           return 0;
00859         }
00860       input->precision = g_realloc
00861         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00862       if (xmlHasProp (child, XML_PRECISION))
00863         input->precision[input->nvariables]
00864           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00865       else
00866         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00867       if (input->algorithm == ALGORITHM_SWEEP)
00868         {
00869           if (xmlHasProp (child, XML_NSWEEPS))
00870             {
00871               input->nsweeps = (unsigned int *)
00872                 g_realloc (input->nsweeps,
00873                            (1 + input->nvariables) * sizeof (unsigned int));
```

```
00874               input->nsweeps[input->nvariables]
00875                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00876             }
00877           else
00878             {
00879               show_error (gettext ("No sweeps number"));
00880               return 0;
00881             }
00882 #if DEBUG
00883           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00884                    input->nsweeps[input->nvariables],
      input->nsimulations);
00885 #endif
00886         }
00887       if (input->algorithm == ALGORITHM_GENETIC)
00888         {
00889           // Obtaining bits representing each variable
00890           if (xmlHasProp (child, XML_NBITS))
00891             {
00892               input->nbits = (unsigned int *)
00893                 g_realloc (input->nbits,
00894                            (1 + input->nvariables) * sizeof (unsigned int));
00895               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00896               if (error_code || !i)
00897                 {
00898                   show_error (gettext ("Invalid bit number"));
00899                   return 0;
00900                 }
00901               input->nbits[input->nvariables] = i;
00902             }
00903           else
00904             {
00905               show_error (gettext ("No bits number"));
00906               return 0;
00907             }
00908         }
00909       ++input->nvariables;
00910     }
00911   if (!input->nvariables)
00912     {
00913       show_error (gettext ("No calibration variables"));
00914       return 0;
00915     }
00916
00917   // Getting the working directory
00918   input->directory = g_path_get_dirname (filename);
00919   input->name = g_path_get_basename (filename);
00920
00921   // Closing the XML document
00922   xmlFreeDoc (doc);
00923
00924 #if DEBUG
00925   fprintf (stderr, "input_new: end\n");
00926 #endif
00927
00928   return 1;
00929 }
```

Here is the call graph for this function:



5.1.2.11 void input_save (char * *filename*)

Function to save the input file.

Parameters

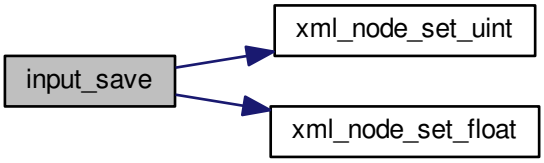
filename	Input file name.

Definition at line 2089 of file calibrator.c.

```
02090 {
02091   unsigned int i, j;
02092   char *buffer;
02093   xmlDoc *doc;
02094   xmlNode *node, *child;
02095   GFile *file, *file2;
02096
02097   // Getting the input file directory
02098   input->name = g_path_get_basename (filename);
02099   input->directory = g_path_get_dirname (filename);
02100   file = g_file_new_for_path (input->directory);
02101
02102   // Opening the input file
02103   doc = xmlNewDoc ((const xmlChar *) "1.0");
02104
02105   // Setting root XML node
02106   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02107   xmlDocSetRootElement (doc, node);
02108
02109   // Adding properties to the root XML node
02110   file2 = g_file_new_for_path (input->simulator);
02111   buffer = g_file_get_relative_path (file, file2);
02112   g_object_unref (file2);
02113   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02114   g_free (buffer);
02115   if (input->evaluator)
02116     {
02117       file2 = g_file_new_for_path (input->evaluator);
02118       buffer = g_file_get_relative_path (file, file2);
02119       g_object_unref (file2);
02120       if (xmlStrlen ((xmlChar *) buffer))
02121         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02122       g_free (buffer);
02123     }
02124   if (input->seed != DEFAULT_RANDOM_SEED)
02125     xml_node_set_uint (node, XML_SEED, input->seed);
02126
02127   // Setting the algorithm
02128   buffer = (char *) g_malloc (64);
02129   switch (input->algorithm)
02130     {
02131     case ALGORITHM_MONTE_CARLO:
02132       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02133       snprintf (buffer, 64, "%u", input->nsimulations);
02134       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02135       snprintf (buffer, 64, "%u", input->niterations);
02136       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02137       snprintf (buffer, 64, "%.3lg", input->tolerance);
02138       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02139       snprintf (buffer, 64, "%u", input->nbest);
02140       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02141       break;
02142     case ALGORITHM_SWEEP:
02143       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02144       snprintf (buffer, 64, "%u", input->niterations);
02145       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02146       snprintf (buffer, 64, "%.3lg", input->tolerance);
02147       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02148       snprintf (buffer, 64, "%u", input->nbest);
02149       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02150       break;
02151     default:
02152       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02153       snprintf (buffer, 64, "%u", input->nsimulations);
02154       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02155       snprintf (buffer, 64, "%u", input->niterations);
02156       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02157       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02158       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02159       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02160       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02161       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02162       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02163       break;
02164     }
02165   g_free (buffer);
02166
02167   // Setting the experimental data
02168   for (i = 0; i < input->nexperiments; ++i)
02169     {
```

```
02170       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02171       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02172       if (input->weight[i] != 1.)
02173         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02174       for (j = 0; j < input->ninputs; ++j)
02175         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02176     }
02177
02178   // Setting the variables data
02179   for (i = 0; i < input->nvariables; ++i)
02180     {
02181       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02182       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02183       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02184       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02185         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02186       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02187       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02188         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02189       if (input->precision[i] != DEFAULT_PRECISION)
02190         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02191       if (input->algorithm == ALGORITHM_SWEEP)
02192         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02193       else if (input->algorithm == ALGORITHM_GENETIC)
02194         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02195     }
02196
02197   // Saving the XML file
02198   xmlSaveFormatFile (filename, doc, 1);
02199
02200   // Freeing memory
02201   xmlFreeDoc (doc);
02202 }
```

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.

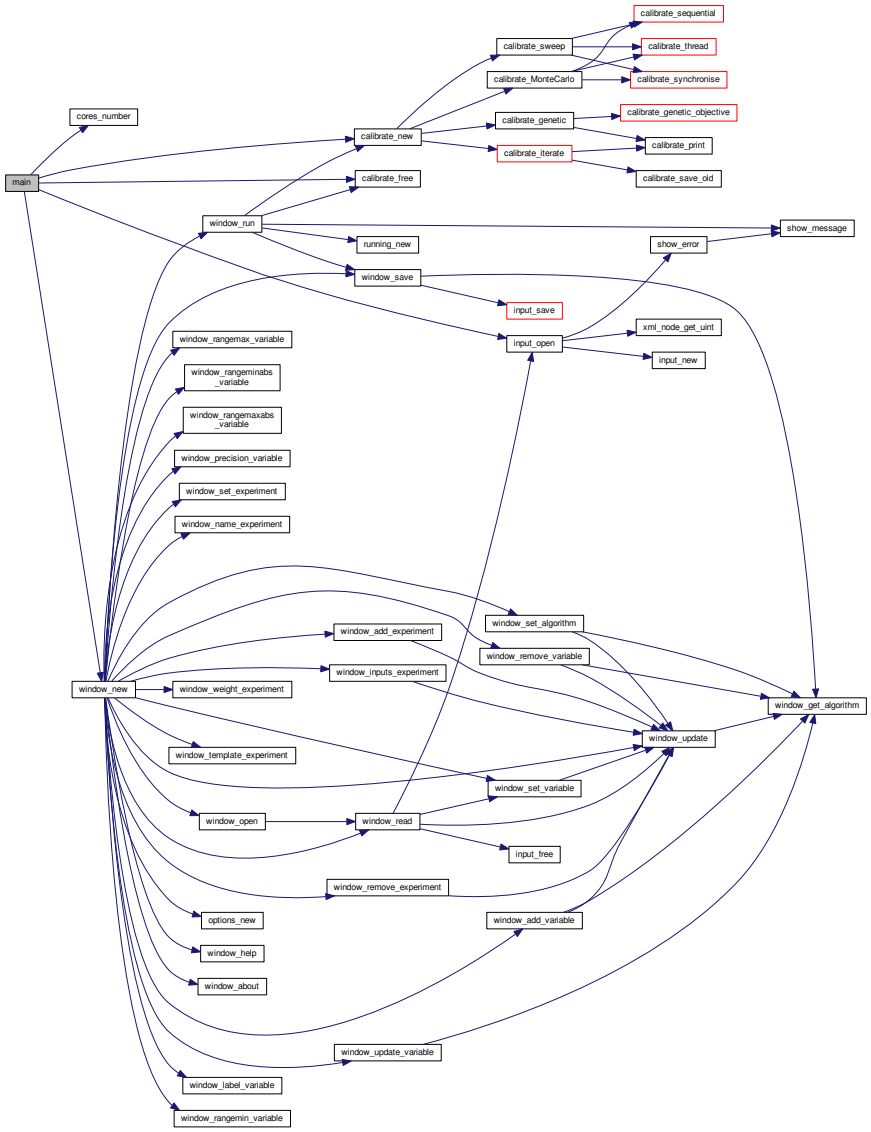
Returns

0 on success, >0 on error.

Definition at line 3784 of file calibrator.c.

```
03785 {
03786 #if HAVE_GTK
03787   int status;
03788 #endif
03789 #if HAVE_MPI
03790   // Starting MPI
03791   MPI_Init (&argn, &argc);
03792   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03793   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03794   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03795 #else
03796   ntasks = 1;
03797 #endif
03798   // Starting pseudo-random numbers generator
03799   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03800   calibrate->seed = DEFAULT_RANDOM_SEED;
03801   // Allowing spaces in the XML data file
03802   xmlKeepBlanksDefault (0);
03803
03804 #if HAVE_GTK
03805
03806   nthreads = cores_number ();
03807   setlocale (LC_ALL, "");
03808   setlocale (LC_NUMERIC, "C");
03809   current_directory = g_get_current_dir ();
03810   bindtextdomain
03811     (PROGRAM_INTERFACE, g_build_filename (current_directory,
     LOCALE_DIR, NULL));
03812   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03813   textdomain (PROGRAM_INTERFACE);
03814   gtk_disable_setlocale ();
03815   window->application = gtk_application_new ("git.jburguete.calibrator",
03816                                              G_APPLICATION_FLAGS_NONE);
03817   g_signal_connect (window->application, "activate", window_new, NULL);
03818   status = g_application_run (G_APPLICATION (window->application), argn, argc);
03819   g_object_unref (window->application);
03820
03821 #else
03822
03823   // Checking syntax
03824   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03825     {
03826       printf ("The syntax is:\ncalibrator [-nthreads x] data_file\n");
03827 #if HAVE_MPI
03828       // Closing MPI
03829       MPI_Finalize ();
03830 #endif
03831       return 1;
03832     }
03833   // Getting threads number
03834   if (argn == 2)
03835     nthreads = cores_number ();
03836   else
03837     nthreads = atoi (argc[2]);
03838   printf ("nthreads=%u\n", nthreads);
03839   // Making calibration
03840   if (input_open (argc[argn - 1]))
03841     calibrate_new ();
03842   // Freeing memory
03843   calibrate_free ();
03844
03845 #endif
03846
03847   // Freeing memory
03848   gsl_rng_free (calibrate->rng);
03849 #if HAVE_MPI
03850   // Closing MPI
03851   MPI_Finalize ();
03852 #endif
03853
03854 #if HAVE_GTK
03855   g_free (current_directory);
03856   return status;
03857 #else
03858   return 0;
03859 #endif
03860 }
```

Here is the call graph for this function:



5.1.2.13 void show_error (char * *msg*)

Function to show a dialog with an error message.

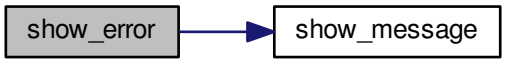
Parameters

msg	Error message.

Definition at line 273 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 243 of file calibrator.c.

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

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2453 of file calibrator.c.

```
02454 {
02455   unsigned int i;
02456   for (i = 0; i < NALGORITHMS; ++i)
02457     if (gtk_toggle_button_get_active
02458         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02459       break;
02460   return i;
02461 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

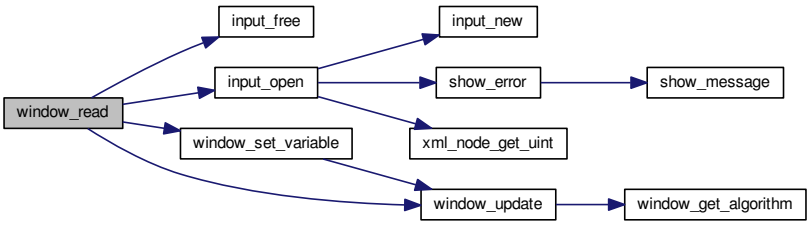
1 on succes, 0 on error.

Definition at line 3169 of file calibrator.c.

```
03170 {
03171   unsigned int i;
03172   char *buffer;
03173 #if DEBUG
03174   fprintf (stderr, "window_read: start\n");
03175 #endif
03176   input_free ();
03177   if (!input_open (filename))
03178     {
03179 #if DEBUG
03180       fprintf (stderr, "window_read: end\n");
03181 #endif
03182       return 0;
03183     }
03184   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03185   puts (buffer);
03186   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03187                                  (window->button_simulator), buffer);
03188   g_free (buffer);
03189   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03190                                 (size_t) input->evaluator);
03191   if (input->evaluator)
03192     {
03193       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03194       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03195                                      (window->button_evaluator), buffer);
03196       g_free (buffer);
03197     }
03198   gtk_toggle_button_set_active
03199     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03200   switch (input->algorithm)
03201     {
03202     case ALGORITHM_MONTE_CARLO:
03203       gtk_spin_button_set_value (window->spin_simulations,
03204                                  (gdouble) input->nsimulations);
03205     case ALGORITHM_SWEEP:
03206       gtk_spin_button_set_value (window->spin_iterations,
03207                                  (gdouble) input->niterations);
03208       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03209       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03210       break;
03211     default:
03212       gtk_spin_button_set_value (window->spin_population,
03213                                  (gdouble) input->nsimulations);
03214       gtk_spin_button_set_value (window->spin_generations,
03215                                  (gdouble) input->niterations);
03216       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03217       gtk_spin_button_set_value (window->spin_reproduction,
03218                                  input->reproduction_ratio);
03219       gtk_spin_button_set_value (window->spin_adaptation,
03220                                  input->adaptation_ratio);
03221     }
03222   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03223   g_signal_handler_block (window->button_experiment,
03224                           window->id_experiment_name);
03225   gtk_combo_box_text_remove_all (window->combo_experiment);
03226   for (i = 0; i < input->nexperiments; ++i)
03227     gtk_combo_box_text_append_text (window->combo_experiment,
03228                                     input->experiment[i]);
03229   g_signal_handler_unblock
03230     (window->button_experiment, window->
      id_experiment_name);
03231   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03232   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03233   g_signal_handler_block (window->combo_variable, window->
```

```
      id_variable);
03234   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03235   gtk_combo_box_text_remove_all (window->combo_variable);
03236   for (i = 0; i < input->nvariables; ++i)
03237     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03238   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03239   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03240   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03241   window_set_variable ();
03242   window_update ();
03243 #if DEBUG
03244   fprintf (stderr, "window_read: end\n");
03245 #endif
03246   return 1;
03247 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

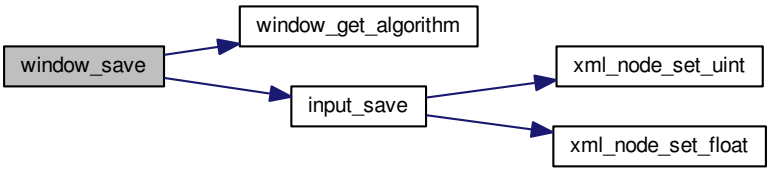
1 on OK, 0 on Cancel.

Definition at line 2272 of file calibrator.c.

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

```
02299           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02300         input->evaluator = gtk_file_chooser_get_filename
02301           (GTK_FILE_CHOOSER (window->button_evaluator));
02302       else
02303         input->evaluator = NULL;
02304
02305       // Setting the algorithm
02306       switch (window_get_algorithm ())
02307         {
02308         case ALGORITHM_MONTE_CARLO:
02309           input->algorithm = ALGORITHM_MONTE_CARLO;
02310           input->nsimulations
02311             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02312           input->niterations
02313             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02314           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02315           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02316           break;
02317         case ALGORITHM_SWEEP:
02318           input->algorithm = ALGORITHM_SWEEP;
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02321           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02322           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02323           break;
02324         default:
02325           input->algorithm = ALGORITHM_GENETIC;
02326           input->nsimulations
02327             = gtk_spin_button_get_value_as_int (window->spin_population);
02328           input->niterations
02329             = gtk_spin_button_get_value_as_int (window->spin_generations);
02330           input->mutation_ratio
02331             = gtk_spin_button_get_value (window->spin_mutation);
02332           input->reproduction_ratio
02333             = gtk_spin_button_get_value (window->spin_reproduction);
02334           input->adaptation_ratio
02335             = gtk_spin_button_get_value (window->spin_adaptation);
02336           break;
02337         }
02338
02339       // Saving the XML file
02340       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02341       input_save (buffer);
02342
02343       // Closing and freeing memory
02344       g_free (buffer);
02345       gtk_widget_destroy (GTK_WIDGET (dlg));
02346 #if DEBUG
02347       fprintf (stderr, "window_save: end\n");
02348 #endif
02349       return 1;
02350     }
02351
02352   // Closing and freeing memory
02353   gtk_widget_destroy (GTK_WIDGET (dlg));
02354 #if DEBUG
02355   fprintf (stderr, "window_save: end\n");
02356 #endif
02357   return 0;
02358 }
```

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 2827 of file calibrator.c.

```
02828 {
02829   unsigned int i, j;
02830   char *buffer;
02831   GFile *file1, *file2;
02832 #if DEBUG
02833   fprintf (stderr, "window_template_experiment: start\n");
02834 #endif
02835   i = (size_t) data;
02836   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02837   file1
02838     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02839   file2 = g_file_new_for_path (input->directory);
02840   buffer = g_file_get_relative_path (file2, file1);
02841   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02842   g_free (buffer);
02843   g_object_unref (file2);
02844   g_object_unref (file1);
02845 #if DEBUG
02846   fprintf (stderr, "window_template_experiment: end\n");
02847 #endif
02848 }
```

5.1.2.19 double xml_node_get_float (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 352 of file calibrator.c.

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

5.1.2.20 int xml_node_get_int (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 290 of file calibrator.c.

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

5.1.2.21 int xml_node_get_uint (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an unsigned integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 321 of file calibrator.c.

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

5.1.2.22 void xml_node_set_float (xmlNode * *node*, const xmlChar * *prop*, double *value*)

Function to set a floating point number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 419 of file calibrator.c.

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

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

Function to set an integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 381 of file calibrator.c.

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

5.1.2.24 void xml_node_set_uint (xmlNode * *node*, const xmlChar * *prop*, unsigned int *value*)

Function to set an unsigned integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Unsigned integer number value.

Definition at line 400 of file calibrator.c.

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

5.1.3 Variable Documentation

5.1.3.1 format

Initial value:

```
= {
  "%.11g", "%.21g", "%.31g", "%.41g", "%.51g", "%.61g", "%.71g", "%.81g",
  "%.91g", "%.101g", "%.111g", "%.121g", "%.131g", "%.141g", "%.151g"
}
```

Array of C-strings with variable formats.

Definition at line 129 of file calibrator.c.

5.1.3.2 precision

Initial value:

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

Array of variable precisions.

Definition at line 134 of file calibrator.c.

5.1.3.3 template

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 124 of file calibrator.c.

5.2 calibrator.c

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012         this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015         this list of conditions and the following disclaimer in the
00016         documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #define _GNU_SOURCE
00037 #include "config.h"
00038 #include <stdio.h>
00039 #include <stdlib.h>
00040 #include <string.h>
00041 #include <math.h>
00042 #include <unistd.h>
00043 #include <locale.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #ifdef G_OS_WIN32
00049 #include <windows.h>
00050 #elif (!__BSD_VISIBLE)
```

```
00051 #include <alloca.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #include "genetic/genetic.h"
00057 #include "calibrator.h"
00058 #if HAVE_GTK
00059 #include <gio/gio.h>
00060 #include <gtk/gtk.h>
00061 #include "interface.h"
00062 #endif
00063
00076 #define DEBUG 0
00077 #if HAVE_GTK
00078 #define ERROR_TYPE GTK_MESSAGE_ERROR
00079 #define INFO_TYPE GTK_MESSAGE_INFO
00080 #else
00081 #define ERROR_TYPE 0
00082 #define INFO_TYPE 0
00083 #endif
00084 #ifdef G_OS_WIN32
00085 #define INPUT_FILE "test-ga-win.xml"
00086 #define RM "del"
00087 #else
00088 #define INPUT_FILE "test-ga.xml"
00089 #define RM "rm"
00090 #endif
00091
00116 int ntasks;
00117 unsigned int nthreads;
00118 char *current_directory;
00119 GMutex mutex[1];
00120 void (*calibrate_step) ();
00121 Input input[1];
00122 Calibrate calibrate[1];
00123
00124 const xmlChar *template[MAX_NINPUTS] = {
00125   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00126   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00127 };
00128
00129 const char *format[NPRECISIONS] = {
00130   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00131   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00132 };
00133
00134 const double precision[NPRECISIONS] = {
00135   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00136   1e-13, 1e-14
00137 };
00138
00139 const char *logo[] = {
00140   "32 32 3 1",
00141   "      c None",
00142   ".     c #0000FF",
00143   "+     c #FF0000",
00144   "                                ",
00145   "                                ",
00146   "                                ",
00147   "     .      .      .      .     ",
00148   "     .      .      .      .     ",
00149   "     .      .      .      .     ",
00150   "     .      .      .      .     ",
00151   "     .      .     +++     .     ",
00152   "     .      .    +++++    .     ",
00153   "     .      .    +++++    .     ",
00154   "     .      .    +++++    .     ",
00155   "    +++     .     +++    +++    ",
00156   "   +++++    .      .    +++++   ",
00157   "   +++++    .      .    +++++   ",
00158   "   +++++    .      .    +++++   ",
00159   "    +++     .      .     +++    ",
00160   "     .      .      .      .     ",
00161   "     .     +++     .      .     ",
00162   "     .    +++++    .      .     ",
00163   "     .    +++++    .      .     ",
00164   "     .    +++++    .      .     ",
00165   "     .     +++     .      .     ",
00166   "     .      .      .      .     ",
00167   "     .      .      .      .     ",
00168   "     .      .      .      .     ",
00169   "     .      .      .      .     ",
00170   "     .      .      .      .     ",
00171   "     .      .      .      .     ",
```

```
00172   "     .      .      .      .     ",
00173   "                                ",
00174   "                                ",
00175   "                                "
00176 };
00177
00178 /*
00179 const char * logo[] = {
00180 "32 32 3 1",
00181 "   c #FFFFFFFFFFFF",
00182 ".  c #00000000FFFF",
00183 "X  c #FFFF00000000",
00184 "                                ",
00185 "                                ",
00186 "                                ",
00187 "     .      .      .      .     ",
00188 "     .      .      .      .     ",
00189 "     .      .      .      .     ",
00190 "     .      .      .      .     ",
00191 "     .      .     XXX     .     ",
00192 "     .      .    XXXXX    .     ",
00193 "     .      .    XXXXX    .     ",
00194 "     .      .    XXXXX    .     ",
00195 "    XXX     .     XXX    XXX    ",
00196 "   XXXXX    .      .    XXXXX   ",
00197 "   XXXXX    .      .    XXXXX   ",
00198 "   XXXXX    .      .    XXXXX   ",
00199 "    XXX     .      .     XXX    ",
00200 "     .      .      .      .     ",
00201 "     .     XXX     .      .     ",
00202 "     .    XXXXX    .      .     ",
00203 "     .    XXXXX    .      .     ",
00204 "     .    XXXXX    .      .     ",
00205 "     .     XXX     .      .     ",
00206 "     .      .      .      .     ",
00207 "     .      .      .      .     ",
00208 "     .      .      .      .     ",
00209 "     .      .      .      .     ",
00210 "     .      .      .      .     ",
00211 "     .      .      .      .     ",
00212 "     .      .      .      .     ",
00213 "                                ",
00214 "                                ",
00215 "                                "};
00216 */
00217
00218 #if HAVE_GTK
00219
00227 Options options[1];
00228 Running running[1];
00229 Window window[1];
00230 #endif
00231
00242 void
00243 show_message (char *title, char *msg, int type)
00244 {
00245 #if HAVE_GTK
00246   GtkMessageDialog *dlg;
00247
00248   // Creating the dialog
00249   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00250     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00251
00252   // Setting the dialog title
00253   gtk_window_set_title (GTK_WINDOW (dlg), title);
00254
00255   // Showing the dialog and waiting response
00256   gtk_dialog_run (GTK_DIALOG (dlg));
00257
00258   // Closing and freeing memory
00259   gtk_widget_destroy (GTK_WIDGET (dlg));
00260
00261 #else
00262   printf ("%s: %s\n", title, msg);
00263 #endif
00264 }
00265
00272 void
00273 show_error (char *msg)
00274 {
00275   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00276 }
00277
00289 int
00290 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00291 {
00292   int i = 0;
```

```
00293   xmlChar *buffer;
00294   buffer = xmlGetProp (node, prop);
00295   if (!buffer)
00296     *error_code = 1;
00297   else
00298     {
00299       if (sscanf ((char *) buffer, "%d", &i) != 1)
00300         *error_code = 2;
00301       else
00302         *error_code = 0;
00303       xmlFree (buffer);
00304     }
00305   return i;
00306 }
00307
00320 unsigned int
00321 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00322 {
00323   unsigned int i = 0;
00324   xmlChar *buffer;
00325   buffer = xmlGetProp (node, prop);
00326   if (!buffer)
00327     *error_code = 1;
00328   else
00329     {
00330       if (sscanf ((char *) buffer, "%u", &i) != 1)
00331         *error_code = 2;
00332       else
00333         *error_code = 0;
00334       xmlFree (buffer);
00335     }
00336   return i;
00337 }
00338
00351 double
00352 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00353 {
00354   double x = 0.;
00355   xmlChar *buffer;
00356   buffer = xmlGetProp (node, prop);
00357   if (!buffer)
00358     *error_code = 1;
00359   else
00360     {
00361       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00362         *error_code = 2;
00363       else
00364         *error_code = 0;
00365       xmlFree (buffer);
00366     }
00367   return x;
00368 }
00369
00380 void
00381 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00382 {
00383   xmlChar buffer[64];
00384   snprintf ((char *) buffer, 64, "%d", value);
00385   xmlSetProp (node, prop, buffer);
00386 }
00387
00399 void
00400 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00401 {
00402   xmlChar buffer[64];
00403   snprintf ((char *) buffer, 64, "%u", value);
00404   xmlSetProp (node, prop, buffer);
00405 }
00406
00418 void
00419 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00420 {
00421   xmlChar buffer[64];
00422   snprintf ((char *) buffer, 64, "%.14lg", value);
00423   xmlSetProp (node, prop, buffer);
00424 }
00425
00430 void
00431 input_new ()
00432 {
00433   unsigned int i;
00434 #if DEBUG
00435   fprintf (stderr, "input_init: start\n");
00436 #endif
00437   input->nvariables = input->nexperiments = input->ninputs = 0;
00438   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
```

```
00439   input->experiment = input->label = NULL;
00440   input->precision = input->nsweeps = input->nbits = NULL;
00441   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00442     = input->weight = NULL;
00443   for (i = 0; i < MAX_NINPUTS; ++i)
00444     input->template[i] = NULL;
00445 #if DEBUG
00446   fprintf (stderr, "input_init: end\n");
00447 #endif
00448 }
00449
00457 int
00458 input_open (char *filename)
00459 {
00460   int error_code;
00461   unsigned int i;
00462   char buffer2[64];
00463   xmlChar *buffer;
00464   xmlDoc *doc;
00465   xmlNode *node, *child;
00466
00467 #if DEBUG
00468   fprintf (stderr, "input_new: start\n");
00469 #endif
00470
00471   // Resetting input data
00472   input_new ();
00473
00474   // Parsing the input file
00475   doc = xmlParseFile (filename);
00476   if (!doc)
00477     {
00478       show_error (gettext ("Unable to parse the input file"));
00479       return 0;
00480     }
00481
00482   // Getting the root node
00483   node = xmlDocGetRootElement (doc);
00484   if (xmlStrcmp (node->name, XML_CALIBRATE))
00485     {
00486       show_error (gettext ("Bad root XML node"));
00487       return 0;
00488     }
00489
00490   // Opening simulator program name
00491   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00492   if (!input->simulator)
00493     {
00494       show_error (gettext ("Bad simulator program"));
00495       return 0;
00496     }
00497
00498   // Opening evaluator program name
00499   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00500
00501   // Obtaining pseudo-random numbers generator seed
00502   if (!xmlHasProp (node, XML_SEED))
00503     input->seed = DEFAULT_RANDOM_SEED;
00504   else
00505     {
00506       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00507       if (error_code)
00508         {
00509           show_error (gettext ("Bad pseudo-random numbers generator seed"));
00510           return 0;
00511         }
00512     }
00513
00514   // Opening algorithm
00515   buffer = xmlGetProp (node, XML_ALGORITHM);
00516   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00517     {
00518       input->algorithm = ALGORITHM_MONTE_CARLO;
00519
00520       // Obtaining simulations number
00521       input->nsimulations
00522         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00523       if (error_code)
00524         {
00525           show_error (gettext ("Bad simulations number"));
00526           return 0;
00527         }
00528     }
00529   else if (!xmlStrcmp (buffer, XML_SWEEP))
00530     input->algorithm = ALGORITHM_SWEEP;
00531   else if (!xmlStrcmp (buffer, XML_GENETIC))
```

```
00532     {
00533       input->algorithm = ALGORITHM_GENETIC;
00534
00535       // Obtaining population
00536       if (xmlHasProp (node, XML_NPOPULATION))
00537         {
00538           input->nsimulations
00539             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00540           if (error_code || input->nsimulations < 3)
00541             {
00542               show_error (gettext ("Invalid population number"));
00543               return 0;
00544             }
00545         }
00546       else
00547         {
00548           show_error (gettext ("No population number"));
00549           return 0;
00550         }
00551
00552       // Obtaining generations
00553       if (xmlHasProp (node, XML_NGENERATIONS))
00554         {
00555           input->niterations
00556             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00557           if (error_code || !input->niterations)
00558             {
00559               show_error (gettext ("Invalid generation number"));
00560               return 0;
00561             }
00562         }
00563       else
00564         {
00565           show_error (gettext ("No generation number"));
00566           return 0;
00567         }
00568
00569       // Obtaining mutation probability
00570       if (xmlHasProp (node, XML_MUTATION))
00571         {
00572           input->mutation_ratio
00573             = xml_node_get_float (node, XML_MUTATION, &error_code);
00574           if (error_code || input->mutation_ratio < 0.
00575               || input->mutation_ratio >= 1.)
00576             {
00577               show_error (gettext ("Invalid mutation probability"));
00578               return 0;
00579             }
00580         }
00581       else
00582         {
00583           show_error (gettext ("No mutation probability"));
00584           return 0;
00585         }
00586
00587       // Obtaining reproduction probability
00588       if (xmlHasProp (node, XML_REPRODUCTION))
00589         {
00590           input->reproduction_ratio
00591             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00592           if (error_code || input->reproduction_ratio < 0.
00593               || input->reproduction_ratio >= 1.0)
00594             {
00595               show_error (gettext ("Invalid reproduction probability"));
00596               return 0;
00597             }
00598         }
00599       else
00600         {
00601           show_error (gettext ("No reproduction probability"));
00602           return 0;
00603         }
00604
00605       // Obtaining adaptation probability
00606       if (xmlHasProp (node, XML_ADAPTATION))
00607         {
00608           input->adaptation_ratio
00609             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00610           if (error_code || input->adaptation_ratio < 0.
00611               || input->adaptation_ratio >= 1.)
00612             {
00613               show_error (gettext ("Invalid adaptation probability"));
00614               return 0;
00615             }
00616         }
00617       else
00618         {
```

```
00619           show_error (gettext ("No adaptation probability"));
00620           return 0;
00621         }
00622
00623       // Checking survivals
00624       i = input->mutation_ratio * input->nsimulations;
00625       i += input->reproduction_ratio * input->nsimulations;
00626       i += input->adaptation_ratio * input->nsimulations;
00627       if (i > input->nsimulations - 2)
00628         {
00629           show_error
00630             (gettext
00631              ("No enough survival entities to reproduce the population"));
00632           return 0;
00633         }
00634     }
00635   else
00636     {
00637       show_error (gettext ("Unknown algorithm"));
00638       return 0;
00639     }
00640
00641   if (input->algorithm == ALGORITHM_MONTE_CARLO
00642       || input->algorithm == ALGORITHM_SWEEP)
00643     {
00644
00645       // Obtaining iterations number
00646       input->niterations
00647         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00648       if (error_code == 1)
00649         input->niterations = 1;
00650       else if (error_code)
00651         {
00652           show_error (gettext ("Bad iterations number"));
00653           return 0;
00654         }
00655
00656       // Obtaining best number
00657       if (xmlHasProp (node, XML_NBEST))
00658         {
00659           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00660           if (error_code || !input->nbest)
00661             {
00662               show_error (gettext ("Invalid best number"));
00663               return 0;
00664             }
00665         }
00666       else
00667         input->nbest = 1;
00668
00669       // Obtaining tolerance
00670       if (xmlHasProp (node, XML_TOLERANCE))
00671         {
00672           input->tolerance
00673             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00674           if (error_code || input->tolerance < 0.)
00675             {
00676               show_error (gettext ("Invalid tolerance"));
00677               return 0;
00678             }
00679         }
00680       else
00681         input->tolerance = 0.;
00682     }
00683
00684   // Reading the experimental data
00685   for (child = node->children; child; child = child->next)
00686     {
00687       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00688         break;
00689 #if DEBUG
00690       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00691 #endif
00692       if (xmlHasProp (child, XML_NAME))
00693         {
00694           input->experiment
00695             = g_realloc (input->experiment,
00696                          (1 + input->nexperiments) * sizeof (char *));
00697           input->experiment[input->nexperiments]
00698             = (char *) xmlGetProp (child, XML_NAME);
00699         }
00700       else
00701         {
00702           show_error (gettext ("No experiment file name"));
00703           return 0;
00704         }
```

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

```
00792 #endif
00793     }
00794   if (!input->nexperiments)
00795     {
00796       show_error (gettext ("No calibration experiments"));
00797       return 0;
00798     }
00799
00800   // Reading the variables data
00801   for (; child; child = child->next)
00802     {
00803       if (xmlStrcmp (child->name, XML_VARIABLE))
00804         {
00805           show_error (gettext ("Bad XML node"));
00806           return 0;
00807         }
00808       if (xmlHasProp (child, XML_NAME))
00809         {
00810           input->label = g_realloc
00811             (input->label, (1 + input->nvariables) * sizeof (char *));
00812           input->label[input->nvariables]
00813             = (char *) xmlGetProp (child, XML_NAME);
00814         }
00815       else
00816         {
00817           show_error (gettext ("No variable name"));
00818           return 0;
00819         }
00820       if (xmlHasProp (child, XML_MINIMUM))
00821         {
00822           input->rangemin = g_realloc
00823             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00824           input->rangeminabs = g_realloc
00825             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00826           input->rangemin[input->nvariables]
00827             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00828           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00829             {
00830               input->rangeminabs[input->nvariables]
00831                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00832             }
00833           else
00834             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00835         }
00836       else
00837         {
00838           show_error (gettext ("No minimum range"));
00839           return 0;
00840         }
00841       if (xmlHasProp (child, XML_MAXIMUM))
00842         {
00843           input->rangemax = g_realloc
00844             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00845           input->rangemaxabs = g_realloc
00846             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00847           input->rangemax[input->nvariables]
00848             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00849           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00850             input->rangemaxabs[input->nvariables]
00851               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00852           else
00853             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00854         }
00855       else
00856         {
00857           show_error (gettext ("No maximum range"));
00858           return 0;
00859         }
00860       input->precision = g_realloc
00861         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00862       if (xmlHasProp (child, XML_PRECISION))
00863         input->precision[input->nvariables]
00864           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00865       else
00866         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00867       if (input->algorithm == ALGORITHM_SWEEP)
00868         {
00869           if (xmlHasProp (child, XML_NSWEEPS))
00870             {
00871               input->nsweeps = (unsigned int *)
00872                 g_realloc (input->nsweeps,
00873                            (1 + input->nvariables) * sizeof (unsigned int));
00874               input->nsweeps[input->nvariables]
00875                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
```

```
00876             }
00877           else
00878             {
00879               show_error (gettext ("No sweeps number"));
00880               return 0;
00881             }
00882 #if DEBUG
00883           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00884                    input->nsweeps[input->nvariables], input->
      nsimulations);
00885 #endif
00886         }
00887       if (input->algorithm == ALGORITHM_GENETIC)
00888         {
00889           // Obtaining bits representing each variable
00890           if (xmlHasProp (child, XML_NBITS))
00891             {
00892               input->nbits = (unsigned int *)
00893                 g_realloc (input->nbits,
00894                            (1 + input->nvariables) * sizeof (unsigned int));
00895               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00896               if (error_code || !i)
00897                 {
00898                   show_error (gettext ("Invalid bit number"));
00899                   return 0;
00900                 }
00901               input->nbits[input->nvariables] = i;
00902             }
00903           else
00904             {
00905               show_error (gettext ("No bits number"));
00906               return 0;
00907             }
00908         }
00909       ++input->nvariables;
00910     }
00911   if (!input->nvariables)
00912     {
00913       show_error (gettext ("No calibration variables"));
00914       return 0;
00915     }
00916
00917   // Getting the working directory
00918   input->directory = g_path_get_dirname (filename);
00919   input->name = g_path_get_basename (filename);
00920
00921   // Closing the XML document
00922   xmlFreeDoc (doc);
00923
00924 #if DEBUG
00925   fprintf (stderr, "input_new: end\n");
00926 #endif
00927
00928   return 1;
00929 }
00930
00935 void
00936 input_free ()
00937 {
00938   unsigned int i, j;
00939 #if DEBUG
00940   fprintf (stderr, "input_free: start\n");
00941 #endif
00942   g_free (input->name);
00943   g_free (input->directory);
00944   for (i = 0; i < input->nexperiments; ++i)
00945     {
00946       xmlFree (input->experiment[i]);
00947       for (j = 0; j < input->ninputs; ++j)
00948         xmlFree (input->template[j][i]);
00949     }
00950   g_free (input->experiment);
00951   for (i = 0; i < input->ninputs; ++i)
00952     g_free (input->template[i]);
00953   for (i = 0; i < input->nvariables; ++i)
00954     xmlFree (input->label[i]);
00955   g_free (input->label);
00956   g_free (input->precision);
00957   g_free (input->rangemin);
00958   g_free (input->rangemax);
00959   g_free (input->rangeminabs);
00960   g_free (input->rangemaxabs);
00961   g_free (input->weight);
00962   g_free (input->nsweeps);
00963   g_free (input->nbits);
00964   xmlFree (input->evaluator);
00965   xmlFree (input->simulator);
```

```
00966   input->nexperiments = input->ninputs = input->nvariables = 0;
00967 #if DEBUG
00968   fprintf (stderr, "input_free: end\n");
00969 #endif
00970 }
00971
00983 void
00984 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
00985 {
00986   unsigned int i;
00987   char buffer[32], value[32], *buffer2, *buffer3, *content;
00988   FILE *file;
00989   gsize length;
00990   GRegex *regex;
00991
00992 #if DEBUG
00993   fprintf (stderr, "calibrate_input: start\n");
00994 #endif
00995
00996   // Checking the file
00997   if (!template)
00998     goto calibrate_input_end;
00999
01000   // Opening template
01001   content = g_mapped_file_get_contents (template);
01002   length = g_mapped_file_get_length (template);
01003 #if DEBUG
01004   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01005            content);
01006 #endif
01007   file = fopen (input, "w");
01008
01009   // Parsing template
01010   for (i = 0; i < calibrate->nvariables; ++i)
01011     {
01012 #if DEBUG
01013       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01014 #endif
01015       snprintf (buffer, 32, "@variable%u@", i + 1);
01016       regex = g_regex_new (buffer, 0, 0, NULL);
01017       if (i == 0)
01018         {
01019           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01020                                              calibrate->label[i], 0, NULL);
01021 #if DEBUG
01022           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01023 #endif
01024         }
01025       else
01026         {
01027           length = strlen (buffer3);
01028           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01029                                              calibrate->label[i], 0, NULL);
01030           g_free (buffer3);
01031         }
01032       g_regex_unref (regex);
01033       length = strlen (buffer2);
01034       snprintf (buffer, 32, "@value%u@", i + 1);
01035       regex = g_regex_new (buffer, 0, 0, NULL);
01036       snprintf (value, 32, format[calibrate->precision[i]],
01037                 calibrate->value[simulation * calibrate->nvariables + i]);
01038
01039 #if DEBUG
01040       fprintf (stderr, "calibrate_input: value=%s\n", value);
01041 #endif
01042       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01043                                          0, NULL);
01044       g_free (buffer2);
01045       g_regex_unref (regex);
01046     }
01047
01048   // Saving input file
01049   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01050   g_free (buffer3);
01051   fclose (file);
01052
01053 calibrate_input_end:
01054 #if DEBUG
01055   fprintf (stderr, "calibrate_input: end\n");
01056 #endif
01057   return;
01058 }
01059
01070 double
01071 calibrate_parse (unsigned int simulation, unsigned int experiment)
01072 {
01073   unsigned int i;
```

```
01074   double e;
01075   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01076     *buffer3, *buffer4;
01077   FILE *file_result;
01078
01079 #if DEBUG
01080   fprintf (stderr, "calibrate_parse: start\n");
01081   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01082            experiment);
01083 #endif
01084
01085   // Opening input files
01086   for (i = 0; i < calibrate->ninputs; ++i)
01087     {
01088       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01091 #endif
01092       calibrate_input (simulation, &input[i][0],
01093                        calibrate->file[i][experiment]);
01094     }
01095   for (; i < MAX_NINPUTS; ++i)
01096     strcpy (&input[i][0], "");
01097 #if DEBUG
01098   fprintf (stderr, "calibrate_parse: parsing end\n");
01099 #endif
01100
01101   // Performing the simulation
01102   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01103   buffer2 = g_path_get_dirname (calibrate->simulator);
01104   buffer3 = g_path_get_basename (calibrate->simulator);
01105   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01106   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01107             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01108             input[6], input[7], output);
01109   g_free (buffer4);
01110   g_free (buffer3);
01111   g_free (buffer2);
01112 #if DEBUG
01113   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01114 #endif
01115   system (buffer);
01116
01117   // Checking the objective value function
01118   if (calibrate->evaluator)
01119     {
01120       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01121       buffer2 = g_path_get_dirname (calibrate->evaluator);
01122       buffer3 = g_path_get_basename (calibrate->evaluator);
01123       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01124       snprintf (buffer, 512, "\"%s\" %s %s %s",
01125                 buffer4, output, calibrate->experiment[experiment], result);
01126       g_free (buffer4);
01127       g_free (buffer3);
01128       g_free (buffer2);
01129 #if DEBUG
01130       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01131 #endif
01132       system (buffer);
01133       file_result = fopen (result, "r");
01134       e = atof (fgets (buffer, 512, file_result));
01135       fclose (file_result);
01136     }
01137   else
01138     {
01139       strcpy (result, "");
01140       file_result = fopen (output, "r");
01141       e = atof (fgets (buffer, 512, file_result));
01142       fclose (file_result);
01143     }
01144
01145   // Removing files
01146 #if !DEBUG
01147   for (i = 0; i < calibrate->ninputs; ++i)
01148     {
01149       if (calibrate->file[i][0])
01150         {
01151           snprintf (buffer, 512, RM " %s", &input[i][0]);
01152           system (buffer);
01153         }
01154     }
01155   snprintf (buffer, 512, RM " %s %s", output, result);
01156   system (buffer);
01157 #endif
01158
01159 #if DEBUG
01160   fprintf (stderr, "calibrate_parse: end\n");
```

```
01161 #endif
01162
01163   // Returning the objective function
01164   return e * calibrate->weight[experiment];
01165 }
01166
01171 void
01172 calibrate_print ()
01173 {
01174   unsigned int i;
01175   char buffer[512];
01176 #if HAVE_MPI
01177   if (!calibrate->mpi_rank)
01178     {
01179 #endif
01180       printf ("THE BEST IS\n");
01181       fprintf (calibrate->file_result, "THE BEST IS\n");
01182       printf ("error=%.15le\n", calibrate->error_old[0]);
01183       fprintf (calibrate->file_result, "error=%.15le\n",
01184                calibrate->error_old[0]);
01185       for (i = 0; i < calibrate->nvariables; ++i)
01186         {
01187           snprintf (buffer, 512, "%s=%s\n",
01188                     calibrate->label[i], format[calibrate->precision[i]]);
01189           printf (buffer, calibrate->value_old[i]);
01190           fprintf (calibrate->file_result, buffer, calibrate->
      value_old[i]);
01191         }
01192       fflush (calibrate->file_result);
01193 #if HAVE_MPI
01194     }
01195 #endif
01196 }
01197
01206 void
01207 calibrate_save_variables (unsigned int simulation, double error)
01208 {
01209   unsigned int i;
01210   char buffer[64];
01211 #if DEBUG
01212   fprintf (stderr, "calibrate_save_variables: start\n");
01213 #endif
01214   for (i = 0; i < calibrate->nvariables; ++i)
01215     {
01216       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01217       fprintf (calibrate->file_variables, buffer,
01218                calibrate->value[simulation * calibrate->nvariables + i]);
01219     }
01220   fprintf (calibrate->file_variables, "%.14le\n", error);
01221 #if DEBUG
01222   fprintf (stderr, "calibrate_save_variables: end\n");
01223 #endif
01224 }
01225
01234 void
01235 calibrate_best_thread (unsigned int simulation, double value)
01236 {
01237   unsigned int i, j;
01238   double e;
01239 #if DEBUG
01240   fprintf (stderr, "calibrate_best_thread: start\n");
01241 #endif
01242   if (calibrate->nsaveds < calibrate->nbest
01243       || value < calibrate->error_best[calibrate->nsaveds - 1])
01244     {
01245       g_mutex_lock (mutex);
01246       if (calibrate->nsaveds < calibrate->nbest)
01247         ++calibrate->nsaveds;
01248       calibrate->error_best[calibrate->nsaveds - 1] = value;
01249       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01250       for (i = calibrate->nsaveds; --i;)
01251         {
01252           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01253             {
01254               j = calibrate->simulation_best[i];
01255               e = calibrate->error_best[i];
01256               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01257               calibrate->error_best[i] = calibrate->error_best[i - 1];
01258               calibrate->simulation_best[i - 1] = j;
01259               calibrate->error_best[i - 1] = e;
01260             }
01261           else
01262             break;
01263         }
01264       g_mutex_unlock (mutex);
01265     }
```

```
01266 #if DEBUG
01267   fprintf (stderr, "calibrate_best_thread: end\n");
01268 #endif
01269 }
01270
01279 void
01280 calibrate_best_sequential (unsigned int simulation, double value)
01281 {
01282   unsigned int i, j;
01283   double e;
01284 #if DEBUG
01285   fprintf (stderr, "calibrate_best_sequential: start\n");
01286 #endif
01287   if (calibrate->nsaveds < calibrate->nbest
01288       || value < calibrate->error_best[calibrate->nsaveds - 1])
01289     {
01290       if (calibrate->nsaveds < calibrate->nbest)
01291         ++calibrate->nsaveds;
01292       calibrate->error_best[calibrate->nsaveds - 1] = value;
01293       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01294       for (i = calibrate->nsaveds; --i;)
01295         {
01296           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01297             {
01298               j = calibrate->simulation_best[i];
01299               e = calibrate->error_best[i];
01300               calibrate->simulation_best[i] = calibrate->
    simulation_best[i - 1];
01301               calibrate->error_best[i] = calibrate->error_best[i - 1];
01302               calibrate->simulation_best[i - 1] = j;
01303               calibrate->error_best[i - 1] = e;
01304             }
01305           else
01306             break;
01307         }
01308     }
01309 #if DEBUG
01310   fprintf (stderr, "calibrate_best_sequential: end\n");
01311 #endif
01312 }
01313
01321 void *
01322 calibrate_thread (ParallelData * data)
01323 {
01324   unsigned int i, j, thread;
01325   double e;
01326 #if DEBUG
01327   fprintf (stderr, "calibrate_thread: start\n");
01328 #endif
01329   thread = data->thread;
01330 #if DEBUG
01331   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01332            calibrate->thread[thread], calibrate->thread[thread + 1]);
01333 #endif
01334   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01335     {
01336       e = 0.;
01337       for (j = 0; j < calibrate->nexperiments; ++j)
01338         e += calibrate_parse (i, j);
01339       calibrate_best_thread (i, e);
01340       g_mutex_lock (mutex);
01341       calibrate_save_variables (i, e);
01342       g_mutex_unlock (mutex);
01343 #if DEBUG
01344       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01345 #endif
01346     }
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_thread: end\n");
01349 #endif
01350   g_thread_exit (NULL);
01351   return NULL;
01352 }
01353
01358 void
01359 calibrate_sequential ()
01360 {
01361   unsigned int i, j;
01362   double e;
01363 #if DEBUG
01364   fprintf (stderr, "calibrate_sequential: start\n");
01365   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01366            calibrate->nstart, calibrate->nend);
01367 #endif
01368   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01369     {
01370       e = 0.;
```

```
01371       for (j = 0; j < calibrate->nexperiments; ++j)
01372         e += calibrate_parse (i, j);
01373       calibrate_best_sequential (i, e);
01374       calibrate_save_variables (i, e);
01375 #if DEBUG
01376       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01377 #endif
01378     }
01379 #if DEBUG
01380   fprintf (stderr, "calibrate_sequential: end\n");
01381 #endif
01382 }
01383
01395 void
01396 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01397                  double *error_best)
01398 {
01399   unsigned int i, j, k, s[calibrate->nbest];
01400   double e[calibrate->nbest];
01401 #if DEBUG
01402   fprintf (stderr, "calibrate_merge: start\n");
01403 #endif
01404   i = j = k = 0;
01405   do
01406     {
01407       if (i == calibrate->nsaveds)
01408         {
01409           s[k] = simulation_best[j];
01410           e[k] = error_best[j];
01411           ++j;
01412           ++k;
01413           if (j == nsaveds)
01414             break;
01415         }
01416       else if (j == nsaveds)
01417         {
01418           s[k] = calibrate->simulation_best[i];
01419           e[k] = calibrate->error_best[i];
01420           ++i;
01421           ++k;
01422           if (i == calibrate->nsaveds)
01423             break;
01424         }
01425       else if (calibrate->error_best[i] > error_best[j])
01426         {
01427           s[k] = simulation_best[j];
01428           e[k] = error_best[j];
01429           ++j;
01430           ++k;
01431         }
01432       else
01433         {
01434           s[k] = calibrate->simulation_best[i];
01435           e[k] = calibrate->error_best[i];
01436           ++i;
01437           ++k;
01438         }
01439     }
01440   while (k < calibrate->nbest);
01441   calibrate->nsaveds = k;
01442   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01443   memcpy (calibrate->error_best, e, k * sizeof (double));
01444 #if DEBUG
01445   fprintf (stderr, "calibrate_merge: end\n");
01446 #endif
01447 }
01448
01453 #if HAVE_MPI
01454 void
01455 calibrate_synchronise ()
01456 {
01457   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01458   double error_best[calibrate->nbest];
01459   MPI_Status mpi_stat;
01460 #if DEBUG
01461   fprintf (stderr, "calibrate_synchronise: start\n");
01462 #endif
01463   if (calibrate->mpi_rank == 0)
01464     {
01465       for (i = 1; i < ntasks; ++i)
01466         {
01467           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01468           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01469                     MPI_COMM_WORLD, &mpi_stat);
01470           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01471                     MPI_COMM_WORLD, &mpi_stat);
01472           calibrate_merge (nsaveds, simulation_best, error_best);
```

```
01473         }
01474     }
01475   else
01476     {
01477       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01478       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01479                 MPI_COMM_WORLD);
01480       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01481                 MPI_COMM_WORLD);
01482     }
01483 #if DEBUG
01484   fprintf (stderr, "calibrate_synchronise: end\n");
01485 #endif
01486 }
01487 #endif
01488
01493 void
01494 calibrate_sweep ()
01495 {
01496   unsigned int i, j, k, l;
01497   double e;
01498   GThread *thread[nthreads];
01499   ParallelData data[nthreads];
01500 #if DEBUG
01501   fprintf (stderr, "calibrate_sweep: start\n");
01502 #endif
01503   for (i = 0; i < calibrate->nsimulations; ++i)
01504     {
01505       k = i;
01506       for (j = 0; j < calibrate->nvariables; ++j)
01507         {
01508           l = k % calibrate->nsweeps[j];
01509           k /= calibrate->nsweeps[j];
01510           e = calibrate->rangemin[j];
01511           if (calibrate->nsweeps[j] > 1)
01512             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01513               / (calibrate->nsweeps[j] - 1);
01514           calibrate->value[i * calibrate->nvariables + j] = e;
01515         }
01516     }
01517   calibrate->nsaveds = 0;
01518   if (nthreads <= 1)
01519     calibrate_sequential ();
01520   else
01521     {
01522       for (i = 0; i < nthreads; ++i)
01523         {
01524           data[i].thread = i;
01525           thread[i]
01526             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01527         }
01528       for (i = 0; i < nthreads; ++i)
01529         g_thread_join (thread[i]);
01530     }
01531 #if HAVE_MPI
01532   // Communicating tasks results
01533   calibrate_synchronise ();
01534 #endif
01535 #if DEBUG
01536   fprintf (stderr, "calibrate_sweep: end\n");
01537 #endif
01538 }
01539
01544 void
01545 calibrate_MonteCarlo ()
01546 {
01547   unsigned int i, j;
01548   GThread *thread[nthreads];
01549   ParallelData data[nthreads];
01550 #if DEBUG
01551   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01552 #endif
01553   for (i = 0; i < calibrate->nsimulations; ++i)
01554     for (j = 0; j < calibrate->nvariables; ++j)
01555       calibrate->value[i * calibrate->nvariables + j]
01556         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01557         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01558   calibrate->nsaveds = 0;
01559   if (nthreads <= 1)
01560     calibrate_sequential ();
01561   else
01562     {
01563       for (i = 0; i < nthreads; ++i)
01564         {
01565           data[i].thread = i;
01566           thread[i]
01567             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
```

```
01568         }
01569       for (i = 0; i < nthreads; ++i)
01570         g_thread_join (thread[i]);
01571     }
01572 #if HAVE_MPI
01573   // Communicating tasks results
01574   calibrate_synchronise ();
01575 #endif
01576 #if DEBUG
01577   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01578 #endif
01579 }
01580
01588 double
01589 calibrate_genetic_objective (Entity * entity)
01590 {
01591   unsigned int j;
01592   double objective;
01593   char buffer[64];
01594 #if DEBUG
01595   fprintf (stderr, "calibrate_genetic_objective: start\n");
01596 #endif
01597   for (j = 0; j < calibrate->nvariables; ++j)
01598     {
01599       calibrate->value[entity->id * calibrate->nvariables + j]
01600         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01601     }
01602   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01603     objective += calibrate_parse (entity->id, j);
01604   g_mutex_lock (mutex);
01605   for (j = 0; j < calibrate->nvariables; ++j)
01606     {
01607       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01608       fprintf (calibrate->file_variables, buffer,
01609                genetic_get_variable (entity, calibrate->genetic_variable + j));
01610     }
01611   fprintf (calibrate->file_variables, "%.14le\n", objective);
01612   g_mutex_unlock (mutex);
01613 #if DEBUG
01614   fprintf (stderr, "calibrate_genetic_objective: end\n");
01615 #endif
01616   return objective;
01617 }
01618
01623 void
01624 calibrate_genetic ()
01625 {
01626   char *best_genome;
01627   double best_objective, *best_variable;
01628 #if DEBUG
01629   fprintf (stderr, "calibrate_genetic: start\n");
01630   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
01631            nthreads);
01632   fprintf (stderr,
01633            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
01634            calibrate->nvariables, calibrate->nsimulations,
01635            calibrate->niterations);
01636   fprintf (stderr,
01637            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01638            calibrate->mutation_ratio, calibrate->
     reproduction_ratio,
01639            calibrate->adaptation_ratio);
01640 #endif
01641   genetic_algorithm_default (calibrate->nvariables,
01642                              calibrate->genetic_variable,
01643                              calibrate->nsimulations,
01644                              calibrate->niterations,
01645                              calibrate->mutation_ratio,
01646                              calibrate->reproduction_ratio,
01647                              calibrate->adaptation_ratio,
01648                              &calibrate_genetic_objective,
01649                              &best_genome, &best_variable, &best_objective);
01650 #if DEBUG
01651   fprintf (stderr, "calibrate_genetic: the best\n");
01652 #endif
01653   calibrate->error_old = (double *) g_malloc (sizeof (double));
01654   calibrate->value_old
01655     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
01656   calibrate->error_old[0] = best_objective;
01657   memcpy (calibrate->value_old, best_variable,
01658           calibrate->nvariables * sizeof (double));
01659   g_free (best_genome);
01660   g_free (best_variable);
01661   calibrate_print ();
01662 #if DEBUG
01663   fprintf (stderr, "calibrate_genetic: end\n");
01664 #endif
```

```
01665 }
01666
01671 void
01672 calibrate_save_old ()
01673 {
01674   unsigned int i, j;
01675 #if DEBUG
01676   fprintf (stderr, "calibrate_save_old: start\n");
01677 #endif
01678   memcpy (calibrate->error_old, calibrate->error_best,
01679           calibrate->nbest * sizeof (double));
01680   for (i = 0; i < calibrate->nbest; ++i)
01681     {
01682       j = calibrate->simulation_best[i];
01683       memcpy (calibrate->value_old + i * calibrate->nvariables,
01684               calibrate->value + j * calibrate->nvariables,
01685               calibrate->nvariables * sizeof (double));
01686     }
01687 #if DEBUG
01688   for (i = 0; i < calibrate->nvariables; ++i)
01689     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
01690              i, calibrate->value_old[i]);
01691   fprintf (stderr, "calibrate_save_old: end\n");
01692 #endif
01693 }
01694
01700 void
01701 calibrate_merge_old ()
01702 {
01703   unsigned int i, j, k;
01704   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
01705     *enew, *eold;
01706 #if DEBUG
01707   fprintf (stderr, "calibrate_merge_old: start\n");
01708 #endif
01709   enew = calibrate->error_best;
01710   eold = calibrate->error_old;
01711   i = j = k = 0;
01712   do
01713     {
01714       if (*enew < *eold)
01715         {
01716           memcpy (v + k * calibrate->nvariables,
01717                   calibrate->value
01718                   + calibrate->simulation_best[i] * calibrate->
      nvariables,
01719                   calibrate->nvariables * sizeof (double));
01720           e[k] = *enew;
01721           ++k;
01722           ++enew;
01723           ++i;
01724         }
01725       else
01726         {
01727           memcpy (v + k * calibrate->nvariables,
01728                   calibrate->value_old + j * calibrate->nvariables,
01729                   calibrate->nvariables * sizeof (double));
01730           e[k] = *eold;
01731           ++k;
01732           ++eold;
01733           ++j;
01734         }
01735     }
01736   while (k < calibrate->nbest);
01737   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
01738   memcpy (calibrate->error_old, e, k * sizeof (double));
01739 #if DEBUG
01740   fprintf (stderr, "calibrate_merge_old: end\n");
01741 #endif
01742 }
01743
01749 void
01750 calibrate_refine ()
01751 {
01752   unsigned int i, j;
01753   double d;
01754 #if HAVE_MPI
01755   MPI_Status mpi_stat;
01756 #endif
01757 #if DEBUG
01758   fprintf (stderr, "calibrate_refine: start\n");
01759 #endif
01760 #if HAVE_MPI
01761   if (!calibrate->mpi_rank)
01762     {
01763 #endif
```

```
01764       for (j = 0; j < calibrate->nvariables; ++j)
01765         {
01766           calibrate->rangemin[j] = calibrate->rangemax[j]
01767             = calibrate->value_old[j];
01768         }
01769       for (i = 0; ++i < calibrate->nbest;)
01770         {
01771           for (j = 0; j < calibrate->nvariables; ++j)
01772             {
01773               calibrate->rangemin[j]
01774                 = fmin (calibrate->rangemin[j],
01775                         calibrate->value_old[i * calibrate->nvariables + j]);
01776               calibrate->rangemax[j]
01777                 = fmax (calibrate->rangemax[j],
01778                         calibrate->value_old[i * calibrate->nvariables + j]);
01779             }
01780         }
01781       for (j = 0; j < calibrate->nvariables; ++j)
01782         {
01783           d = 0.5 * calibrate->tolerance
01784             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01785           calibrate->rangemin[j] -= d;
01786           calibrate->rangemin[j]
01787             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
01788           calibrate->rangemax[j] += d;
01789           calibrate->rangemax[j]
01790             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
01791           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
01792                   calibrate->rangemin[j], calibrate->rangemax[j]);
01793           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
01794                    calibrate->label[j], calibrate->rangemin[j],
01795                    calibrate->rangemax[j]);
01796         }
01797 #if HAVE_MPI
01798       for (i = 1; i < ntasks; ++i)
01799         {
01800           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
01801                     1, MPI_COMM_WORLD);
01802           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
01803                     1, MPI_COMM_WORLD);
01804         }
01805     }
01806   else
01807     {
01808       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01809                 MPI_COMM_WORLD, &mpi_stat);
01810       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01811                 MPI_COMM_WORLD, &mpi_stat);
01812     }
01813 #endif
01814 #if DEBUG
01815   fprintf (stderr, "calibrate_refine: end\n");
01816 #endif
01817 }
01818
01823 void
01824 calibrate_iterate ()
01825 {
01826   unsigned int i;
01827 #if DEBUG
01828   fprintf (stderr, "calibrate_iterate: start\n");
01829 #endif
01830   calibrate->error_old
01831     = (double *) g_malloc (calibrate->nbest * sizeof (double));
01832   calibrate->value_old = (double *)
01833     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
01834   calibrate_step ();
01835   calibrate_save_old ();
01836   calibrate_refine ();
01837   calibrate_print ();
01838   for (i = 1; i < calibrate->niterations; ++i)
01839     {
01840       calibrate_step ();
01841       calibrate_merge_old ();
01842       calibrate_refine ();
01843       calibrate_print ();
01844     }
01845 #if DEBUG
01846   fprintf (stderr, "calibrate_iterate: end\n");
01847 #endif
01848 }
01849
01854 void
01855 calibrate_free ()
01856 {
01857   unsigned int i, j;
01858 #if DEBUG
```

```
01859   fprintf (stderr, "calibrate_free: start\n");
01860 #endif
01861   for (i = 0; i < calibrate->nexperiments; ++i)
01862     {
01863       for (j = 0; j < calibrate->ninputs; ++j)
01864         g_mapped_file_unref (calibrate->file[j][i]);
01865     }
01866   for (i = 0; i < calibrate->ninputs; ++i)
01867     g_free (calibrate->file[i]);
01868   g_free (calibrate->error_old);
01869   g_free (calibrate->value_old);
01870   g_free (calibrate->value);
01871   g_free (calibrate->genetic_variable);
01872 #if DEBUG
01873   fprintf (stderr, "calibrate_free: end\n");
01874 #endif
01875 }
01876
01881 void
01882 calibrate_new ()
01883 {
01884   unsigned int i, j, *nbits;
01885
01886 #if DEBUG
01887   fprintf (stderr, "calibrate_new: start\n");
01888 #endif
01889
01890   // Initing pseudo-random numbers generator
01891   gsl_rng_set (calibrate->rng, calibrate->seed);
01892
01893   // Replacing the working dir
01894   chdir (input->directory);
01895
01896   // Obtaining the simulator file
01897   calibrate->simulator = input->simulator;
01898
01899   // Obtaining the evaluator file
01900   calibrate->evaluator = input->evaluator;
01901
01902   // Obtaining the pseudo-random numbers generator seed
01903   calibrate->seed = input->seed;
01904
01905   // Reading the algorithm
01906   calibrate->algorithm = input->algorithm;
01907   switch (calibrate->algorithm)
01908     {
01909     case ALGORITHM_MONTE_CARLO:
01910       calibrate_step = calibrate_MonteCarlo;
01911       break;
01912     case ALGORITHM_SWEEP:
01913       calibrate_step = calibrate_sweep;
01914       break;
01915     default:
01916       calibrate_step = calibrate_genetic;
01917       calibrate->mutation_ratio = input->mutation_ratio;
01918       calibrate->reproduction_ratio = input->
    reproduction_ratio;
01919       calibrate->adaptation_ratio = input->adaptation_ratio;
01920     }
01921   calibrate->nsimulations = input->nsimulations;
01922   calibrate->niterations = input->niterations;
01923   calibrate->nbest = input->nbest;
01924   calibrate->tolerance = input->tolerance;
01925
01926   calibrate->simulation_best
01927     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
01928   calibrate->error_best
01929     = (double *) alloca (calibrate->nbest * sizeof (double));
01930
01931   // Reading the experimental data
01932 #if DEBUG
01933   fprintf (stderr, "calibrate_new: current directory=%s\n",
01934            g_get_current_dir ());
01935 #endif
01936   calibrate->nexperiments = input->nexperiments;
01937   calibrate->ninputs = input->ninputs;
01938   calibrate->experiment = input->experiment;
01939   calibrate->weight = input->weight;
01940   for (i = 0; i < input->ninputs; ++i)
01941     {
01942       calibrate->template[i] = input->template[i];
01943       calibrate->file[i]
01944         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
01945     }
01946   for (i = 0; i < input->nexperiments; ++i)
01947     {
01948 #if DEBUG
```

```
01949       fprintf (stderr, "calibrate_new: i=%u\n", i);
01950       fprintf (stderr, "calibrate_new: experiment=%s\n",
01951                calibrate->experiment[i]);
01952       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
01953 #endif
01954       for (j = 0; j < input->ninputs; ++j)
01955         {
01956 #if DEBUG
01957           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
01958           fprintf (stderr, "calibrate_new: experiment=%u template%u=%s\n",
01959                    i, j + 1, calibrate->template[j][i]);
01960 #endif
01961           calibrate->file[j][i]
01962             = g_mapped_file_new (input->template[j][i], 0, NULL);
01963         }
01964     }
01965
01966   // Reading the variables data
01967 #if DEBUG
01968   fprintf (stderr, "calibrate_new: reading variables\n");
01969 #endif
01970   calibrate->nvariables = input->nvariables;
01971   calibrate->label = input->label;
01972   calibrate->rangemin = input->rangemin;
01973   calibrate->rangeminabs = input->rangeminabs;
01974   calibrate->rangemax = input->rangemax;
01975   calibrate->rangemaxabs = input->rangemaxabs;
01976   calibrate->precision = input->precision;
01977   calibrate->nsweeps = input->nsweeps;
01978   nbits = input->nbits;
01979   if (input->algorithm == ALGORITHM_SWEEP)
01980     calibrate->nsimulations = 1;
01981   else if (input->algorithm == ALGORITHM_GENETIC)
01982     for (i = 0; i < input->nvariables; ++i)
01983       {
01984         if (calibrate->algorithm == ALGORITHM_SWEEP)
01985           {
01986             calibrate->nsimulations *= input->nsweeps[i];
01987 #if DEBUG
01988             fprintf (stderr, "calibrate_new: nsweeps=%u nsimulations=%u\n",
01989                      calibrate->nsweeps[i], calibrate->nsimulations);
01990 #endif
01991           }
01992       }
01993
01994   // Allocating values
01995 #if DEBUG
01996   fprintf (stderr, "calibrate_new: allocating variables\n");
01997   fprintf (stderr, "calibrate_new: nvariables=%u\n", calibrate->nvariables);
01998 #endif
01999   calibrate->genetic_variable = NULL;
02000   if (calibrate->algorithm == ALGORITHM_GENETIC)
02001     {
02002       calibrate->genetic_variable = (GeneticVariable *)
02003         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02004       for (i = 0; i < calibrate->nvariables; ++i)
02005         {
02006 #if DEBUG
02007           fprintf (stderr, "calibrate_new: i=%u min=%lg max=%lg nbits=%u\n",
02008                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02009 #endif
02010           calibrate->genetic_variable[i].minimum = calibrate->
      rangemin[i];
02011           calibrate->genetic_variable[i].maximum = calibrate->
      rangemax[i];
02012           calibrate->genetic_variable[i].nbits = nbits[i];
02013         }
02014     }
02015 #if DEBUG
02016   fprintf (stderr, "calibrate_new: nvariables=%u nsimulations=%u\n",
02017            calibrate->nvariables, calibrate->nsimulations);
02018 #endif
02019   calibrate->value = (double *) g_malloc (calibrate->nsimulations *
02020                                           calibrate->nvariables *
02021                                           sizeof (double));
02022
02023   // Calculating simulations to perform on each task
02024 #if HAVE_MPI
02025 #if DEBUG
02026   fprintf (stderr, "calibrate_new: rank=%u ntasks=%u\n",
02027            calibrate->mpi_rank, ntasks);
02028 #endif
02029   calibrate->nstart = calibrate->mpi_rank * calibrate->
      nsimulations / ntasks;
02030   calibrate->nend = (1 + calibrate->mpi_rank) * calibrate->
      nsimulations
02031     / ntasks;
```

```
02032 #else
02033   calibrate->nstart = 0;
02034   calibrate->nend = calibrate->nsimulations;
02035 #endif
02036 #if DEBUG
02037   fprintf (stderr, "calibrate_new: nstart=%u nend=%u\n", calibrate->nstart,
02038            calibrate->nend);
02039 #endif
02040
02041   // Calculating simulations to perform on each thread
02042   calibrate->thread
02043     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02044   for (i = 0; i <= nthreads; ++i)
02045     {
02046       calibrate->thread[i] = calibrate->nstart
02047         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02048 #if DEBUG
02049       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02050                calibrate->thread[i]);
02051 #endif
02052     }
02053
02054   // Opening result files
02055   calibrate->file_result = fopen ("result", "w");
02056   calibrate->file_variables = fopen ("variables", "w");
02057
02058   // Performing the algorithm
02059   switch (calibrate->algorithm)
02060     {
02061       // Genetic algorithm
02062     case ALGORITHM_GENETIC:
02063       calibrate_genetic ();
02064       break;
02065
02066       // Iterative algorithm
02067     default:
02068       calibrate_iterate ();
02069     }
02070
02071   // Closing result files
02072   fclose (calibrate->file_variables);
02073   fclose (calibrate->file_result);
02074
02075 #if DEBUG
02076   fprintf (stderr, "calibrate_new: end\n");
02077 #endif
02078 }
02079
02080 #if HAVE_GTK
02081
02088 void
02089 input_save (char *filename)
02090 {
02091   unsigned int i, j;
02092   char *buffer;
02093   xmlDoc *doc;
02094   xmlNode *node, *child;
02095   GFile *file, *file2;
02096
02097   // Getting the input file directory
02098   input->name = g_path_get_basename (filename);
02099   input->directory = g_path_get_dirname (filename);
02100   file = g_file_new_for_path (input->directory);
02101
02102   // Opening the input file
02103   doc = xmlNewDoc ((const xmlChar *) "1.0");
02104
02105   // Setting root XML node
02106   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02107   xmlDocSetRootElement (doc, node);
02108
02109   // Adding properties to the root XML node
02110   file2 = g_file_new_for_path (input->simulator);
02111   buffer = g_file_get_relative_path (file, file2);
02112   g_object_unref (file2);
02113   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02114   g_free (buffer);
02115   if (input->evaluator)
02116     {
02117       file2 = g_file_new_for_path (input->evaluator);
02118       buffer = g_file_get_relative_path (file, file2);
02119       g_object_unref (file2);
02120       if (xmlStrlen ((xmlChar *) buffer))
02121         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02122       g_free (buffer);
02123     }
02124   if (input->seed != DEFAULT_RANDOM_SEED)
```

```
02125     xml_node_set_uint (node, XML_SEED, input->seed);
02126
02127   // Setting the algorithm
02128   buffer = (char *) g_malloc (64);
02129   switch (input->algorithm)
02130     {
02131     case ALGORITHM_MONTE_CARLO:
02132       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02133       snprintf (buffer, 64, "%u", input->nsimulations);
02134       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02135       snprintf (buffer, 64, "%u", input->niterations);
02136       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02137       snprintf (buffer, 64, "%.3lg", input->tolerance);
02138       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02139       snprintf (buffer, 64, "%u", input->nbest);
02140       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02141       break;
02142     case ALGORITHM_SWEEP:
02143       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02144       snprintf (buffer, 64, "%u", input->niterations);
02145       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02146       snprintf (buffer, 64, "%.3lg", input->tolerance);
02147       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02148       snprintf (buffer, 64, "%u", input->nbest);
02149       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02150       break;
02151     default:
02152       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02153       snprintf (buffer, 64, "%u", input->nsimulations);
02154       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02155       snprintf (buffer, 64, "%u", input->niterations);
02156       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02157       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02158       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02159       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02160       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02161       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02162       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02163       break;
02164     }
02165   g_free (buffer);
02166
02167   // Setting the experimental data
02168   for (i = 0; i < input->nexperiments; ++i)
02169     {
02170       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02171       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02172       if (input->weight[i] != 1.)
02173         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02174       for (j = 0; j < input->ninputs; ++j)
02175         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02176     }
02177
02178   // Setting the variables data
02179   for (i = 0; i < input->nvariables; ++i)
02180     {
02181       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02182       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02183       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02184       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02185         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02186       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02187       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02188         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02189       if (input->precision[i] != DEFAULT_PRECISION)
02190         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02191       if (input->algorithm == ALGORITHM_SWEEP)
02192         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02193       else if (input->algorithm == ALGORITHM_GENETIC)
02194         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02195     }
02196
02197   // Saving the XML file
02198   xmlSaveFormatFile (filename, doc, 1);
02199
02200   // Freeing memory
02201   xmlFreeDoc (doc);
02202 }
02203
```

```
02208 void
02209 options_new ()
02210 {
02211   options->label_processors
02212     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02213   options->spin_processors
02214     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02215   gtk_spin_button_set_value (options->spin_processors, (gdouble)
      nthreads);
02216   options->label_seed = (GtkLabel *)
02217     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02218   options->spin_seed = (GtkSpinButton *)
02219     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02220   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02221   options->grid = (GtkGrid *) gtk_grid_new ();
02222   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02223                    0, 0, 1, 1);
02224   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02225                    1, 0, 1, 1);
02226   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 1, 1, 1);
02227   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 1, 1, 1);
02228   gtk_widget_show_all (GTK_WIDGET (options->grid));
02229   options->dialog = (GtkDialog *)
02230     gtk_dialog_new_with_buttons (gettext ("Options"),
02231                                  window->window,
02232                                  GTK_DIALOG_MODAL,
02233                                  gettext ("_OK"), GTK_RESPONSE_OK,
02234                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02235                                  NULL);
02236   gtk_container_add
02237     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02238      GTK_WIDGET (options->grid));
02239   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02240     {
02241       nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02242       input->seed
02243         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02244     }
02245   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02246 }
02247
02252 void
02253 running_new ()
02254 {
02255   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02256   running->dialog = (GtkDialog *)
02257     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02258                                  window->window,
02259                                  GTK_DIALOG_DESTROY_WITH_PARENT, NULL, NULL);
02260   gtk_container_add
02261     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02262      GTK_WIDGET (running->label));
02263   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02264 }
02265
02271 int
02272 window_save ()
02273 {
02274   char *buffer;
02275   GtkFileChooserDialog *dlg;
02276
02277 #if DEBUG
02278   fprintf (stderr, "window_save: start\n");
02279 #endif
02280
02281   // Opening the saving dialog
02282   dlg = (GtkFileChooserDialog *)
02283     gtk_file_chooser_dialog_new (gettext ("Save file"),
02284                                  window->window,
02285                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02286                                  gettext ("_Cancel"),
02287                                  GTK_RESPONSE_CANCEL,
02288                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02289   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02290
02291   // If OK response then saving
02292   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02293     {
02294
02295       // Adding properties to the root XML node
02296       input->simulator = gtk_file_chooser_get_filename
02297         (GTK_FILE_CHOOSER (window->button_simulator));
02298       if (gtk_toggle_button_get_active
02299           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02300         input->evaluator = gtk_file_chooser_get_filename
02301           (GTK_FILE_CHOOSER (window->button_evaluator));
02302       else
```

```
02303         input->evaluator = NULL;
02304
02305       // Setting the algorithm
02306       switch (window_get_algorithm ())
02307         {
02308         case ALGORITHM_MONTE_CARLO:
02309           input->algorithm = ALGORITHM_MONTE_CARLO;
02310           input->nsimulations
02311             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02312           input->niterations
02313             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02314           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02315           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02316           break;
02317         case ALGORITHM_SWEEP:
02318           input->algorithm = ALGORITHM_SWEEP;
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02321           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02322           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02323           break;
02324         default:
02325           input->algorithm = ALGORITHM_GENETIC;
02326           input->nsimulations
02327             = gtk_spin_button_get_value_as_int (window->spin_population);
02328           input->niterations
02329             = gtk_spin_button_get_value_as_int (window->spin_generations);
02330           input->mutation_ratio
02331             = gtk_spin_button_get_value (window->spin_mutation);
02332           input->reproduction_ratio
02333             = gtk_spin_button_get_value (window->spin_reproduction);
02334           input->adaptation_ratio
02335             = gtk_spin_button_get_value (window->spin_adaptation);
02336           break;
02337         }
02338
02339       // Saving the XML file
02340       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02341       input_save (buffer);
02342
02343       // Closing and freeing memory
02344       g_free (buffer);
02345       gtk_widget_destroy (GTK_WIDGET (dlg));
02346 #if DEBUG
02347       fprintf (stderr, "window_save: end\n");
02348 #endif
02349       return 1;
02350     }
02351
02352   // Closing and freeing memory
02353   gtk_widget_destroy (GTK_WIDGET (dlg));
02354 #if DEBUG
02355   fprintf (stderr, "window_save: end\n");
02356 #endif
02357   return 0;
02358 }
02359
02364 void
02365 window_run ()
02366 {
02367   unsigned int i;
02368   char *msg, *msg2, buffer[64], buffer2[64];
02369 #if DEBUG
02370   fprintf (stderr, "window_run: start\n");
02371 #endif
02372   if (!window_save ())
02373     {
02374 #if DEBUG
02375       fprintf (stderr, "window_run: end\n");
02376 #endif
02377       return;
02378     }
02379   running_new ();
02380   while (g_main_context_pending (NULL))
02381     g_main_context_iteration (NULL, FALSE);
02382   calibrate_new ();
02383   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02384   snprintf (buffer, 64, "error=%.15le\n", calibrate->error_old[0]);
02385   msg2 = g_strdup (buffer);
02386   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02387     {
02388       snprintf (buffer, 64, "%s=%s\n",
02389                 calibrate->label[i], format[calibrate->precision[i]]);
```

```
02390       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02391       msg = g_strconcat (msg2, buffer2, NULL);
02392       g_free (msg2);
02393     }
02394   show_message (gettext ("Best result"), msg2, INFO_TYPE);
02395   g_free (msg2);
02396   calibrate_free ();
02397 #if DEBUG
02398   fprintf (stderr, "window_run: end\n");
02399 #endif
02400 }
02401
02406 void
02407 window_help ()
02408 {
02409   char *buffer, *buffer2;
02410   buffer2 = g_build_filename (current_directory, "manuals",
02411                               gettext ("user-manual.pdf"), NULL);
02412   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02413   g_free (buffer2);
02414   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02415   g_free (buffer);
02416 }
02417
02422 void
02423 window_about ()
02424 {
02425   gchar *authors[] = {
02426     "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02427     "Borja Latorre Garcés (borja.latorre@csic.es)",
02428     NULL
02429   };
02430   gtk_show_about_dialog (window->window,
02431                          "program_name",
02432                          "Calibrator",
02433                          "comments",
02434                          gettext ("A software to make calibrations of "
02435                                   "empirical parameters"),
02436                          "authors", authors,
02437                          "translator-credits",
02438                          "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02439                          "version", "1.1.27", "copyright",
02440                          "Copyright 2012-2015 Javier Burguete Tolosa",
02441                          "logo", window->logo,
02442                          "website-label", gettext ("Website"),
02443                          "website",
02444                          "https://github.com/jburguete/calibrator", NULL);
02445 }
02446
02452 int
02453 window_get_algorithm ()
02454 {
02455   unsigned int i;
02456   for (i = 0; i < NALGORITHMS; ++i)
02457     if (gtk_toggle_button_get_active
02458         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02459       break;
02460   return i;
02461 }
02462
02467 void
02468 window_update ()
02469 {
02470   unsigned int i;
02471   gtk_widget_set_sensitive
02472     (GTK_WIDGET (window->button_evaluator),
02473      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02474                                    (window->check_evaluator)));
02475   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02476   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02477   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02478   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02479   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02480   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
02481   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02482   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02483   gtk_widget_hide (GTK_WIDGET (window->label_population));
02484   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02485   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02486   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02487   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02488   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
02489   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02490   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
02491   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
02492   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
02493   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
```

```
02494   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
02495   gtk_widget_hide (GTK_WIDGET (window->label_bits));
02496   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
02497   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
02498   switch (window_get_algorithm ())
02499     {
02500     case ALGORITHM_MONTE_CARLO:
02501       gtk_widget_show (GTK_WIDGET (window->label_simulations));
02502       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
02503       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02504       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02505       if (i > 1)
02506         {
02507           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02508           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02509           gtk_widget_show (GTK_WIDGET (window->label_bests));
02510           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02511         }
02512       break;
02513     case ALGORITHM_SWEEP:
02514       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02515       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02516       gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02517       gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02518       if (i > 1)
02519         {
02520           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02521           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02522           gtk_widget_show (GTK_WIDGET (window->label_bests));
02523           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02524         }
02525       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
02526       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
02527       break;
02528     default:
02529       gtk_widget_show (GTK_WIDGET (window->label_population));
02530       gtk_widget_show (GTK_WIDGET (window->spin_population));
02531       gtk_widget_show (GTK_WIDGET (window->label_generations));
02532       gtk_widget_show (GTK_WIDGET (window->spin_generations));
02533       gtk_widget_show (GTK_WIDGET (window->label_mutation));
02534       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
02535       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
02536       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
02537       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
02538       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
02539       gtk_widget_show (GTK_WIDGET (window->label_bits));
02540       gtk_widget_show (GTK_WIDGET (window->spin_bits));
02541     }
02542   gtk_widget_set_sensitive
02543     (GTK_WIDGET (window->button_remove_experiment), input->
    nexperiments > 1);
02544   gtk_widget_set_sensitive
02545     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
02546   for (i = 0; i < input->ninputs; ++i)
02547     {
02548       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02549       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02550       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
02551       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
02552       g_signal_handler_block
02553         (window->check_template[i], window->id_template[i]);
02554       g_signal_handler_block (window->button_template[i], window->
    id_input[i]);
02555       gtk_toggle_button_set_active
02556         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
02557       g_signal_handler_unblock
02558         (window->button_template[i], window->id_input[i]);
02559       g_signal_handler_unblock
02560         (window->check_template[i], window->id_template[i]);
02561     }
02562   if (i > 0)
02563     {
02564       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
02565       gtk_widget_set_sensitive
02566         (GTK_WIDGET (window->button_template[i - 1]),
02567          gtk_toggle_button_get_active
02568          GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
02569     }
02570   if (i < MAX_NINPUTS)
02571     {
02572       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02573       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02574       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
02575       gtk_widget_set_sensitive
02576         (GTK_WIDGET (window->button_template[i]),
02577          gtk_toggle_button_get_active
```

```
02578           GTK_TOGGLE_BUTTON (window->check_template[i]));
02579       g_signal_handler_block
02580         (window->check_template[i], window->id_template[i]);
02581       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02582       gtk_toggle_button_set_active
02583         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02584       g_signal_handler_unblock
02585         (window->button_template[i], window->id_input[i]);
02586       g_signal_handler_unblock
02587         (window->check_template[i], window->id_template[i]);
02588     }
02589   while (++i < MAX_NINPUTS)
02590     {
02591       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02592       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02593     }
02594   gtk_widget_set_sensitive
02595     (GTK_WIDGET (window->spin_minabs),
02596      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02597   gtk_widget_set_sensitive
02598     (GTK_WIDGET (window->spin_maxabs),
02599      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02600 }
02601
02606 void
02607 window_set_algorithm ()
02608 {
02609   unsigned int i;
02610   i = window_get_algorithm ();
02611   switch (i)
02612     {
02613     case ALGORITHM_SWEEP:
02614       input->nsweeps = (unsigned int *) g_realloc
02615         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02616       break;
02617     case ALGORITHM_GENETIC:
02618       input->nbits = (unsigned int *) g_realloc
02619         (input->nbits, input->nvariables * sizeof (unsigned int));
02620     }
02621   window_update ();
02622 }
02623
02628 void
02629 window_set_experiment ()
02630 {
02631   unsigned int i, j;
02632   char *buffer1, *buffer2;
02633 #if DEBUG
02634   fprintf (stderr, "window_set_experiment: start\n");
02635 #endif
02636   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02637   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02638   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02639   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02640   g_free (buffer1);
02641   g_signal_handler_block
02642     (window->button_experiment, window->id_experiment_name);
02643   gtk_file_chooser_set_filename
02644     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
02645   g_signal_handler_unblock
02646     (window->button_experiment, window->id_experiment_name);
02647   g_free (buffer2);
02648   for (j = 0; j < input->ninputs; ++j)
02649     {
02650       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02651       buffer2
02652         = g_build_filename (input->directory, input->template[j][i], NULL);
02653       gtk_file_chooser_set_filename
02654         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02655       g_free (buffer2);
02656       g_signal_handler_unblock
02657         (window->button_template[j], window->id_input[j]);
02658     }
02659 #if DEBUG
02660   fprintf (stderr, "window_set_experiment: end\n");
02661 #endif
02662 }
02663
02668 void
02669 window_remove_experiment ()
02670 {
02671   unsigned int i, j;
02672   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02673   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
```

```
02674   gtk_combo_box_text_remove (window->combo_experiment, i);
02675   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02676   xmlFree (input->experiment[i]);
02677   --input->nexperiments;
02678   for (j = i; j < input->nexperiments; ++j)
02679     {
02680       input->experiment[j] = input->experiment[j + 1];
02681       input->weight[j] = input->weight[j + 1];
02682     }
02683   j = input->nexperiments - 1;
02684   if (i > j)
02685     i = j;
02686   for (j = 0; j < input->ninputs; ++j)
02687     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02688   g_signal_handler_block
02689     (window->button_experiment, window->id_experiment_name);
02690   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02691   g_signal_handler_unblock
02692     (window->button_experiment, window->id_experiment_name);
02693   for (j = 0; j < input->ninputs; ++j)
02694     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02695   window_update ();
02696 }
02697
02702 void
02703 window_add_experiment ()
02704 {
02705   unsigned int i, j;
02706   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02707   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02708   gtk_combo_box_text_insert_text
02709     (window->combo_experiment, i, input->experiment[i]);
02710   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02711   input->experiment = (char **) g_realloc
02712     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02713   input->weight = (double *) g_realloc
02714     (input->weight, (input->nexperiments + 1) * sizeof (double));
02715   for (j = input->nexperiments - 1; j > i; --j)
02716     {
02717       input->experiment[j + 1] = input->experiment[j];
02718       input->weight[j + 1] = input->weight[j];
02719     }
02720   input->experiment[j + 1]
02721     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02722   input->weight[j + 1] = input->weight[j];
02723   ++input->nexperiments;
02724   for (j = 0; j < input->ninputs; ++j)
02725     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02726   g_signal_handler_block
02727     (window->button_experiment, window->id_experiment_name);
02728   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02729   g_signal_handler_unblock
02730     (window->button_experiment, window->id_experiment_name);
02731   for (j = 0; j < input->ninputs; ++j)
02732     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02733   window_update ();
02734 }
02735
02740 void
02741 window_name_experiment ()
02742 {
02743   unsigned int i;
02744   char *buffer;
02745   GFile *file1, *file2;
02746 #if DEBUG
02747   fprintf (stderr, "window_name_experiment: start\n");
02748 #endif
02749   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02750   file1
02751     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02752   file2 = g_file_new_for_path (input->directory);
02753   buffer = g_file_get_relative_path (file2, file1);
02754   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02755   gtk_combo_box_text_remove (window->combo_experiment, i);
02756   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02757   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02758   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02759   g_free (buffer);
```

```
02760   g_object_unref (file2);
02761   g_object_unref (file1);
02762 #if DEBUG
02763   fprintf (stderr, "window_name_experiment: end\n");
02764 #endif
02765 }
02766
02771 void
02772 window_weight_experiment ()
02773 {
02774   unsigned int i;
02775 #if DEBUG
02776   fprintf (stderr, "window_weight_experiment: start\n");
02777 #endif
02778   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02779   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02780 #if DEBUG
02781   fprintf (stderr, "window_weight_experiment: end\n");
02782 #endif
02783 }
02784
02790 void
02791 window_inputs_experiment ()
02792 {
02793   unsigned int j;
02794 #if DEBUG
02795   fprintf (stderr, "window_inputs_experiment: start\n");
02796 #endif
02797   j = input->ninputs - 1;
02798   if (j
02799       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02800                                         (window->check_template[j])))
02801     --input->ninputs;
02802   if (input->ninputs < MAX_NINPUTS
02803       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02804                                        (window->check_template[j])))
02805     {
02806       ++input->ninputs;
02807       for (j = 0; j < input->ninputs; ++j)
02808         {
02809           input->template[j] = (char **)
02810             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02811         }
02812     }
02813   window_update ();
02814 #if DEBUG
02815   fprintf (stderr, "window_inputs_experiment: end\n");
02816 #endif
02817 }
02818
02826 void
02827 window_template_experiment (void *data)
02828 {
02829   unsigned int i, j;
02830   char *buffer;
02831   GFile *file1, *file2;
02832 #if DEBUG
02833   fprintf (stderr, "window_template_experiment: start\n");
02834 #endif
02835   i = (size_t) data;
02836   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02837   file1
02838     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02839   file2 = g_file_new_for_path (input->directory);
02840   buffer = g_file_get_relative_path (file2, file1);
02841   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02842   g_free (buffer);
02843   g_object_unref (file2);
02844   g_object_unref (file1);
02845 #if DEBUG
02846   fprintf (stderr, "window_template_experiment: end\n");
02847 #endif
02848 }
02849
02854 void
02855 window_set_variable ()
02856 {
02857   unsigned int i;
02858 #if DEBUG
02859   fprintf (stderr, "window_set_variable: start\n");
02860 #endif
02861   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02862   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02863   gtk_entry_set_text (window->entry_variable, input->label[i]);
02864   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
```

```
02865   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02866   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
02867   if (input->rangeminabs[i] != -G_MAXDOUBLE)
02868     {
02869       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
02870       gtk_toggle_button_set_active
02871         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
02872     }
02873   else
02874     {
02875       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
02876       gtk_toggle_button_set_active
02877         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
02878     }
02879   if (input->rangemaxabs[i] != G_MAXDOUBLE)
02880     {
02881       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
02882       gtk_toggle_button_set_active
02883         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
02884     }
02885   else
02886     {
02887       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
02888       gtk_toggle_button_set_active
02889         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
02890     }
02891   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
02892   switch (input->algorithm)
02893     {
02894     case ALGORITHM_SWEEP:
02895       gtk_spin_button_set_value (window->spin_sweeps,
02896                                  (gdouble) input->nsweeps[i]);
02897       break;
02898     case ALGORITHM_GENETIC:
02899       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02900       break;
02901     }
02902   window_update ();
02903 #if DEBUG
02904   fprintf (stderr, "window_set_variable: end\n");
02905 #endif
02906 }
02907
02912 void
02913 window_remove_variable ()
02914 {
02915   unsigned int i, j;
02916   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02917   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02918   gtk_combo_box_text_remove (window->combo_variable, i);
02919   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02920   xmlFree (input->label[i]);
02921   --input->nvariables;
02922   for (j = i; j < input->nvariables; ++j)
02923     {
02924       input->label[j] = input->label[j + 1];
02925       input->rangemin[j] = input->rangemin[j + 1];
02926       input->rangemax[j] = input->rangemax[j + 1];
02927       input->rangeminabs[j] = input->rangeminabs[j + 1];
02928       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
02929       input->precision[j] = input->precision[j + 1];
02930       switch (window_get_algorithm ())
02931         {
02932         case ALGORITHM_SWEEP:
02933           input->nsweeps[j] = input->nsweeps[j + 1];
02934           break;
02935         case ALGORITHM_GENETIC:
02936           input->nbits[j] = input->nbits[j + 1];
02937         }
02938     }
02939   j = input->nvariables - 1;
02940   if (i > j)
02941     i = j;
02942   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02943   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
02944   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02945   window_update ();
02946 }
02947
```

```
02952 void
02953 window_add_variable ()
02954 {
02955   unsigned int i, j;
02956 #if DEBUG
02957   fprintf (stderr, "window_add_variable: start\n");
02958 #endif
02959   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02960   g_signal_handler_block (window->combo_variable, window->
      id_variable);
02961   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
02962   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02963   input->label = (char **) g_realloc
02964     (input->label, (input->nvariables + 1) * sizeof (char *));
02965   input->rangemin = (double *) g_realloc
02966     (input->rangemin, (input->nvariables + 1) * sizeof (double));
02967   input->rangemax = (double *) g_realloc
02968     (input->rangemax, (input->nvariables + 1) * sizeof (double));
02969   input->rangeminabs = (double *) g_realloc
02970     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
02971   input->rangemaxabs = (double *) g_realloc
02972     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
02973   input->precision = (unsigned int *) g_realloc
02974     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
02975   for (j = input->nvariables - 1; j > i; --j)
02976     {
02977       input->label[j + 1] = input->label[j];
02978       input->rangemin[j + 1] = input->rangemin[j];
02979       input->rangemax[j + 1] = input->rangemax[j];
02980       input->rangeminabs[j + 1] = input->rangeminabs[j];
02981       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02982       input->precision[j + 1] = input->precision[j];
02983     }
02984   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
02985   input->rangemin[j + 1] = input->rangemin[j];
02986   input->rangemax[j + 1] = input->rangemax[j];
02987   input->rangeminabs[j + 1] = input->rangeminabs[j];
02988   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
02989   input->precision[j + 1] = input->precision[j];
02990   switch (window_get_algorithm ())
02991     {
02992     case ALGORITHM_SWEEP:
02993       input->nsweeps = (unsigned int *) g_realloc
02994         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
02995       for (j = input->nvariables - 1; j > i; --j)
02996         input->nsweeps[j + 1] = input->nsweeps[j];
02997       input->nsweeps[j + 1] = input->nsweeps[j];
02998       break;
02999     case ALGORITHM_GENETIC:
03000       input->nbits = (unsigned int *) g_realloc
03001         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03002       for (j = input->nvariables - 1; j > i; --j)
03003         input->nbits[j + 1] = input->nbits[j];
03004       input->nbits[j + 1] = input->nbits[j];
03005     }
03006   ++input->nvariables;
03007   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03008   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03009   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03010   window_update ();
03011 #if DEBUG
03012   fprintf (stderr, "window_add_variable: end\n");
03013 #endif
03014 }
03015
03020 void
03021 window_label_variable ()
03022 {
03023   unsigned int i;
03024   const char *buffer;
03025 #if DEBUG
03026   fprintf (stderr, "window_label_variable: start\n");
03027 #endif
03028   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03029   buffer = gtk_entry_get_text (window->entry_variable);
03030   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03031   gtk_combo_box_text_remove (window->combo_variable, i);
03032   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03033   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03034   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03035 #if DEBUG
```

```
03036   fprintf (stderr, "window_label_variable: end\n");
03037 #endif
03038 }
03039
03044 void
03045 window_precision_variable ()
03046 {
03047   unsigned int i;
03048 #if DEBUG
03049   fprintf (stderr, "window_precision_variable: start\n");
03050 #endif
03051   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03052   input->precision[i]
03053     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03054   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03055   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03056   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03057   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03058 #if DEBUG
03059   fprintf (stderr, "window_precision_variable: end\n");
03060 #endif
03061 }
03062
03067 void
03068 window_rangemin_variable ()
03069 {
03070   unsigned int i;
03071 #if DEBUG
03072   fprintf (stderr, "window_rangemin_variable: start\n");
03073 #endif
03074   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03075   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03076 #if DEBUG
03077   fprintf (stderr, "window_rangemin_variable: end\n");
03078 #endif
03079 }
03080
03085 void
03086 window_rangemax_variable ()
03087 {
03088   unsigned int i;
03089 #if DEBUG
03090   fprintf (stderr, "window_rangemax_variable: start\n");
03091 #endif
03092   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03093   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03094 #if DEBUG
03095   fprintf (stderr, "window_rangemax_variable: end\n");
03096 #endif
03097 }
03098
03103 void
03104 window_rangeminabs_variable ()
03105 {
03106   unsigned int i;
03107 #if DEBUG
03108   fprintf (stderr, "window_rangeminabs_variable: start\n");
03109 #endif
03110   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03111   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03112 #if DEBUG
03113   fprintf (stderr, "window_rangeminabs_variable: end\n");
03114 #endif
03115 }
03116
03121 void
03122 window_rangemaxabs_variable ()
03123 {
03124   unsigned int i;
03125 #if DEBUG
03126   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03127 #endif
03128   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03129   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03130 #if DEBUG
03131   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03132 #endif
03133 }
03134
03139 void
03140 window_update_variable ()
03141 {
03142   unsigned int i;
03143 #if DEBUG
03144   fprintf (stderr, "window_update_variable: start\n");
```

```
03145 #endif
03146   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03147   switch (window_get_algorithm ())
03148     {
03149     case ALGORITHM_SWEEP:
03150       input->nsweeps[i]
03151         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03152       break;
03153     case ALGORITHM_GENETIC:
03154       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03155     }
03156 #if DEBUG
03157   fprintf (stderr, "window_update_variable: end\n");
03158 #endif
03159 }
03160
03168 int
03169 window_read (char *filename)
03170 {
03171   unsigned int i;
03172   char *buffer;
03173 #if DEBUG
03174   fprintf (stderr, "window_read: start\n");
03175 #endif
03176   input_free ();
03177   if (!input_open (filename))
03178     {
03179 #if DEBUG
03180       fprintf (stderr, "window_read: end\n");
03181 #endif
03182       return 0;
03183     }
03184   buffer = g_build_filename (input->directory, input->simulator, NULL);
03185   puts (buffer);
03186   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03187                                  (window->button_simulator), buffer);
03188   g_free (buffer);
03189   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03190                                 (size_t) input->evaluator);
03191   if (input->evaluator)
03192     {
03193       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03194       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03195                                      (window->button_evaluator), buffer);
03196       g_free (buffer);
03197     }
03198   gtk_toggle_button_set_active
03199     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03200   switch (input->algorithm)
03201     {
03202     case ALGORITHM_MONTE_CARLO:
03203       gtk_spin_button_set_value (window->spin_simulations,
03204                                  (gdouble) input->nsimulations);
03205     case ALGORITHM_SWEEP:
03206       gtk_spin_button_set_value (window->spin_iterations,
03207                                  (gdouble) input->niterations);
03208       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03209       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03210       break;
03211     default:
03212       gtk_spin_button_set_value (window->spin_population,
03213                                  (gdouble) input->nsimulations);
03214       gtk_spin_button_set_value (window->spin_generations,
03215                                  (gdouble) input->niterations);
03216       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03217       gtk_spin_button_set_value (window->spin_reproduction,
03218                                  input->reproduction_ratio);
03219       gtk_spin_button_set_value (window->spin_adaptation,
03220                                  input->adaptation_ratio);
03221     }
03222   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03223   g_signal_handler_block (window->button_experiment,
03224                           window->id_experiment_name);
03225   gtk_combo_box_text_remove_all (window->combo_experiment);
03226   for (i = 0; i < input->nexperiments; ++i)
03227     gtk_combo_box_text_append_text (window->combo_experiment,
03228                                     input->experiment[i]);
03229   g_signal_handler_unblock
03230     (window->button_experiment, window->id_experiment_name);
03231   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03232   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
```

```
03233   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03234   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03235   gtk_combo_box_text_remove_all (window->combo_variable);
03236   for (i = 0; i < input->nvariables; ++i)
03237     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03238   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03239   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03240   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03241   window_set_variable ();
03242   window_update ();
03243 #if DEBUG
03244   fprintf (stderr, "window_read: end\n");
03245 #endif
03246   return 1;
03247 }
03248
03253 void
03254 window_open ()
03255 {
03256   char *buffer;
03257   GtkFileChooserDialog *dlg;
03258   dlg = (GtkFileChooserDialog *)
03259     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03260                                  window->window,
03261                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03262                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03263                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03264   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03265     {
03266       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03267       if (!window_read (buffer))
03268         g_application_quit (G_APPLICATION (window->application));
03269       g_free (buffer);
03270     }
03271   gtk_widget_destroy (GTK_WIDGET (dlg));
03272 }
03273
03278 void
03279 window_new ()
03280 {
03281   unsigned int i;
03282   char *buffer, *buffer2, buffer3[64];
03283   GtkViewport *viewport;
03284   char *label_algorithm[NALGORITHMS] = {
03285     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03286   };
03287
03288   // Creating the window
03289   window->window
03290     = (GtkWindow *) gtk_application_window_new (window->application);
03291
03292   // Setting the window title
03293   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03294
03295   // Creating the open button
03296   window->button_open = (GtkToolButton *) gtk_tool_button_new
03297     (gtk_image_new_from_icon_name ("document-open",
03298                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03299      gettext ("Open"));
03300   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03301
03302   // Creating the save button
03303   window->button_save = (GtkToolButton *) gtk_tool_button_new
03304     (gtk_image_new_from_icon_name ("document-save",
03305                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03306      gettext ("Save"));
03307   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
03308                     NULL);
03309
03310   // Creating the run button
03311   window->button_run = (GtkToolButton *) gtk_tool_button_new
03312     (gtk_image_new_from_icon_name ("system-run",
03313                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03314      gettext ("Run"));
03315   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03316
03317   // Creating the options button
03318   window->button_options = (GtkToolButton *) gtk_tool_button_new
03319     (gtk_image_new_from_icon_name ("preferences-system",
03320                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03321      gettext ("Options"));
```

```
03322   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03323
03324   // Creating the help button
03325   window->button_help = (GtkToolButton *) gtk_tool_button_new
03326     (gtk_image_new_from_icon_name ("help-browser",
03327                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03328      gettext ("Help"));
03329   g_signal_connect (window->button_help, "clicked", window_help, NULL);
03330
03331   // Creating the about button
03332   window->button_about = (GtkToolButton *) gtk_tool_button_new
03333     (gtk_image_new_from_icon_name ("help-about",
03334                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03335      gettext ("About"));
03336   g_signal_connect (window->button_about, "clicked", window_about, NULL);
03337
03338   // Creating the exit button
03339   window->button_exit = (GtkToolButton *) gtk_tool_button_new
03340     (gtk_image_new_from_icon_name ("application-exit",
03341                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03342      gettext ("Exit"));
03343   g_signal_connect_swapped (window->button_exit, "clicked",
03344                             (void (*)) gtk_widget_destroy, window->window);
03345
03346   // Creating the buttons bar
03347   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
03348   gtk_toolbar_insert
03349     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
03350   gtk_toolbar_insert
03351     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
03352   gtk_toolbar_insert
03353     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
03354   gtk_toolbar_insert
03355     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
03356   gtk_toolbar_insert
03357     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
03358   gtk_toolbar_insert
03359     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
03360   gtk_toolbar_insert
03361     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
03362   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
03363
03364   // Creating the simulator program label and entry
03365   window->label_simulator
03366     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
03367   window->button_simulator = (GtkFileChooserButton *)
03368     gtk_file_chooser_button_new (gettext ("Simulator program"),
03369                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03370   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
03371                                gettext ("Simulator program executable file"));
03372
03373   // Creating the evaluator program label and entry
03374   window->check_evaluator = (GtkCheckButton *)
03375     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
03376   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
03377   window->button_evaluator = (GtkFileChooserButton *)
03378     gtk_file_chooser_button_new (gettext ("Evaluator program"),
03379                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03380   gtk_widget_set_tooltip_text
03381     (GTK_WIDGET (window->button_evaluator),
03382      gettext ("Optional evaluator program executable file"));
03383
03384   // Creating the algorithm properties
03385   window->label_simulations = (GtkLabel *) gtk_label_new
03386     (gettext ("Simulations number"));
03387   window->spin_simulations
03388     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03389   window->label_iterations = (GtkLabel *)
03390     gtk_label_new (gettext ("Iterations number"));
03391   window->spin_iterations
03392     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03393   g_signal_connect
03394     (window->spin_iterations, "value-changed", window_update, NULL);
03395   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
03396   window->spin_tolerance
03397     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03398   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
03399   window->spin_bests
03400     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03401   window->label_population
03402     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
03403   window->spin_population
03404     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03405   window->label_generations
03406     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
03407   window->spin_generations
```

```
03408     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03409   window->label_mutation
03410     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
03411   window->spin_mutation
03412     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03413   window->label_reproduction
03414     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
03415   window->spin_reproduction
03416     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03417   window->label_adaptation
03418     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
03419   window->spin_adaptation
03420     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03421
03422   // Creating the array of algorithms
03423   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
03424   window->button_algorithm[0] = (GtkRadioButton *)
03425     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
03426   gtk_grid_attach (window->grid_algorithm,
03427                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
03428   g_signal_connect (window->button_algorithm[0], "clicked",
03429                     window_set_algorithm, NULL);
03430   for (i = 0; ++i < NALGORITHMS;)
03431     {
03432       window->button_algorithm[i] = (GtkRadioButton *)
03433         gtk_radio_button_new_with_mnemonic
03434         (gtk_radio_button_get_group (window->button_algorithm[0]),
03435          label_algorithm[i]);
03436       gtk_grid_attach (window->grid_algorithm,
03437                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
03438       g_signal_connect (window->button_algorithm[i], "clicked",
03439                         window_set_algorithm, NULL);
03440     }
03441   gtk_grid_attach (window->grid_algorithm,
03442                    GTK_WIDGET (window->label_simulations), 0,
03443                    NALGORITHMS, 1, 1);
03444   gtk_grid_attach (window->grid_algorithm,
03445                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
03446   gtk_grid_attach (window->grid_algorithm,
03447                    GTK_WIDGET (window->label_iterations), 0,
03448                    NALGORITHMS + 1, 1, 1);
03449   gtk_grid_attach (window->grid_algorithm,
03450                    GTK_WIDGET (window->spin_iterations), 1,
03451                    NALGORITHMS + 1, 1, 1);
03452   gtk_grid_attach (window->grid_algorithm,
03453                    GTK_WIDGET (window->label_tolerance), 0,
03454                    NALGORITHMS + 2, 1, 1);
03455   gtk_grid_attach (window->grid_algorithm,
03456                    GTK_WIDGET (window->spin_tolerance), 1,
03457                    NALGORITHMS + 2, 1, 1);
03458   gtk_grid_attach (window->grid_algorithm,
03459                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
03460   gtk_grid_attach (window->grid_algorithm,
03461                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
03462   gtk_grid_attach (window->grid_algorithm,
03463                    GTK_WIDGET (window->label_population), 0,
03464                    NALGORITHMS + 4, 1, 1);
03465   gtk_grid_attach (window->grid_algorithm,
03466                    GTK_WIDGET (window->spin_population), 1,
03467                    NALGORITHMS + 4, 1, 1);
03468   gtk_grid_attach (window->grid_algorithm,
03469                    GTK_WIDGET (window->label_generations), 0,
03470                    NALGORITHMS + 5, 1, 1);
03471   gtk_grid_attach (window->grid_algorithm,
03472                    GTK_WIDGET (window->spin_generations), 1,
03473                    NALGORITHMS + 5, 1, 1);
03474   gtk_grid_attach (window->grid_algorithm,
03475                    GTK_WIDGET (window->label_mutation), 0,
03476                    NALGORITHMS + 6, 1, 1);
03477   gtk_grid_attach (window->grid_algorithm,
03478                    GTK_WIDGET (window->spin_mutation), 1,
03479                    NALGORITHMS + 6, 1, 1);
03480   gtk_grid_attach (window->grid_algorithm,
03481                    GTK_WIDGET (window->label_reproduction), 0,
03482                    NALGORITHMS + 7, 1, 1);
03483   gtk_grid_attach (window->grid_algorithm,
03484                    GTK_WIDGET (window->spin_reproduction), 1,
03485                    NALGORITHMS + 7, 1, 1);
03486   gtk_grid_attach (window->grid_algorithm,
03487                    GTK_WIDGET (window->label_adaptation), 0,
03488                    NALGORITHMS + 8, 1, 1);
03489   gtk_grid_attach (window->grid_algorithm,
03490                    GTK_WIDGET (window->spin_adaptation), 1,
03491                    NALGORITHMS + 8, 1, 1);
03492   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
03493   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
03494                      GTK_WIDGET (window->grid_algorithm));
```

```
03495
03496   // Creating the variable widgets
03497   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
03498   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_variable),
03499                                gettext ("Variables selector"));
03500   window->id_variable = g_signal_connect
03501     (window->combo_variable, "changed", window_set_variable, NULL);
03502   window->button_add_variable
03503     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03504                                                    GTK_ICON_SIZE_BUTTON);
03505   g_signal_connect
03506     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
03507   gtk_widget_set_tooltip_text (GTK_WIDGET
03508                                (window->button_add_variable),
03509                                gettext ("Add variable"));
03510   window->button_remove_variable
03511     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03512                                                    GTK_ICON_SIZE_BUTTON);
03513   g_signal_connect
03514     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
03515   gtk_widget_set_tooltip_text (GTK_WIDGET
03516                                (window->button_remove_variable),
03517                                gettext ("Remove variable"));
03518   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
03519   window->entry_variable = (GtkEntry *) gtk_entry_new ();
03520   window->id_variable_label = g_signal_connect
03521     (window->entry_variable, "changed", window_label_variable, NULL);
03522   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
03523   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
03524     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03525   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03526   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
03527   window->scrolled_min
03528     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03529   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
03530                      GTK_WIDGET (viewport));
03531   g_signal_connect (window->spin_min, "value-changed",
03532                     window_rangemin_variable, NULL);
03533   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
03534   window->spin_max = (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_max));
03538   window->scrolled_max
03539     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03540   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
03541                      GTK_WIDGET (viewport));
03542   g_signal_connect (window->spin_max, "value-changed",
03543                     window_rangemax_variable, NULL);
03544   window->check_minabs = (GtkCheckButton *)
03545     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
03546   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
03547   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03548     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03549   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03550   gtk_container_add (GTK_CONTAINER (viewport),
03551                      GTK_WIDGET (window->spin_minabs));
03552   window->scrolled_minabs
03553     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03554   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
03555                      GTK_WIDGET (viewport));
03556   g_signal_connect (window->spin_minabs, "value-changed",
03557                     window_rangeminabs_variable, NULL);
03558   window->check_maxabs = (GtkCheckButton *)
03559     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
03560   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
03561   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03562     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03563   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03564   gtk_container_add (GTK_CONTAINER (viewport),
03565                      GTK_WIDGET (window->spin_maxabs));
03566   window->scrolled_maxabs
03567     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03568   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
03569                      GTK_WIDGET (viewport));
03570   g_signal_connect (window->spin_maxabs, "value-changed",
03571                     window_rangemaxabs_variable, NULL);
03572   window->label_precision
03573     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
03574   window->spin_precision = (GtkSpinButton *)
03575     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
03576   g_signal_connect (window->spin_precision, "value-changed",
03577                     window_precision_variable, NULL);
03578   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
03579   window->spin_sweeps
```

```
03580     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03581   g_signal_connect
03582     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
03583   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
03584   window->spin_bits
03585     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03586   g_signal_connect
03587     (window->spin_bits, "value-changed", window_update_variable, NULL);
03588   window->grid_variable = (GtkGrid *) gtk_grid_new ();
03589   gtk_grid_attach (window->grid_variable,
03590                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
03591   gtk_grid_attach (window->grid_variable,
03592                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
03593   gtk_grid_attach (window->grid_variable,
03594                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
03595   gtk_grid_attach (window->grid_variable,
03596                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
03597   gtk_grid_attach (window->grid_variable,
03598                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
03599   gtk_grid_attach (window->grid_variable,
03600                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
03601   gtk_grid_attach (window->grid_variable,
03602                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
03603   gtk_grid_attach (window->grid_variable,
03604                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
03605   gtk_grid_attach (window->grid_variable,
03606                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
03607   gtk_grid_attach (window->grid_variable,
03608                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
03609   gtk_grid_attach (window->grid_variable,
03610                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
03611   gtk_grid_attach (window->grid_variable,
03612                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
03613   gtk_grid_attach (window->grid_variable,
03614                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
03615   gtk_grid_attach (window->grid_variable,
03616                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
03617   gtk_grid_attach (window->grid_variable,
03618                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
03619   gtk_grid_attach (window->grid_variable,
03620                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
03621   gtk_grid_attach (window->grid_variable,
03622                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
03623   gtk_grid_attach (window->grid_variable,
03624                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
03625   gtk_grid_attach (window->grid_variable,
03626                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
03627   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
03628   gtk_container_add (GTK_CONTAINER (window->frame_variable),
03629                      GTK_WIDGET (window->grid_variable));
03630
03631   // Creating the experiment widgets
03632   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03633   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03634                                gettext ("Experiment selector"));
03635   window->id_experiment = g_signal_connect
03636     (window->combo_experiment, "changed", window_set_experiment, NULL)
    ;
03637   window->button_add_experiment
03638     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03639                                                    GTK_ICON_SIZE_BUTTON);
03640   g_signal_connect
03641     (window->button_add_experiment, "clicked",
    window_add_experiment, NULL);
03642   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03643                                gettext ("Add experiment"));
03644   window->button_remove_experiment
03645     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03646                                                    GTK_ICON_SIZE_BUTTON);
03647   g_signal_connect (window->button_remove_experiment, "clicked",
03648                     window_remove_experiment, NULL);
03649   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03650                                gettext ("Remove experiment"));
03651   window->label_experiment
03652     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
03653   window->button_experiment = (GtkFileChooserButton *)
03654     gtk_file_chooser_button_new (gettext ("Experimental data file"),
03655                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03656   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03657                                gettext ("Experimental data file"));
03658   window->id_experiment_name
03659     = g_signal_connect (window->button_experiment, "selection-changed",
03660                         window_name_experiment, NULL);
03661   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
03662   window->spin_weight
03663     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03664   gtk_widget_set_tooltip_text
```

```
03665     (GTK_WIDGET (window->spin_weight),
03666      gettext ("Weight factor to build the objective function"));
03667   g_signal_connect
03668     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
03669   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03670   gtk_grid_attach (window->grid_experiment,
03671                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03672   gtk_grid_attach (window->grid_experiment,
03673                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03674   gtk_grid_attach (window->grid_experiment,
03675                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03676   gtk_grid_attach (window->grid_experiment,
03677                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03678   gtk_grid_attach (window->grid_experiment,
03679                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03680   gtk_grid_attach (window->grid_experiment,
03681                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03682   gtk_grid_attach (window->grid_experiment,
03683                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03684   for (i = 0; i < MAX_NINPUTS; ++i)
03685     {
03686       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03687       window->check_template[i] = (GtkCheckButton *)
03688         gtk_check_button_new_with_label (buffer3);
03689       window->id_template[i]
03690         = g_signal_connect (window->check_template[i], "toggled",
03691                             window_inputs_experiment, NULL);
03692       gtk_grid_attach (window->grid_experiment,
03693                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03694       window->button_template[i] = (GtkFileChooserButton *)
03695         gtk_file_chooser_button_new (gettext ("Input template"),
03696                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03697       gtk_widget_set_tooltip_text
03698         (GTK_WIDGET (window->button_template[i]),
03699          gettext ("Experimental input template file"));
03700       window->id_input[i]
03701         = g_signal_connect_swapped (window->button_template[i],
03702                                     "selection-changed",
03703                                     (void (*)) window_template_experiment,
03704                                     (void *) (size_t) i);
03705       gtk_grid_attach (window->grid_experiment,
03706                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03707     }
03708   window->frame_experiment
03709     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03710   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
03711                      GTK_WIDGET (window->grid_experiment));
03712
03713   // Creating the grid and attaching the widgets to the grid
03714   window->grid = (GtkGrid *) gtk_grid_new ();
03715   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 6, 1);
03716   gtk_grid_attach (window->grid,
03717                    GTK_WIDGET (window->label_simulator), 0, 1, 1, 1);
03718   gtk_grid_attach (window->grid,
03719                    GTK_WIDGET (window->button_simulator), 1, 1, 1, 1);
03720   gtk_grid_attach (window->grid,
03721                    GTK_WIDGET (window->check_evaluator), 2, 1, 1, 1);
03722   gtk_grid_attach (window->grid,
03723                    GTK_WIDGET (window->button_evaluator), 3, 1, 1, 1);
03724   gtk_grid_attach (window->grid,
03725                    GTK_WIDGET (window->frame_algorithm), 0, 2, 2, 1);
03726   gtk_grid_attach (window->grid,
03727                    GTK_WIDGET (window->frame_variable), 2, 2, 2, 1);
03728   gtk_grid_attach (window->grid,
03729                    GTK_WIDGET (window->frame_experiment), 4, 2, 2, 1);
03730   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
03731
03732   // Setting the window logo
03733   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03734   gtk_window_set_icon (window->window, window->logo);
03735
03736   // Showing the window
03737   gtk_widget_show_all (GTK_WIDGET (window->window));
03738
03739   // In Windows the default scrolled size is wrong
03740 #ifdef G_OS_WIN32
03741   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03742   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03743   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03744   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03745 #endif
03746
03747   // Reading initial example
03748   buffer2 = g_get_current_dir ();
03749   buffer = g_build_filename (buffer2, "tests", "test1", INPUT_FILE, NULL);
```

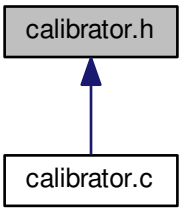
```
03750   g_free (buffer2);
03751   window_read (buffer);
03752   g_free (buffer);
03753 }
03754
03755 #endif
03756
03762 int
03763 cores_number ()
03764 {
03765 #ifdef G_OS_WIN32
03766   SYSTEM_INFO sysinfo;
03767   GetSystemInfo (&sysinfo);
03768   return sysinfo.dwNumberOfProcessors;
03769 #else
03770   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03771 #endif
03772 }
03773
03783 int
03784 main (int argn, char **argc)
03785 {
03786 #if HAVE_GTK
03787   int status;
03788 #endif
03789 #if HAVE_MPI
03790   // Starting MPI
03791   MPI_Init (&argn, &argc);
03792   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03793   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03794   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03795 #else
03796   ntasks = 1;
03797 #endif
03798   // Starting pseudo-random numbers generator
03799   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03800   calibrate->seed = DEFAULT_RANDOM_SEED;
03801   // Allowing spaces in the XML data file
03802   xmlKeepBlanksDefault (0);
03803
03804 #if HAVE_GTK
03805
03806   nthreads = cores_number ();
03807   setlocale (LC_ALL, "");
03808   setlocale (LC_NUMERIC, "C");
03809   current_directory = g_get_current_dir ();
03810   bindtextdomain
03811     (PROGRAM_INTERFACE, g_build_filename (current_directory,
      LOCALE_DIR, NULL));
03812   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03813   textdomain (PROGRAM_INTERFACE);
03814   gtk_disable_setlocale ();
03815   window->application = gtk_application_new ("git.jburguete.calibrator",
03816                                              G_APPLICATION_FLAGS_NONE);
03817   g_signal_connect (window->application, "activate", window_new, NULL);
03818   status = g_application_run (G_APPLICATION (window->application), argn, argc);
03819   g_object_unref (window->application);
03820
03821 #else
03822
03823   // Checking syntax
03824   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03825     {
03826       printf ("The syntax is:\ncalibrator [-nthreads x] data_file\n");
03827 #if HAVE_MPI
03828       // Closing MPI
03829       MPI_Finalize ();
03830 #endif
03831       return 1;
03832     }
03833   // Getting threads number
03834   if (argn == 2)
03835     nthreads = cores_number ();
03836   else
03837     nthreads = atoi (argc[2]);
03838   printf ("nthreads=%u\n", nthreads);
03839   // Making calibration
03840   if (input_open (argc[argn - 1]))
03841     calibrate_new ();
03842   // Freeing memory
03843   calibrate_free ();
03844
03845 #endif
03846
03847   // Freeing memory
03848   gsl_rng_free (calibrate->rng);
03849 #if HAVE_MPI
```

```
03850   // Closing MPI
03851   MPI_Finalize ();
03852 #endif
03853
03854 #if HAVE_GTK
03855   g_free (current_directory);
03856   return status;
03857 #else
03858   return 0;
03859 #endif
03860 }
```

5.3 calibrator.h File Reference

Header file of the calibrator.

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



Data Structures

- struct Input

  Struct to define the calibration input file.
- struct Calibrate

  Struct to define the calibration data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

  Enum to define the algorithms.

Functions

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

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

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

  Function to get an integer number of a XML node property.

- unsigned int xml_node_get_uint (xmlNode *node, const xmlChar *prop, int *error_code)

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

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

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

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

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

  Function to create a new Input struct.
- int input_open (char *filename)

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

  Function to free the memory of the input file data.
- void calibrate_input (unsigned int simulation, char *input, GMappedFile *template)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_new ()

  Function to open and perform a calibration.

5.3.1 Detailed Description

Header file of the calibrator.

Authors

Javier Burguete.

Copyright

Copyright 2012-2015, all rights reserved.

Definition in file calibrator.h.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

Definition at line 49 of file calibrator.h.

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

5.3.3 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1280 of file calibrator.c.

```
01281 {
01282   unsigned int i, j;
01283   double e;
01284 #if DEBUG
01285   fprintf (stderr, "calibrate_best_sequential: start\n");
01286 #endif
01287   if (calibrate->nsaveds < calibrate->nbest
01288       || value < calibrate->error_best[calibrate->nsaveds - 1])
01289     {
01290       if (calibrate->nsaveds < calibrate->nbest)
01291         ++calibrate->nsaveds;
01292       calibrate->error_best[calibrate->nsaveds - 1] = value;
01293       calibrate->simulation_best[calibrate->
    nsaveds - 1] = simulation;
01294       for (i = calibrate->nsaveds; --i;)
01295         {
01296           if (calibrate->error_best[i] < calibrate->
```

```
      error_best[i - 1])
01297             {
01298               j = calibrate->simulation_best[i];
01299               e = calibrate->error_best[i];
01300               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01301               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01302               calibrate->simulation_best[i - 1] = j;
01303               calibrate->error_best[i - 1] = e;
01304             }
01305           else
01306             break;
01307         }
01308     }
01309 #if DEBUG
01310   fprintf (stderr, "calibrate_best_sequential: end\n");
01311 #endif
01312 }
```

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 1235 of file calibrator.c.

```
01236 {
01237   unsigned int i, j;
01238   double e;
01239 #if DEBUG
01240   fprintf (stderr, "calibrate_best_thread: start\n");
01241 #endif
01242   if (calibrate->nsaveds < calibrate->nbest
01243       || value < calibrate->error_best[calibrate->nsaveds - 1])
01244     {
01245       g_mutex_lock (mutex);
01246       if (calibrate->nsaveds < calibrate->nbest)
01247         ++calibrate->nsaveds;
01248       calibrate->error_best[calibrate->nsaveds - 1] = value;
01249       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01250       for (i = calibrate->nsaveds; --i;)
01251         {
01252           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01253             {
01254               j = calibrate->simulation_best[i];
01255               e = calibrate->error_best[i];
01256               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01257               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01258               calibrate->simulation_best[i - 1] = j;
01259               calibrate->error_best[i - 1] = e;
01260             }
01261           else
01262             break;
01263         }
01264       g_mutex_unlock (mutex);
01265     }
01266 #if DEBUG
01267   fprintf (stderr, "calibrate_best_thread: end\n");
01268 #endif
01269 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

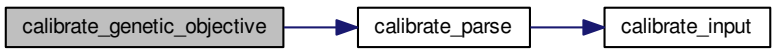
Returns

objective function value.

Definition at line 1589 of file calibrator.c.

```
01590 {
01591   unsigned int j;
01592   double objective;
01593   char buffer[64];
01594 #if DEBUG
01595   fprintf (stderr, "calibrate_genetic_objective: start\n");
01596 #endif
01597   for (j = 0; j < calibrate->nvariables; ++j)
01598     {
01599       calibrate->value[entity->id * calibrate->nvariables + j]
01600         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01601     }
01602   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01603     objective += calibrate_parse (entity->id, j);
01604   g_mutex_lock (mutex);
01605   for (j = 0; j < calibrate->nvariables; ++j)
01606     {
01607       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01608       fprintf (calibrate->file_variables, buffer,
01609                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01610     }
01611   fprintf (calibrate->file_variables, "%.14le\n", objective);
01612   g_mutex_unlock (mutex);
01613 #if DEBUG
01614   fprintf (stderr, "calibrate_genetic_objective: end\n");
01615 #endif
01616   return objective;
01617 }
```

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 984 of file calibrator.c.

```
00985 {
00986   unsigned int i;
00987   char buffer[32], value[32], *buffer2, *buffer3, *content;
00988   FILE *file;
00989   gsize length;
00990   GRegex *regex;
00991
```

```
00992 #if DEBUG
00993   fprintf (stderr, "calibrate_input: start\n");
00994 #endif
00995
00996   // Checking the file
00997   if (!template)
00998     goto calibrate_input_end;
00999
01000   // Opening template
01001   content = g_mapped_file_get_contents (template);
01002   length = g_mapped_file_get_length (template);
01003 #if DEBUG
01004   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01005            content);
01006 #endif
01007   file = fopen (input, "w");
01008
01009   // Parsing template
01010   for (i = 0; i < calibrate->nvariables; ++i)
01011     {
01012 #if DEBUG
01013       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01014 #endif
01015       snprintf (buffer, 32, "@variable%u@", i + 1);
01016       regex = g_regex_new (buffer, 0, 0, NULL);
01017       if (i == 0)
01018         {
01019           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01020                                              calibrate->label[i], 0, NULL);
01021 #if DEBUG
01022           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01023 #endif
01024         }
01025       else
01026         {
01027           length = strlen (buffer3);
01028           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01029                                              calibrate->label[i], 0, NULL);
01030           g_free (buffer3);
01031         }
01032       g_regex_unref (regex);
01033       length = strlen (buffer2);
01034       snprintf (buffer, 32, "@value%u@", i + 1);
01035       regex = g_regex_new (buffer, 0, 0, NULL);
01036       snprintf (value, 32, format[calibrate->precision[i]],
01037                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01038
01039 #if DEBUG
01040       fprintf (stderr, "calibrate_input: value=%s\n", value);
01041 #endif
01042       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01043                                          0, NULL);
01044       g_free (buffer2);
01045       g_regex_unref (regex);
01046     }
01047
01048   // Saving input file
01049   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01050   g_free (buffer3);
01051   fclose (file);
01052
01053 calibrate_input_end:
01054 #if DEBUG
01055   fprintf (stderr, "calibrate_input: end\n");
01056 #endif
01057   return;
01058 }
```

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 1396 of file calibrator.c.

```
01398 {
01399   unsigned int i, j, k, s[calibrate->nbest];
01400   double e[calibrate->nbest];
01401 #if DEBUG
01402   fprintf (stderr, "calibrate_merge: start\n");
01403 #endif
01404   i = j = k = 0;
01405   do
01406     {
01407       if (i == calibrate->nsaveds)
01408         {
01409           s[k] = simulation_best[j];
01410           e[k] = error_best[j];
01411           ++j;
01412           ++k;
01413           if (j == nsaveds)
01414             break;
01415         }
01416       else if (j == nsaveds)
01417         {
01418           s[k] = calibrate->simulation_best[i];
01419           e[k] = calibrate->error_best[i];
01420           ++i;
01421           ++k;
01422           if (i == calibrate->nsaveds)
01423             break;
01424         }
01425       else if (calibrate->error_best[i] > error_best[j])
01426         {
01427           s[k] = simulation_best[j];
01428           e[k] = error_best[j];
01429           ++j;
01430           ++k;
01431         }
01432       else
01433         {
01434           s[k] = calibrate->simulation_best[i];
01435           e[k] = calibrate->error_best[i];
01436           ++i;
01437           ++k;
01438         }
01439     }
01440   while (k < calibrate->nbest);
01441   calibrate->nsaveds = k;
01442   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01443   memcpy (calibrate->error_best, e, k * sizeof (double));
01444 #if DEBUG
01445   fprintf (stderr, "calibrate_merge: end\n");
01446 #endif
01447 }
```

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 1071 of file calibrator.c.

```
01072 {
01073   unsigned int i;
01074   double e;
01075   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
```

```
01076     *buffer3, *buffer4;
01077   FILE *file_result;
01078
01079 #if DEBUG
01080   fprintf (stderr, "calibrate_parse: start\n");
01081   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01082            experiment);
01083 #endif
01084
01085   // Opening input files
01086   for (i = 0; i < calibrate->ninputs; ++i)
01087     {
01088       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01089 #if DEBUG
01090       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01091 #endif
01092       calibrate_input (simulation, &input[i][0],
01093                        calibrate->file[i][experiment]);
01094     }
01095   for (; i < MAX_NINPUTS; ++i)
01096     strcpy (&input[i][0], "");
01097 #if DEBUG
01098   fprintf (stderr, "calibrate_parse: parsing end\n");
01099 #endif
01100
01101   // Performing the simulation
01102   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01103   buffer2 = g_path_get_dirname (calibrate->simulator);
01104   buffer3 = g_path_get_basename (calibrate->simulator);
01105   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01106   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01107             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01108             input[6], input[7], output);
01109   g_free (buffer4);
01110   g_free (buffer3);
01111   g_free (buffer2);
01112 #if DEBUG
01113   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01114 #endif
01115   system (buffer);
01116
01117   // Checking the objective value function
01118   if (calibrate->evaluator)
01119     {
01120       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01121       buffer2 = g_path_get_dirname (calibrate->evaluator);
01122       buffer3 = g_path_get_basename (calibrate->evaluator);
01123       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01124       snprintf (buffer, 512, "\"%s\" %s %s %s",
01125                 buffer4, output, calibrate->experiment[experiment], result);
01126       g_free (buffer4);
01127       g_free (buffer3);
01128       g_free (buffer2);
01129 #if DEBUG
01130       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01131 #endif
01132       system (buffer);
01133       file_result = fopen (result, "r");
01134       e = atof (fgets (buffer, 512, file_result));
01135       fclose (file_result);
01136     }
01137   else
01138     {
01139       strcpy (result, "");
01140       file_result = fopen (output, "r");
01141       e = atof (fgets (buffer, 512, file_result));
01142       fclose (file_result);
01143     }
01144
01145   // Removing files
01146 #if !DEBUG
01147   for (i = 0; i < calibrate->ninputs; ++i)
01148     {
01149       if (calibrate->file[i][0])
01150         {
01151           snprintf (buffer, 512, RM " %s", &input[i][0]);
01152           system (buffer);
01153         }
01154     }
01155   snprintf (buffer, 512, RM " %s %s", output, result);
01156   system (buffer);
01157 #endif
01158
01159 #if DEBUG
01160   fprintf (stderr, "calibrate_parse: end\n");
01161 #endif
01162
```

```
01163   // Returning the objective function
01164   return e * calibrate->weight[experiment];
01165 }
```

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 1207 of file calibrator.c.

```
01208 {
01209   unsigned int i;
01210   char buffer[64];
01211 #if DEBUG
01212   fprintf (stderr, "calibrate_save_variables: start\n");
01213 #endif
01214   for (i = 0; i < calibrate->nvariables; ++i)
01215     {
01216       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01217       fprintf (calibrate->file_variables, buffer,
01218                calibrate->value[simulation * calibrate->
      nvariables + i]);
01219     }
01220   fprintf (calibrate->file_variables, "%.14le\n", error);
01221 #if DEBUG
01222   fprintf (stderr, "calibrate_save_variables: end\n");
01223 #endif
01224 }
```

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

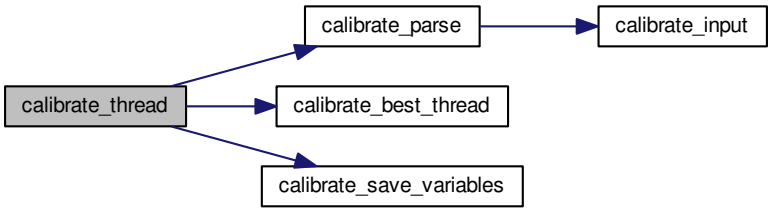
NULL

Definition at line 1322 of file calibrator.c.

```
01323 {
01324   unsigned int i, j, thread;
01325   double e;
01326 #if DEBUG
```

```
01327   fprintf (stderr, "calibrate_thread: start\n");
01328 #endif
01329   thread = data->thread;
01330 #if DEBUG
01331   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01332            calibrate->thread[thread], calibrate->thread[thread + 1]);
01333 #endif
01334   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01335     {
01336       e = 0.;
01337       for (j = 0; j < calibrate->nexperiments; ++j)
01338         e += calibrate_parse (i, j);
01339       calibrate_best_thread (i, e);
01340       g_mutex_lock (mutex);
01341       calibrate_save_variables (i, e);
01342       g_mutex_unlock (mutex);
01343 #if DEBUG
01344       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01345 #endif
01346     }
01347 #if DEBUG
01348   fprintf (stderr, "calibrate_thread: end\n");
01349 #endif
01350   g_thread_exit (NULL);
01351   return NULL;
01352 }
```

Here is the call graph for this function:



5.3.3.9 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 458 of file calibrator.c.

```
00459 {
00460   int error_code;
00461   unsigned int i;
00462   char buffer2[64];
00463   xmlChar *buffer;
00464   xmlDoc *doc;
00465   xmlNode *node, *child;
00466
00467 #if DEBUG
00468   fprintf (stderr, "input_new: start\n");
00469 #endif
00470
00471   // Resetting input data
```

```
00472   input_new ();
00473
00474   // Parsing the input file
00475   doc = xmlParseFile (filename);
00476   if (!doc)
00477     {
00478       show_error (gettext ("Unable to parse the input file"));
00479       return 0;
00480     }
00481
00482   // Getting the root node
00483   node = xmlDocGetRootElement (doc);
00484   if (xmlStrcmp (node->name, XML_CALIBRATE))
00485     {
00486       show_error (gettext ("Bad root XML node"));
00487       return 0;
00488     }
00489
00490   // Opening simulator program name
00491   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00492   if (!input->simulator)
00493     {
00494       show_error (gettext ("Bad simulator program"));
00495       return 0;
00496     }
00497
00498   // Opening evaluator program name
00499   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00500
00501   // Obtaining pseudo-random numbers generator seed
00502   if (!xmlHasProp (node, XML_SEED))
00503     input->seed = DEFAULT_RANDOM_SEED;
00504   else
00505     {
00506       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00507       if (error_code)
00508         {
00509           show_error (gettext ("Bad pseudo-random numbers generator seed"));
00510           return 0;
00511         }
00512     }
00513
00514   // Opening algorithm
00515   buffer = xmlGetProp (node, XML_ALGORITHM);
00516   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00517     {
00518       input->algorithm = ALGORITHM_MONTE_CARLO;
00519
00520       // Obtaining simulations number
00521       input->nsimulations
00522         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00523       if (error_code)
00524         {
00525           show_error (gettext ("Bad simulations number"));
00526           return 0;
00527         }
00528     }
00529   else if (!xmlStrcmp (buffer, XML_SWEEP))
00530     input->algorithm = ALGORITHM_SWEEP;
00531   else if (!xmlStrcmp (buffer, XML_GENETIC))
00532     {
00533       input->algorithm = ALGORITHM_GENETIC;
00534
00535       // Obtaining population
00536       if (xmlHasProp (node, XML_NPOPULATION))
00537         {
00538           input->nsimulations
00539             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00540           if (error_code || input->nsimulations < 3)
00541             {
00542               show_error (gettext ("Invalid population number"));
00543               return 0;
00544             }
00545         }
00546       else
00547         {
00548           show_error (gettext ("No population number"));
00549           return 0;
00550         }
00551
00552       // Obtaining generations
00553       if (xmlHasProp (node, XML_NGENERATIONS))
00554         {
00555           input->niterations
00556             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00557           if (error_code || !input->niterations)
00558             {
```

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

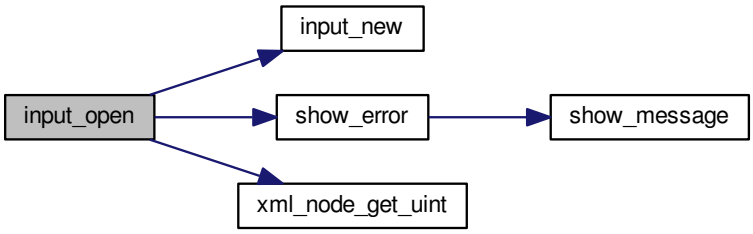
```
00644
00645       // Obtaining iterations number
00646       input->niterations
00647         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00648       if (error_code == 1)
00649         input->niterations = 1;
00650       else if (error_code)
00651         {
00652           show_error (gettext ("Bad iterations number"));
00653           return 0;
00654         }
00655
00656       // Obtaining best number
00657       if (xmlHasProp (node, XML_NBEST))
00658         {
00659           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00660           if (error_code || !input->nbest)
00661             {
00662               show_error (gettext ("Invalid best number"));
00663               return 0;
00664             }
00665         }
00666       else
00667         input->nbest = 1;
00668
00669       // Obtaining tolerance
00670       if (xmlHasProp (node, XML_TOLERANCE))
00671         {
00672           input->tolerance
00673             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00674           if (error_code || input->tolerance < 0.)
00675             {
00676               show_error (gettext ("Invalid tolerance"));
00677               return 0;
00678             }
00679         }
00680       else
00681         input->tolerance = 0.;
00682     }
00683
00684   // Reading the experimental data
00685   for (child = node->children; child; child = child->next)
00686     {
00687       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00688         break;
00689 #if DEBUG
00690       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00691 #endif
00692       if (xmlHasProp (child, XML_NAME))
00693         {
00694           input->experiment
00695             = g_realloc (input->experiment,
00696                          (1 + input->nexperiments) * sizeof (char *));
00697           input->experiment[input->nexperiments]
00698             = (char *) xmlGetProp (child, XML_NAME);
00699         }
00700       else
00701         {
00702           show_error (gettext ("No experiment file name"));
00703           return 0;
00704         }
00705 #if DEBUG
00706       fprintf (stderr, "input_new: experiment=%s\n",
00707                input->experiment[input->nexperiments]);
00708 #endif
00709       input->weight = g_realloc (input->weight,
00710                                  (1 + input->nexperiments) * sizeof (double));
00711       if (xmlHasProp (child, XML_WEIGHT))
00712         input->weight[input->nexperiments]
00713           = xml_node_get_float (child, XML_WEIGHT, &error_code);
00714       else
00715         input->weight[input->nexperiments] = 1.;
00716 #if DEBUG
00717       fprintf (stderr, "input_new: weight=%lg\n",
00718                input->weight[input->nexperiments]);
00719 #endif
00720       if (!input->nexperiments)
00721         input->ninputs = 0;
00722 #if DEBUG
00723       fprintf (stderr, "input_new: template[0]\n");
00724 #endif
00725       if (xmlHasProp (child, XML_TEMPLATE1))
00726         {
00727           input->template[0]
00728             = (char **) g_realloc (input->template[0],
00729                                    (1 + input->nexperiments) * sizeof (char *));
```

```
00730           input->template[0][input->nexperiments]
00731             = (char *) xmlGetProp (child, template[0]);
00732 #if DEBUG
00733           fprintf (stderr, "input_new: experiment=%u template1=%s\n",
00734                    input->nexperiments,
00735                    input->template[0][input->nexperiments]);
00736 #endif
00737           if (!input->nexperiments)
00738             ++input->ninputs;
00739 #if DEBUG
00740           fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00741 #endif
00742         }
00743       else
00744         {
00745           show_error (gettext ("No experiment template"));
00746           return 0;
00747         }
00748       for (i = 1; i < MAX_NINPUTS; ++i)
00749         {
00750 #if DEBUG
00751           fprintf (stderr, "input_new: template%u\n", i + 1);
00752 #endif
00753           if (xmlHasProp (child, template[i]))
00754             {
00755               if (input->nexperiments && input->ninputs < 2)
00756                 {
00757                   snprintf (buffer2, 64,
00758                             gettext ("Experiment %u: bad templates number"),
00759                             input->nexperiments + 1);
00760                   show_error (buffer2);
00761                   return 0;
00762                 }
00763               input->template[i] = (char **)
00764                 g_realloc (input->template[i],
00765                            (1 + input->nexperiments) * sizeof (char *));
00766               input->template[i][input->nexperiments]
00767                 = (char *) xmlGetProp (child, template[i]);
00768 #if DEBUG
00769               fprintf (stderr, "input_new: experiment=%u template%u=%s\n",
00770                        input->nexperiments, i + 1,
00771                        input->template[i][input->nexperiments]);
00772 #endif
00773               if (!input->nexperiments)
00774                 ++input->ninputs;
00775 #if DEBUG
00776               fprintf (stderr, "input_new: ninputs=%u\n", input->ninputs);
00777 #endif
00778             }
00779           else if (input->nexperiments && input->ninputs > 1)
00780             {
00781               snprintf (buffer2, 64, gettext ("No experiment %u template%u"),
00782                         input->nexperiments + 1, i + 1);
00783               show_error (buffer2);
00784               return 0;
00785             }
00786           else
00787             break;
00788         }
00789       ++input->nexperiments;
00790 #if DEBUG
00791       fprintf (stderr, "input_new: nexperiments=%u\n", input->nexperiments);
00792 #endif
00793     }
00794   if (!input->nexperiments)
00795     {
00796       show_error (gettext ("No calibration experiments"));
00797       return 0;
00798     }
00799
00800   // Reading the variables data
00801   for (; child; child = child->next)
00802     {
00803       if (xmlStrcmp (child->name, XML_VARIABLE))
00804         {
00805           show_error (gettext ("Bad XML node"));
00806           return 0;
00807         }
00808       if (xmlHasProp (child, XML_NAME))
00809         {
00810           input->label = g_realloc
00811             (input->label, (1 + input->nvariables) * sizeof (char *));
00812           input->label[input->nvariables]
00813             = (char *) xmlGetProp (child, XML_NAME);
00814         }
00815       else
00816         {
```

```
00817           show_error (gettext ("No variable name"));
00818           return 0;
00819         }
00820       if (xmlHasProp (child, XML_MINIMUM))
00821         {
00822           input->rangemin = g_realloc
00823             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00824           input->rangeminabs = g_realloc
00825             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00826           input->rangemin[input->nvariables]
00827             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00828           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00829             {
00830               input->rangeminabs[input->nvariables]
00831                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00832             }
00833           else
00834             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00835         }
00836       else
00837         {
00838           show_error (gettext ("No minimum range"));
00839           return 0;
00840         }
00841       if (xmlHasProp (child, XML_MAXIMUM))
00842         {
00843           input->rangemax = g_realloc
00844             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00845           input->rangemaxabs = g_realloc
00846             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00847           input->rangemax[input->nvariables]
00848             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00849           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00850             input->rangemaxabs[input->nvariables]
00851               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00852           else
00853             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00854         }
00855       else
00856         {
00857           show_error (gettext ("No maximum range"));
00858           return 0;
00859         }
00860       input->precision = g_realloc
00861         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00862       if (xmlHasProp (child, XML_PRECISION))
00863         input->precision[input->nvariables]
00864           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00865       else
00866         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00867       if (input->algorithm == ALGORITHM_SWEEP)
00868         {
00869           if (xmlHasProp (child, XML_NSWEEPS))
00870             {
00871               input->nsweeps = (unsigned int *)
00872                 g_realloc (input->nsweeps,
00873                            (1 + input->nvariables) * sizeof (unsigned int));
00874               input->nsweeps[input->nvariables]
00875                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00876             }
00877           else
00878             {
00879               show_error (gettext ("No sweeps number"));
00880               return 0;
00881             }
00882 #if DEBUG
00883           fprintf (stderr, "input_new: nsweeps=%u nsimulations=%u\n",
00884                    input->nsweeps[input->nvariables],
      input->nsimulations);
00885 #endif
00886         }
00887       if (input->algorithm == ALGORITHM_GENETIC)
00888         {
00889           // Obtaining bits representing each variable
00890           if (xmlHasProp (child, XML_NBITS))
00891             {
00892               input->nbits = (unsigned int *)
00893                 g_realloc (input->nbits,
00894                            (1 + input->nvariables) * sizeof (unsigned int));
00895               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00896               if (error_code || !i)
00897                 {
00898                   show_error (gettext ("Invalid bit number"));
00899                   return 0;
```

```
00900                 }
00901               input->nbits[input->nvariables] = i;
00902             }
00903           else
00904             {
00905               show_error (gettext ("No bits number"));
00906               return 0;
00907             }
00908         }
00909       ++input->nvariables;
00910     }
00911   if (!input->nvariables)
00912     {
00913       show_error (gettext ("No calibration variables"));
00914       return 0;
00915     }
00916
00917   // Getting the working directory
00918   input->directory = g_path_get_dirname (filename);
00919   input->name = g_path_get_basename (filename);
00920
00921   // Closing the XML document
00922   xmlFreeDoc (doc);
00923
00924 #if DEBUG
00925   fprintf (stderr, "input_new: end\n");
00926 #endif
00927
00928   return 1;
00929 }
```

Here is the call graph for this function:



5.3.3.10 void show_error (char * *msg*)

Function to show a dialog with an error message.

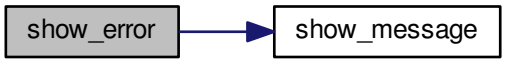
Parameters

msg	Error message.

Definition at line 273 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 243 of file calibrator.c.

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

5.3.3.12 double xml_node_get_float (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

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

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 352 of file calibrator.c.

```
00353 {
```

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

5.3.3.13 int xml_node_get_int (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Integer number value.

Definition at line 290 of file calibrator.c.

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

5.3.3.14 unsigned int xml_node_get_uint (xmlNode * *node*, const xmlChar * *prop*, int * *error_code*)

Function to get an unsigned integer number of a XML node property.

Parameters

node	XML node.
prop	XML property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 321 of file calibrator.c.

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

5.3.3.15 void xml_node_set_float (xmlNode * *node*, const xmlChar * *prop*, double *value*)

Function to set a floating point number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Floating point number value.

Definition at line 419 of file calibrator.c.

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

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

Function to set an integer number in a XML node property.

Parameters

node	XML node.
prop	XML property.
value	Integer number value.

Definition at line 381 of file calibrator.c.

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

5.3.3.17 void xml_node_set_uint (xmlNode * *node*, const xmlChar * *prop*, unsigned int *value*)

Function to set an unsigned integer number in a XML node property.

Parameters

node	XML node.

prop	XML property.
value	Unsigned integer number value.

Definition at line 400 of file calibrator.c.

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

5.4 calibrator.h

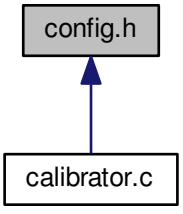
```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef CALIBRATOR__H
00037 #define CALIBRATOR__H 1
00038
00049 enum Algorithm
00050 {
00051   ALGORITHM_MONTE_CARLO = 0,
00052   ALGORITHM_SWEEP = 1,
00053   ALGORITHM_GENETIC = 2
00054 };
00055
00060 typedef struct
00061 {
00118   char *simulator, *evaluator, **experiment, **template[MAX_NINPUTS], **label,
00119     *directory, *name;
00120   double *rangemin, *rangemax, *rangeminabs, *rangemaxabs, *weight, tolerance,
00121     mutation_ratio, reproduction_ratio, adaptation_ratio;
00122   unsigned int nvariables, nexperiments, ninputs, nsimulations, algorithm,
00123     *precision, *nsweeps, *nbits, niterations, nbest;
00124   unsigned long int seed;
00125 } Input;
00126
00131 typedef struct
00132 {
00213   char *simulator, *evaluator, **experiment, **template[MAX_NINPUTS], **label;
00214   unsigned int nvariables, nexperiments, ninputs, nsimulations, algorithm,
00215     *precision, *nsweeps, nstart, nend, *thread, niterations, nbest, nsaveds,
00216     *simulation_best;
00217   unsigned long int seed;
00218   double *value, *rangemin, *rangemax, *rangeminabs, *rangemaxabs, *error_best,
00219     *weight, *value_old, *error_old, tolerance, mutation_ratio,
00220     reproduction_ratio, adaptation_ratio;
00221   FILE *file_result, *file_variables;
00222   gsl_rng *rng;
00223   GMappedFile **file[MAX_NINPUTS];
00224   GeneticVariable *genetic_variable;
00225 #if HAVE_MPI
00226   int mpi_rank;
00227 #endif
00228 } Calibrate;
```

```
00229
00234 typedef struct
00235 {
00240   unsigned int thread;
00241 } ParallelData;
00242
00243 // Public functions
00244 void show_message (char *title, char *msg, int type);
00245 void show_error (char *msg);
00246 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00247 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00248                                 int *error_code);
00249 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00250                            int *error_code);
00251 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00252 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00253                         unsigned int value);
00254 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00255 void input_new ();
00256 int input_open (char *filename);
00257 void input_free ();
00258 void calibrate_input (unsigned int simulation, char *input,
00259                       GMappedFile * template);
00260 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00261 void calibrate_print ();
00262 void calibrate_save_variables (unsigned int simulation, double error);
00263 void calibrate_best_thread (unsigned int simulation, double value);
00264 void calibrate_best_sequential (unsigned int simulation, double value);
00265 void *calibrate_thread (ParallelData * data);
00266 void calibrate_sequential ();
00267 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00268                       double *error_best);
00269 #if HAVE_MPI
00270 void calibrate_synchronise ();
00271 #endif
00272 void calibrate_sweep ();
00273 void calibrate_MonteCarlo ();
00274 double calibrate_genetic_objective (Entity * entity);
00275 void calibrate_genetic ();
00276 void calibrate_save_old ();
00277 void calibrate_merge_old ();
00278 void calibrate_refine ();
00279 void calibrate_iterate ();
00280 void calibrate_new ();
00281
00282 #endif
```

5.5 config.h File Reference

Configuration header file.

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



Macros

- #define MAX_NINPUTS 8

Maximum number of input files in the simulator program.

- #define NALGORITHMS 3

  Number of algorithms.
- #define NPRECISIONS 15

  Number of precisions.
- #define DEFAULT_ALGORITHM "Monte-Carlo"

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

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

  Macro to set the default pseudo-random numbers seed.
- #define LOCALE_DIR "locales"

  Locales directory.
- #define PROGRAM_INTERFACE "calibrator"

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  npopulation XML label.

- #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
  nsimulations XML label.
- #define XML_NSWEEPS (const xmlChar*)"nsweeps"
  nsweeps XML label.
- #define XML_PRECISION (const xmlChar*)"precision"
  precision XML label.
- #define XML_REPRODUCTION (const xmlChar*)"reproduction"
  reproduction XML label.
- #define XML_SIMULATOR (const xmlChar*)"simulator"
  simulator XML label.
- #define XML_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

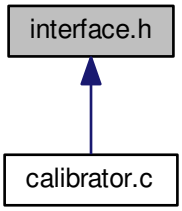
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary 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
00048 #define MAX_NINPUTS 8
00049 #define NALGORITHMS 3
00050 #define NPRECISIONS 15
00051
00052 // Default choices
00053
00062 #define DEFAULT_ALGORITHM "Monte-Carlo"
00063 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00064 #define DEFAULT_RANDOM_SEED 7007
00065
00066 // Interface labels
00067
00074 #define LOCALE_DIR "locales"
00075 #define PROGRAM_INTERFACE "calibrator"
00076
00077 // XML labels
00078
00153 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00154 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00155 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00156 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00157 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00158 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00159 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00160 #define XML_GENETIC (const xmlChar*)"genetic"
00161 #define XML_MINIMUM (const xmlChar*)"minimum"
00162 #define XML_MAXIMUM (const xmlChar*)"maximum"
00163 #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"
00164 #define XML_MUTATION (const xmlChar*)"mutation"
00165 #define XML_NAME (const xmlChar*)"name"
00166 #define XML_NBEST (const xmlChar*)"nbest"
00167 #define XML_NBITS (const xmlChar*)"nbits"
00168 #define XML_NGENERATIONS (const xmlChar*)"ngenerations"
00169 #define XML_NITERATIONS (const xmlChar*)"niterations"
00170 #define XML_NPOPULATION (const xmlChar*)"npopulation"
00171 #define XML_NSIMULATIONS (const xmlChar*)"nsimulations"
00172 #define XML_NSWEEPS (const xmlChar*)"nsweeps"
00173 #define XML_PRECISION (const xmlChar*)"precision"
00174 #define XML_REPRODUCTION (const xmlChar*)"reproduction"
00175 #define XML_SIMULATOR (const xmlChar*)"simulator"
00176 #define XML_SEED (const xmlChar*)"seed"
00177 #define XML_SWEEP (const xmlChar*)"sweep"
00178 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00179 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00180 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00181 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00182 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00183 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00184 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00185 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00186 #define XML_TOLERANCE (const xmlChar*)"tolerance"
```

```
00187 #define XML_VARIABLE (const xmlChar*)"variable"
00188 #define XML_WEIGHT (const xmlChar*)"weight"
00189
00190 #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

Copyright 2012-2015, all rights reserved.

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 3763 of file calibrator.c.

```
03764 {
03765 #ifdef G_OS_WIN32
03766   SYSTEM_INFO sysinfo;
03767   GetSystemInfo (&sysinfo);
03768   return sysinfo.dwNumberOfProcessors;
03769 #else
03770   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03771 #endif
03772 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

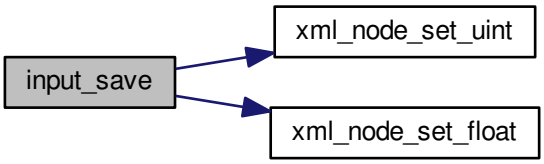
Definition at line 2089 of file calibrator.c.

```
02090 {
02091   unsigned int i, j;
02092   char *buffer;
02093   xmlDoc *doc;
02094   xmlNode *node, *child;
02095   GFile *file, *file2;
02096
02097   // Getting the input file directory
02098   input->name = g_path_get_basename (filename);
02099   input->directory = g_path_get_dirname (filename);
02100   file = g_file_new_for_path (input->directory);
02101
02102   // Opening the input file
02103   doc = xmlNewDoc ((const xmlChar *) "1.0");
02104
02105   // Setting root XML node
02106   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02107   xmlDocSetRootElement (doc, node);
02108
02109   // Adding properties to the root XML node
02110   file2 = g_file_new_for_path (input->simulator);
```

```
02111   buffer = g_file_get_relative_path (file, file2);
02112   g_object_unref (file2);
02113   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02114   g_free (buffer);
02115   if (input->evaluator)
02116     {
02117       file2 = g_file_new_for_path (input->evaluator);
02118       buffer = g_file_get_relative_path (file, file2);
02119       g_object_unref (file2);
02120       if (xmlStrlen ((xmlChar *) buffer))
02121         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02122       g_free (buffer);
02123     }
02124   if (input->seed != DEFAULT_RANDOM_SEED)
02125     xml_node_set_uint (node, XML_SEED, input->seed);
02126
02127   // Setting the algorithm
02128   buffer = (char *) g_malloc (64);
02129   switch (input->algorithm)
02130     {
02131     case ALGORITHM_MONTE_CARLO:
02132       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02133       snprintf (buffer, 64, "%u", input->nsimulations);
02134       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02135       snprintf (buffer, 64, "%u", input->niterations);
02136       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02137       snprintf (buffer, 64, "%.3lg", input->tolerance);
02138       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02139       snprintf (buffer, 64, "%u", input->nbest);
02140       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02141       break;
02142     case ALGORITHM_SWEEP:
02143       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02144       snprintf (buffer, 64, "%u", input->niterations);
02145       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02146       snprintf (buffer, 64, "%.3lg", input->tolerance);
02147       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02148       snprintf (buffer, 64, "%u", input->nbest);
02149       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02150       break;
02151     default:
02152       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02153       snprintf (buffer, 64, "%u", input->nsimulations);
02154       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02155       snprintf (buffer, 64, "%u", input->niterations);
02156       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02157       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02158       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02159       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02160       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02161       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02162       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02163       break;
02164     }
02165   g_free (buffer);
02166
02167   // Setting the experimental data
02168   for (i = 0; i < input->nexperiments; ++i)
02169     {
02170       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02171       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02172       if (input->weight[i] != 1.)
02173         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02174       for (j = 0; j < input->ninputs; ++j)
02175         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02176     }
02177
02178   // Setting the variables data
02179   for (i = 0; i < input->nvariables; ++i)
02180     {
02181       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02182       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02183       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02184       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02185         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02186       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02187       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02188         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02189       if (input->precision[i] != DEFAULT_PRECISION)
02190         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02191       if (input->algorithm == ALGORITHM_SWEEP)
```

```
02192         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02193       else if (input->algorithm == ALGORITHM_GENETIC)
02194         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02195     }
02196
02197   // Saving the XML file
02198   xmlSaveFormatFile (filename, doc, 1);
02199
02200   // Freeing memory
02201   xmlFreeDoc (doc);
02202 }
```

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 2453 of file calibrator.c.

```
02454 {
02455   unsigned int i;
02456   for (i = 0; i < NALGORITHMS; ++i)
02457     if (gtk_toggle_button_get_active
02458         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02459       break;
02460   return i;
02461 }
```

5.7.2.4 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

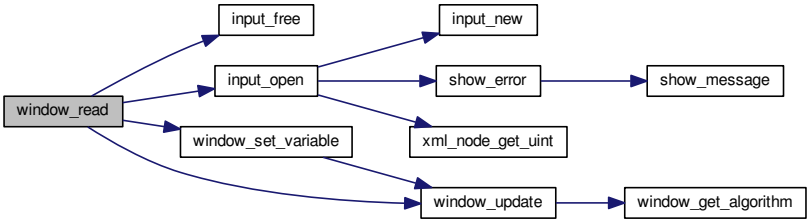
1 on succes, 0 on error.

Definition at line 3169 of file calibrator.c.

```
03170 {
03171   unsigned int i;
03172   char *buffer;
03173 #if DEBUG
03174   fprintf (stderr, "window_read: start\n");
03175 #endif
03176   input_free ();
03177   if (!input_open (filename))
03178     {
03179 #if DEBUG
03180       fprintf (stderr, "window_read: end\n");
03181 #endif
03182       return 0;
03183     }
03184   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03185   puts (buffer);
03186   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03187                                  (window->button_simulator), buffer);
03188   g_free (buffer);
03189   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03190                                 (size_t) input->evaluator);
03191   if (input->evaluator)
03192     {
03193       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03194       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03195                                      (window->button_evaluator), buffer);
03196       g_free (buffer);
03197     }
03198   gtk_toggle_button_set_active
03199     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03200   switch (input->algorithm)
03201     {
03202     case ALGORITHM_MONTE_CARLO:
03203       gtk_spin_button_set_value (window->spin_simulations,
03204                                  (gdouble) input->nsimulations);
03205     case ALGORITHM_SWEEP:
03206       gtk_spin_button_set_value (window->spin_iterations,
03207                                  (gdouble) input->niterations);
03208       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03209       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03210       break;
03211     default:
03212       gtk_spin_button_set_value (window->spin_population,
03213                                  (gdouble) input->nsimulations);
03214       gtk_spin_button_set_value (window->spin_generations,
03215                                  (gdouble) input->niterations);
03216       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03217       gtk_spin_button_set_value (window->spin_reproduction,
03218                                  input->reproduction_ratio);
03219       gtk_spin_button_set_value (window->spin_adaptation,
03220                                  input->adaptation_ratio);
03221     }
03222   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03223   g_signal_handler_block (window->button_experiment,
03224                           window->id_experiment_name);
03225   gtk_combo_box_text_remove_all (window->combo_experiment);
03226   for (i = 0; i < input->nexperiments; ++i)
03227     gtk_combo_box_text_append_text (window->combo_experiment,
03228                                     input->experiment[i]);
03229   g_signal_handler_unblock
03230     (window->button_experiment, window->
      id_experiment_name);
03231   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03232   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03233   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03234   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03235   gtk_combo_box_text_remove_all (window->combo_variable);
03236   for (i = 0; i < input->nvariables; ++i)
03237     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03238   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03239   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03240   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03241   window_set_variable ();
03242   window_update ();
```

```
03243 #if DEBUG
03244   fprintf (stderr, "window_read: end\n");
03245 #endif
03246   return 1;
03247 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

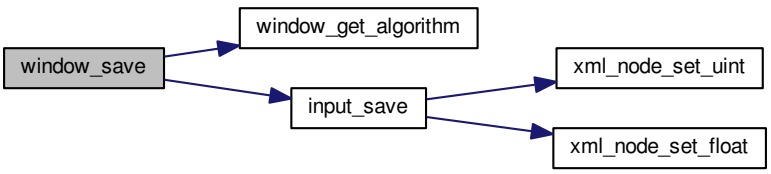
1 on OK, 0 on Cancel.

Definition at line 2272 of file calibrator.c.

```
02273 {
02274   char *buffer;
02275   GtkFileChooserDialog *dlg;
02276
02277 #if DEBUG
02278   fprintf (stderr, "window_save: start\n");
02279 #endif
02280
02281   // Opening the saving dialog
02282   dlg = (GtkFileChooserDialog *)
02283     gtk_file_chooser_dialog_new (gettext ("Save file"),
02284                                  window->window,
02285                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02286                                  gettext ("_Cancel"),
02287                                  GTK_RESPONSE_CANCEL,
02288                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02289   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02290
02291   // If OK response then saving
02292   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02293     {
02294
02295       // Adding properties to the root XML node
02296       input->simulator = gtk_file_chooser_get_filename
02297         (GTK_FILE_CHOOSER (window->button_simulator));
02298       if (gtk_toggle_button_get_active
02299           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02300         input->evaluator = gtk_file_chooser_get_filename
02301           (GTK_FILE_CHOOSER (window->button_evaluator));
02302       else
02303         input->evaluator = NULL;
02304
02305       // Setting the algorithm
02306       switch (window_get_algorithm ())
02307         {
02308         case ALGORITHM_MONTE_CARLO:
02309           input->algorithm = ALGORITHM_MONTE_CARLO;
02310           input->nsimulations
02311             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02312           input->niterations
```

```
02313             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02314           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02315           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02316           break;
02317         case ALGORITHM_SWEEP:
02318           input->algorithm = ALGORITHM_SWEEP;
02319           input->niterations
02320             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02321           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02322           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02323           break;
02324         default:
02325           input->algorithm = ALGORITHM_GENETIC;
02326           input->nsimulations
02327             = gtk_spin_button_get_value_as_int (window->spin_population);
02328           input->niterations
02329             = gtk_spin_button_get_value_as_int (window->spin_generations);
02330           input->mutation_ratio
02331             = gtk_spin_button_get_value (window->spin_mutation);
02332           input->reproduction_ratio
02333             = gtk_spin_button_get_value (window->spin_reproduction);
02334           input->adaptation_ratio
02335             = gtk_spin_button_get_value (window->spin_adaptation);
02336           break;
02337         }
02338
02339       // Saving the XML file
02340       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02341       input_save (buffer);
02342
02343       // Closing and freeing memory
02344       g_free (buffer);
02345       gtk_widget_destroy (GTK_WIDGET (dlg));
02346 #if DEBUG
02347       fprintf (stderr, "window_save: end\n");
02348 #endif
02349       return 1;
02350     }
02351
02352   // Closing and freeing memory
02353   gtk_widget_destroy (GTK_WIDGET (dlg));
02354 #if DEBUG
02355   fprintf (stderr, "window_save: end\n");
02356 #endif
02357   return 0;
02358 }
```

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 2827 of file calibrator.c.

```
02828 {
02829   unsigned int i, j;
02830   char *buffer;
02831   GFile *file1, *file2;
02832 #if DEBUG
02833   fprintf (stderr, "window_template_experiment: start\n");
02834 #endif
02835   i = (size_t) data;
02836   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02837   file1
02838     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02839   file2 = g_file_new_for_path (input->directory);
02840   buffer = g_file_get_relative_path (file2, file1);
02841   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02842   g_free (buffer);
02843   g_object_unref (file2);
02844   g_object_unref (file1);
02845 #if DEBUG
02846   fprintf (stderr, "window_template_experiment: end\n");
02847 #endif
02848 }
```

5.8 interface.h

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef INTERFACE__H
00037 #define INTERFACE__H 1
00038
00043 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00044
00049 typedef struct
00050 {
00059   char *name, *template[MAX_NINPUTS];
00060   double weight;
00061 } Experiment;
00062
00067 typedef struct
00068 {
00087   char *label;
00088   double rangemin, rangemax, rangeminabs, rangemaxabs;
00089   unsigned int precision, nsweeps, nbits;
00090 } Variable;
00091
00096 typedef struct
00097 {
00112   GtkLabel *label_processors, *label_seed;
00113   GtkSpinButton *spin_processors, *spin_seed;
00114   GtkGrid *grid;
```

```
00115   GtkDialog *dialog;
00116 } Options;
00117
00122 typedef struct
00123 {
00130   GtkLabel *label;
00131   GtkDialog *dialog;
00132 } Running;
00133
00138 typedef struct
00139 {
00308   GtkToolButton *button_open, *button_save, *button_run, *button_options,
00309     *button_help, *button_about, *button_exit;
00310   GtkButton *button_add_variable, *button_remove_variable,
00311     *button_add_experiment, *button_remove_experiment;
00312   GtkRadioButton *button_algorithm[NALGORITHMS];
00313   GtkCheckButton *check_evaluator, *check_minabs, *check_maxabs,
00314     *check_template[MAX_NINPUTS];
00315   GtkLabel *label_simulator, *label_simulations, *label_iterations,
00316     *label_tolerance, *label_bests, *label_population, *label_generations,
00317     *label_mutation, *label_reproduction, *label_adaptation, *label_variable,
00318     *label_min, *label_max, *label_precision, *label_sweeps, *label_bits,
00319     *label_experiment, *label_weight;
00320   GtkEntry *entry_variable;
00321   GtkComboBoxText *combo_variable, *combo_experiment;
00322   GtkFileChooserButton *button_simulator, *button_evaluator, *button_experiment,
00323     *button_template[MAX_NINPUTS];
00324   GtkSpinButton *spin_min, *spin_max, *spin_minabs, *spin_maxabs,
00325     *spin_simulations, *spin_iterations, *spin_tolerance, *spin_bests,
00326     *spin_population, *spin_generations, *spin_mutation, *spin_reproduction,
00327     *spin_adaptation, *spin_precision, *spin_sweeps, *spin_bits, *spin_weight;
00328   GtkToolbar *bar_buttons;
00329   GtkGrid *grid, *grid_algorithm, *grid_variable, *grid_experiment;
00330   GtkFrame *frame_algorithm, *frame_variable, *frame_experiment;
00331   GdkPixbuf *logo;
00332   GtkScrolledWindow *scrolled_min, *scrolled_max, *scrolled_minabs,
00333     *scrolled_maxabs;
00334   GtkWindow *window;
00335   GtkApplication *application;
00336   Experiment *experiment;
00337   Variable *variable;
00338   gulong id_experiment, id_experiment_name, id_variable, id_variable_label,
00339     id_template[MAX_NINPUTS], id_input[MAX_NINPUTS];
00340   unsigned int nexperiments, nvariables;
00341 } Window;
00342
00343 // Public functions
00344 void input_save (char *filename);
00345 void options_new ();
00346 void running_new ();
00347 int window_save ();
00348 void window_run ();
00349 void window_help ();
00350 int window_get_algorithm ();
00351 void window_update ();
00352 void window_set_algorithm ();
00353 void window_set_experiment ();
00354 void window_remove_experiment ();
00355 void window_add_experiment ();
00356 void window_name_experiment ();
00357 void window_weight_experiment ();
00358 void window_inputs_experiment ();
00359 void window_template_experiment (void *data);
00360 void window_set_variable ();
00361 void window_remove_variable ();
00362 void window_add_variable ();
00363 void window_label_variable ();
00364 void window_precision_variable ();
00365 void window_rangemin_variable ();
00366 void window_rangemax_variable ();
00367 void window_rangeminabs_variable ();
00368 void window_rangemaxabs_variable ();
00369 void window_update_variable ();
00370 int window_read (char *filename);
00371 void window_open ();
00372 void window_new ();
00373 int cores_number ();
00374
00375 #endif
```

Index