Calibrator

1.0.2

Generated by Doxygen 1.8.8

Thu Nov 12 2015 21:36:07

Contents

Chapter 1

CALIBRATOR

A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 1.0.2: Stable and recommended version.
- 1.1.34: Developing version to do new features.

AUTHORS

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

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

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

OPTIONAL TOOLS AND LIBRARIES

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

FILES

The source code has to have the following files:

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

BUILDING INSTRUCTIONS

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

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

FreeBSD 10.2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

1. Download the latest `genetic` doing on a terminal:

   $ git clone `https://github.com/jburguete/genetic.git`

2. Download this repository:

   $ git clone `https://github.com/jburguete/calibrator.git`

3. Link the latest genetic version to genetic:

```
$ cd calibrator/1.0.2
$ ln -s ../../genetic/0.6.1 genetic
```

4. Build doing on a terminal:

```
$ ./build
```

OpenBSD 5.8

1. Select adequate versions:

```
$ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15
```

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

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

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

```
$ make windist
```

Fedora Linux 23

1. In order to use OpenMPI compilation do in a terminal (in 64 bits version):

```
$ export PATH=$PATH:/usr/lib64/openmpi/bin
```

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

MAKING MANUALS INSTRUCTIONS

On UNIX type systems you need texlive installed. On Windows systems you need MiKTeX. In order to compile the manuals you can type on a terminal:

```
$ make manuals
```

USER INSTRUCTIONS

- Command line in sequential mode:

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

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

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

- The syntax of the simulator has to be:

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

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

```
$ ./evaluator_name simulated_file data_file results_file
```

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

```
$ ./calibrator
```

INPUT FILE FORMAT

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

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

Implemented algorithms are:

- ∗"sweep"∗: Sweep brutal force algorithm. Requires for each variable:

  sweeps: number of sweeps to generate for each variable in every experiment.

  The total number of simulations to run is:

  (number of experiments) x (variable 1 number of sweeps) x ... x (variable n number of sweeps) x (number of iterations)

- ∗"Monte-Carlo"∗: Monte-Carlo brutal force algorithm. Requires on calibrate:

  nsimulations: number of simulations to run in every experiment.

  The total number of simulations to run is:

  (number of experiments) x (number of simulations) x (number of iterations)

- Both brutal force algorithms can be iterated to improve convergence by using the following parameters:

  nbest: number of best simulations to calculate convergence interval on next iteration (default 1).
  tolerance: tolerance parameter to increase convergence interval (default 0).
  niterations: number of iterations (default 1).

- ∗"genetic"∗: Genetic algorithm. Requires the following parameters:

  npopulation: number of population.
  ngenerations: number of generations.
  mutation: mutation ratio.
  reproduction: reproduction ratio.
  adaptation: adaptation ratio.

  and for each variable:

  nbits: number of bits to encode each variable.

  The total number of simulations to run is:

  (number of experiments) x (npopulation) x [1 + (ngenerations - 1) x (mutation + reproduction + adaptation)]

SOME EXAMPLES OF INPUT FILES

Example 1

- The simulator program name is: *pivot*
- The syntax is:

  $ ./pivot input_file output_file

- The program to evaluate the objective function is: *compare*
- The syntax is:

  $ ./compare simulated_file data_file result_file

- The calibration is performed with a *sweep brutal force algorithm*.
- The experimental data files are:

  27-48.txt
  42.txt
  52.txt
  100.txt

- Templates to get input files to simulator for each experiment are:

  template1.js
  template2.js
  template3.js
  template4.js

- The variables to calibrate, ranges, c-string format and sweeps number to perform are:

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

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

_

```
<?xml version="1.0"?>
<calibrate simulator="pivot" evaluator="compare" algorithm="sweep">
    <experiment name="27-48.txt" template1="template1.js"/>
    <experiment name="42.txt" template1="template2.js"/>
    <experiment name="52.txt" template1="template3.js"/>
    <experiment name="100.txt" template1="template4.js"/>
    <variable name="alpha1" minimum="179.70" maximum="180.20" format="%.2lf" nsweeps="5"/>
    <variable name="alpha2" minimum="179.30" maximum="179.60" format="%.2lf" nsweeps="5"/>
    <variable name="random" minimum="0.00" maximum="0.20" format="%.2lf" nsweeps="5"/>
    <variable name="boot-time" minimum="0.0" maximum="3.0" format="%.1lf" nsweeps="5"/>
</calibrate>
```

- A template file as *template1.js*:

_

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

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

—

```
{
  "towers" :
  [
    {
      "length"      : 50.11,
      "velocity"    : 0.02738,
      "alpha1" : 179.95,
      "alpha2" : 179.45,
      "random" : 0.10,
      "boot-time" : 1.5
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.02824,
      "alpha1" : 179.95,
      "alpha2" : 179.45,
      "random" : 0.10,
      "boot-time" : 1.5
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.03008,
      "alpha1" : 179.95,
      "alpha2" : 179.45,
      "random" : 0.10,
      "boot-time" : 1.5
```

```
    },
    {
      "length"    : 50.11,
      "velocity"  : 0.03753,
      "alpha1" : 179.95,
      "alpha2" : 179.45,
      "random" : 0.10,
      "boot-time" : 1.5
    }
  ],
  "cycle-time"     : 71.0,
  "plot-time"     : 1.0,
  "comp-time-step": 0.1,
  "active-percent" : 27.48
}
```

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Calibrate Struct Reference

Struct to define the calibration data.

`#include <calibrator.h>`

Data Fields

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

Array of simulation numbers to calculate on the thread.

- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- unsigned int * simulation_best

  Array of best simulation numbers.
- unsigned long int seed

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

  Array of variable values.
- double * rangemin

  Array of minimum variable values.
- double * rangemax

  Array of maximum variable values.
- double * rangeminabs

  Array of absolute minimum variable values.
- double * rangemaxabs

  Array of absolute maximum variable values.
- double * error_best

  Array of the best minimum errors.
- double * weight

  Array of the experiment weights.
- double * value_old

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

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

  Algorithm tolerance.
- double mutation_ratio

  Mutation probability.
- double reproduction_ratio

  Reproduction probability.
- double adaptation_ratio

  Adaptation probability.
- FILE * file_result

  Result file.
- FILE * file_variables

  Variables file.
- gsl_rng * rng

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

  Matrix of input template files.
- GeneticVariable * genetic_variable

  Array of variables for the genetic algorithm.
- int mpi_rank

  Number of MPI task.

4.1.1 Detailed Description

Struct to define the calibration data.

Definition at line 92 of file calibrator.h.

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

- calibrator.h

4.2 Experiment Struct Reference

Struct to define experiment data.

`#include <interface.h>`

Data Fields

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

4.2.1 Detailed Description

Struct to define experiment data.

Definition at line 46 of file interface.h.

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

- interface.h

4.3 Input Struct Reference

Struct to define the calibration input file.

`#include <calibrator.h>`

Data Fields

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

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

4.3.1 Detailed Description

Struct to define the calibration input file.

Definition at line 54 of file calibrator.h.

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

- calibrator.h

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_processors

  Processors number GtkLabel.
- GtkSpinButton * spin_processors

  Processors number GtkSpinButton.
- GtkLabel * label_seed

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

  Pseudo-random numbers generator seed GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 74 of file interface.h.

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

- interface.h

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <calibrator.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 147 of file calibrator.h.

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

- calibrator.h

4.6 Running Struct Reference

Struct to define the running dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkLabel * label

  Label GtkLabel.

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 90 of file interface.h.

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

- interface.h

4.7 Variable Struct Reference

Struct to define variable data.

`#include <interface.h>`

Data Fields

- char * label

  Variable label.
- double rangemin

  Minimum value.
- double rangemax

  Maximum value.
- double rangeminabs

  Minimum allowed value.
- double rangemaxabs

  Maximum allowed value.
- unsigned int precision

  Precision digits.
- unsigned int nsweeps

  Sweeps number of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define variable data.

Definition at line 58 of file interface.h.

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

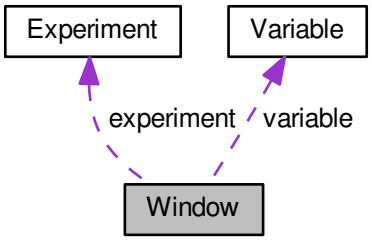
- interface.h

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:



Data Fields

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

Exit GtkToolButton.

- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkFrame * frame_algorithm

  GtkFrame to set the algorithm.
- GtkGrid * grid_algorithm

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

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

  GtkLabel to set the simulations number.
- GtkSpinButton * spin_simulations

  GtkSpinButton to set the simulations number.
- GtkLabel * label_iterations

  GtkLabel to set the iterations number.
- GtkSpinButton * spin_iterations

  GtkSpinButton to set the iterations number.
- GtkLabel * label_tolerance

  GtkLabel to set the tolerance.
- GtkSpinButton * spin_tolerance

  GtkSpinButton to set the tolerance.
- GtkLabel * label_bests

  GtkLabel to set the best number.
- GtkSpinButton * spin_bests

  GtkSpinButton to set the best number.
- GtkLabel * label_population

  GtkLabel to set the population number.
- GtkSpinButton * spin_population

  GtkSpinButton to set the population number.
- GtkLabel * label_generations

  GtkLabel to set the generations number.
- GtkSpinButton * spin_generations

  GtkSpinButton to set the generations number.
- GtkLabel * label_mutation

  GtkLabel to set the mutation ratio.
- GtkSpinButton * spin_mutation

  GtkSpinButton to set the mutation ratio.
- GtkLabel * label_reproduction

  GtkLabel to set the reproduction ratio.
- GtkSpinButton * spin_reproduction

  GtkSpinButton to set the reproduction ratio.
- GtkLabel * label_adaptation

  GtkLabel to set the adaptation ratio.
- GtkSpinButton * spin_adaptation

  GtkSpinButton to set the adaptation ratio.

- GtkFrame * frame_variable

  Variable GtkFrame.
- GtkGrid * grid_variable

  Variable GtkGrid.
- GtkComboBoxText * combo_variable

  GtkComboBoxEntry to select a variable.
- GtkButton * button_add_variable

  GtkButton to add a variable.
- GtkButton * button_remove_variable

  GtkButton to remove a variable.
- GtkLabel * label_variable

  Variable GtkLabel.
- GtkEntry * entry_variable

  GtkEntry to set the variable name.
- GtkLabel * label_min

  Minimum GtkLabel.
- GtkSpinButton * spin_min

  Minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_min

  Minimum GtkScrolledWindow.
- GtkLabel * label_max

  Maximum GtkLabel.
- GtkSpinButton * spin_max

  Maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_max

  Maximum GtkScrolledWindow.
- GtkCheckButton * check_minabs

  Absolute minimum GtkCheckButton.
- GtkSpinButton * spin_minabs

  Absolute minimum GtkSpinButton.
- GtkScrolledWindow * scrolled_minabs

  Absolute minimum GtkScrolledWindow.
- GtkCheckButton * check_maxabs

  Absolute maximum GtkCheckButton.
- GtkSpinButton * spin_maxabs

  Absolute maximum GtkSpinButton.
- GtkScrolledWindow * scrolled_maxabs

  Absolute maximum GtkScrolledWindow.
- GtkLabel * label_precision

  Precision GtkLabel.
- GtkSpinButton * spin_precision

  Precision digits GtkSpinButton.
- GtkLabel * label_sweeps

  Sweeps number GtkLabel.
- GtkSpinButton * spin_sweeps

  Sweeps number GtkSpinButton.
- GtkLabel * label_bits

  Bits number GtkLabel.
- GtkSpinButton * spin_bits

  Bits number GtkSpinButton.
- GtkFrame * frame_experiment

Experiment GtkFrame.

- GtkGrid * grid_experiment

  Experiment GtkGrid.
- GtkComboBoxText * combo_experiment

  Experiment GtkComboBoxEntry.
- GtkButton * button_add_experiment

  GtkButton to add a experiment.
- GtkButton * button_remove_experiment

  GtkButton to remove a experiment.
- GtkLabel * label_experiment

  Experiment GtkLabel.
- GtkFileChooserButton * button_experiment

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

  Weight GtkLabel.
- GtkSpinButton * spin_weight

  Weight GtkSpinButton.
- GtkCheckButton * check_template [MAX_NINPUTS]

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

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

  Logo GdkPixbuf.
- Experiment * experiment

  Array of experiments data.
- Variable * variable

  Array of variables data.
- char * application_directory

  Application directory.
- gulong id_experiment

  Identifier of the combo_experiment signal.
- gulong id_experiment_name

  Identifier of the button_experiment signal.
- gulong id_variable

  Identifier of the combo_variable signal.
- gulong id_variable_label

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

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

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

  Number of experiments.
- unsigned int nvariables

  Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 100 of file interface.h.

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

- interface.h

Chapter 5

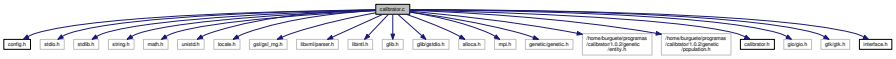
File Documentation

5.1 calibrator.c File Reference

Source file of the calibrator.

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

Include dependency graph for calibrator.c:



Macros

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

  Macro to debug.
- #define ERROR_TYPE GTK_MESSAGE_ERROR

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

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

Macro to define the initial input file.

- #define RM "rm"

  Macro to define the shell remove command.

Functions

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

  Function to show a dialog with a message.

- void show_error (char *msg)

  Function to show a dialog with an error message.

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

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

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

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

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

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

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

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

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

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

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

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

- void input_new ()

  Function to create a new Input struct.

- void input_free ()

  Function to free the memory of the input file data.

- int input_open (char *filename)

  Function to open the input file.

- void calibrate_input (unsigned int simulation, char *input, GMappedFile *template)

  Function to write the simulation input file.

- double calibrate_parse (unsigned int simulation, unsigned int experiment)

  Function to parse input files, simulating and calculating the \ objective function.

- void calibrate_print ()

  Function to print the results.

- void calibrate_save_variables (unsigned int simulation, double error)

  Function to save in a file the variables and the error.

- void calibrate_best_thread (unsigned int simulation, double value)

  Function to save the best simulations of a thread.

- void calibrate_best_sequential (unsigned int simulation, double value)

  Function to save the best simulations.

- void * calibrate_thread (ParallelData *data)

  Function to calibrate on a thread.

- void calibrate_sequential ()

  Function to calibrate sequentially.

- void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 calibration results.

- void calibrate_synchronise ()

  Function to synchronise the calibration results of MPI tasks.

- void calibrate_sweep ()

  Function to calibrate with the sweep algorithm.

- void calibrate_MonteCarlo ()

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

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

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

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

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

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

  Function to iterate the algorithm.
- void calibrate_free ()

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

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

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

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

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

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

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

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

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

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

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

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

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

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

- void window_set_variable ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Main function.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- GMutex mutex [1]

  Mutex struct.
- void(* calibrate_step)()

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

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

  Calibration data.
- const xmlChar * template [MAX_NINPUTS]

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

  Array of C-strings with variable formats.

- const double precision [NPRECISIONS]

  Array of variable precisions.
- const char * logo []

  Logo pixmap.
- Options options [1]

  Options struct to define the options dialog.
- Running running [1]

  Running struct to define the running dialog.
- Window window [1]

  Window struct to define the main interface window.

5.1.1 Detailed Description

Source file of the calibrator.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file calibrator.c.

5.1.2 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1330 of file calibrator.c.

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

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

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

Function to save the best simulations of a thread.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1285 of file calibrator.c.

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

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

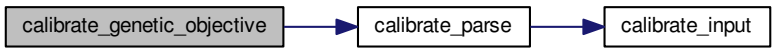
Returns

objective function value.

Definition at line 1639 of file calibrator.c.

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

Here is the call graph for this function:



5.1.2.4 void calibrate_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 1034 of file calibrator.c.

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

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

5.1.2.5 void calibrate_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 calibration results.

Parameters

nsaveds	Number of saved results.
simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1446 of file calibrator.c.

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

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

5.1.2.6 double calibrate_parse (unsigned int *simulation*, unsigned int *experiment*)

Function to parse input files, simulating and calculating the \ objective function.

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1121 of file calibrator.c.

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

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

Here is the call graph for this function:

5.1.2.7 void calibrate_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1257 of file calibrator.c.

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

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

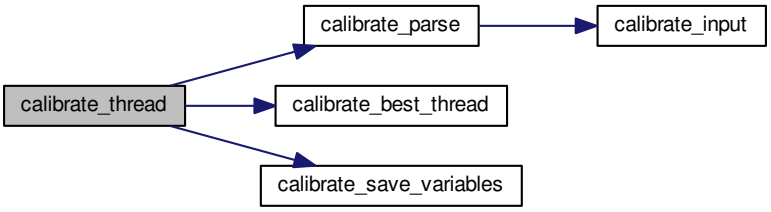
Returns

NULL

Definition at line 1372 of file calibrator.c.

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

Here is the call graph for this function:



5.1.2.9 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 3904 of file calibrator.c.

```
03905 {
03906 #ifdef G_OS_WIN32
03907   SYSTEM_INFO sysinfo;
03908   GetSystemInfo (&sysinfo);
03909   return sysinfo.dwNumberOfProcessors;
03910 #else
03911   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03912 #endif
03913 }
```

5.1.2.10 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 472 of file calibrator.c.

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

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

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

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

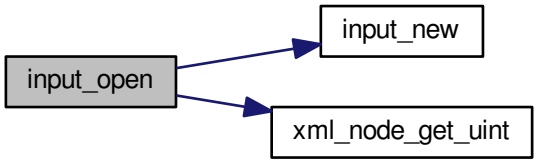
```
00745                 input->weight[input->nexperiments]);
00746 #endif
00747       if (!input->nexperiments)
00748         input->ninputs = 0;
00749 #if DEBUG
00750       fprintf (stderr, "input_open: template[0]\n");
00751 #endif
00752       if (xmlHasProp (child, XML_TEMPLATE1))
00753        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

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

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

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

Here is the call graph for this function:



5.1.2.11 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

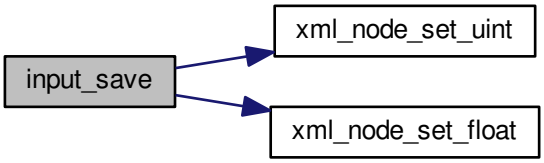
Definition at line 2142 of file calibrator.c.

```
02143 {
02144   unsigned int i, j;
02145   char *buffer;
02146   xmlDoc *doc;
02147   xmlNode *node, *child;
02148   GFile *file, *file2;
02149
02150   // Getting the input file directory
02151   input->name = g_path_get_basename (filename);
02152   input->directory = g_path_get_dirname (filename);
02153   file = g_file_new_for_path (input->directory);
02154
02155   // Opening the input file
02156   doc = xmlNewDoc ((const xmlChar *) "1.0");
02157
02158   // Setting root XML node
02159   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02160   xmlDocSetRootElement (doc, node);
02161
02162   // Adding properties to the root XML node
02163   file2 = g_file_new_for_path (input->simulator);
02164   buffer = g_file_get_relative_path (file, file2);
02165   g_object_unref (file2);
02166   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02167   g_free (buffer);
02168   if (input->evaluator)
02169     {
```

```
02170       file2 = g_file_new_for_path (input->evaluator);
02171       buffer = g_file_get_relative_path (file, file2);
02172       g_object_unref (file2);
02173       if (xmlStrlen ((xmlChar *) buffer))
02174         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02175       g_free (buffer);
02176     }
02177   if (input->seed != DEFAULT_RANDOM_SEED)
02178     xml_node_set_uint (node, XML_SEED, input->seed);
02179
02180   // Setting the algorithm
02181   buffer = (char *) g_malloc (64);
02182   switch (input->algorithm)
02183     {
02184     case ALGORITHM_MONTE_CARLO:
02185       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02186       snprintf (buffer, 64, "%u", input->nsimulations);
02187       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02188       snprintf (buffer, 64, "%u", input->niterations);
02189       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%.3lg", input->tolerance);
02191       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%u", input->nbest);
02193       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02194       break;
02195     case ALGORITHM_SWEEP:
02196       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02197       snprintf (buffer, 64, "%u", input->niterations);
02198       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02199       snprintf (buffer, 64, "%.3lg", input->tolerance);
02200       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%u", input->nbest);
02202       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02203       break;
02204     default:
02205       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02206       snprintf (buffer, 64, "%u", input->nsimulations);
02207       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02208       snprintf (buffer, 64, "%u", input->niterations);
02209       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02211       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02213       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02215       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02216       break;
02217     }
02218   g_free (buffer);
02219
02220   // Setting the experimental data
02221   for (i = 0; i < input->nexperiments; ++i)
02222     {
02223       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02224       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02225       if (input->weight[i] != 1.)
02226         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02227       for (j = 0; j < input->ninputs; ++j)
02228         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02229     }
02230
02231   // Setting the variables data
02232   for (i = 0; i < input->nvariables; ++i)
02233     {
02234       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02235       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02236       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02237       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02238         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02239       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02240       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02241         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02242       if (input->precision[i] != DEFAULT_PRECISION)
02243         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02244       if (input->algorithm == ALGORITHM_SWEEP)
02245         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02246       else if (input->algorithm == ALGORITHM_GENETIC)
02247         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02248     }
```

```
02249
02250   // Saving the XML file
02251   xmlSaveFormatFile (filename, doc, 1);
02252
02253   // Freeing memory
02254   xmlFreeDoc (doc);
02255 }
```

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

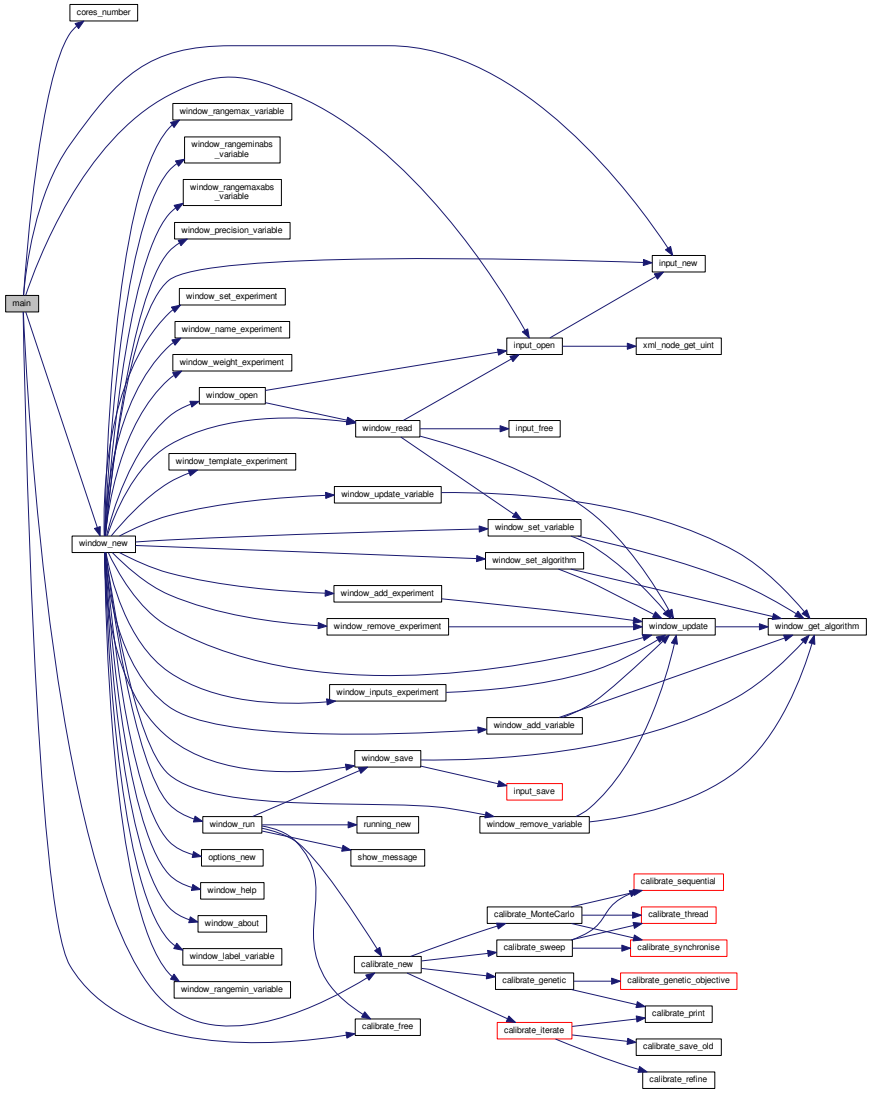
0 on success, >0 on error.

Definition at line 3925 of file calibrator.c.

```
03926 {
03927   // Starting pseudo-random numbers generator
03928   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03929   calibrate->seed = DEFAULT_RANDOM_SEED;
03930
03931   // Allowing spaces in the XML data file
03932   xmlKeepBlanksDefault (0);
03933
03934   // Starting MPI
03935 #if HAVE_MPI
03936   MPI_Init (&argn, &argc);
03937   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03938   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03939   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03940 #else
03941   ntasks = 1;
03942 #endif
03943
03944 #if HAVE_GTK
03945
03946   // Getting threads number
03947   nthreads = cores_number ();
03948
03949   // Setting local language and international floating point numbers notation
03950   setlocale (LC_ALL, "");
03951   setlocale (LC_NUMERIC, "C");
03952   window->application_directory = g_get_current_dir ();
03953   bindtextdomain (PROGRAM_INTERFACE,
03954                   g_build_filename (window->application_directory,
03955                                     LOCALE_DIR, NULL));
03956   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
```

```
03957   textdomain (PROGRAM_INTERFACE);
03958
03959   // Initing GTK+
03960   gtk_disable_setlocale ();
03961   gtk_init (&argn, &argc);
03962
03963   // Opening the main window
03964   window_new ();
03965   gtk_main ();
03966
03967   // Freeing memory
03968   gtk_widget_destroy (GTK_WIDGET (window->window));
03969   g_free (window->application_directory);
03970
03971 #else
03972
03973   // Checking syntax
03974   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03975     {
03976       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03977       return 1;
03978     }
03979
03980   // Getting threads number
03981   if (argn == 2)
03982     nthreads = cores_number ();
03983   else
03984     nthreads = atoi (argc[2]);
03985   printf ("nthreads=%u\n", nthreads);
03986
03987   // Making calibration
03988   input_new ();
03989   if (input_open (argc[argn - 1]))
03990     calibrate_new ();
03991
03992   // Freeing memory
03993   calibrate_free ();
03994
03995 #endif
03996
03997   // Closing MPI
03998 #if HAVE_MPI
03999   MPI_Finalize ();
04000 #endif
04001
04002   // Freeing memory
04003   gsl_rng_free (calibrate->rng);
04004
04005   // Closing
04006   return 0;
04007 }
```

Here is the call graph for this function:



5.1.2.13 void show_error (char * *msg*)

Function to show a dialog with an error message.

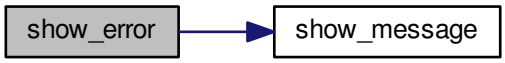
Parameters

msg	Error message.

Definition at line 246 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 216 of file calibrator.c.

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

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2515 of file calibrator.c.

```
02516 {
02517   unsigned int i;
02518   for (i = 0; i < NALGORITHMS; ++i)
02519     if (gtk_toggle_button_get_active
02520         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02521       break;
02522   return i;
02523 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

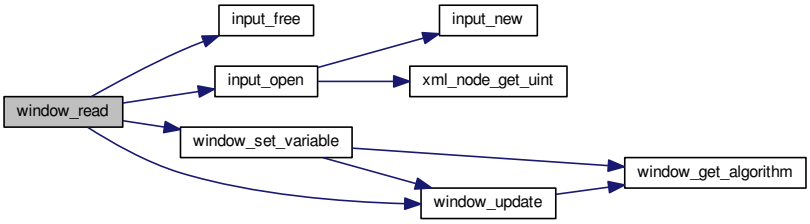
1 on succes, 0 on error.

Definition at line 3266 of file calibrator.c.

```
03267 {
03268   unsigned int i;
03269   char *buffer;
03270 #if DEBUG
03271   fprintf (stderr, "window_read: start\n");
03272 #endif
03273
03274   // Reading new input file
03275   input_free ();
03276   if (!input_open (filename))
03277     return 0;
03278
03279   // Setting GTK+ widgets data
03280   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03282                                  (window->button_simulator), buffer);
03283   g_free (buffer);
03284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03285                                 (size_t) input->evaluator);
03286   if (input->evaluator)
03287     {
03288       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03290                                      (window->button_evaluator), buffer);
03291       g_free (buffer);
03292     }
03293   gtk_toggle_button_set_active
03294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03295   switch (input->algorithm)
03296     {
03297     case ALGORITHM_MONTE_CARLO:
03298       gtk_spin_button_set_value (window->spin_simulations,
03299                                  (gdouble) input->nsimulations);
03300     case ALGORITHM_SWEEP:
03301       gtk_spin_button_set_value (window->spin_iterations,
03302                                  (gdouble) input->niterations);
03303       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03304       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03305       break;
03306     default:
03307       gtk_spin_button_set_value (window->spin_population,
03308                                  (gdouble) input->nsimulations);
03309       gtk_spin_button_set_value (window->spin_generations,
03310                                  (gdouble) input->niterations);
03311       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03312       gtk_spin_button_set_value (window->spin_reproduction,
03313                                  input->reproduction_ratio);
03314       gtk_spin_button_set_value (window->spin_adaptation,
03315                                  input->adaptation_ratio);
03316     }
03317   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03318   g_signal_handler_block (window->button_experiment,
03319                           window->id_experiment_name);
03320   gtk_combo_box_text_remove_all (window->combo_experiment);
03321   for (i = 0; i < input->nexperiments; ++i)
03322     gtk_combo_box_text_append_text (window->combo_experiment,
03323                                     input->experiment[i]);
03324   g_signal_handler_unblock
03325     (window->button_experiment, window->
      id_experiment_name);
03326   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03327   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03328   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03329   g_signal_handler_block (window->entry_variable, window->
```

```
      id_variable_label);
03330   gtk_combo_box_text_remove_all (window->combo_variable);
03331   for (i = 0; i < input->nvariables; ++i)
03332     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03333   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03334   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03335   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03336   window_set_variable ();
03337   window_update ();
03338
03339 #if DEBUG
03340   fprintf (stderr, "window_read: end\n");
03341 #endif
03342   return 1;
03343 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

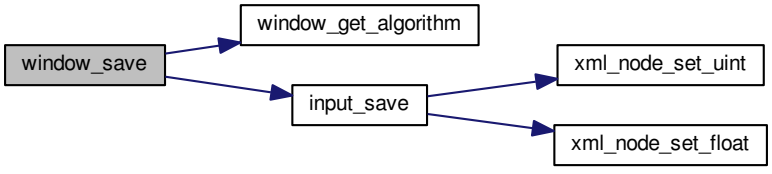
1 on OK, 0 on Cancel.

Definition at line 2330 of file calibrator.c.

```
02331 {
02332   char *buffer;
02333   GtkFileChooserDialog *dlg;
02334
02335 #if DEBUG
02336   fprintf (stderr, "window_save: start\n");
02337 #endif
02338
02339   // Opening the saving dialog
02340   dlg = (GtkFileChooserDialog *)
02341     gtk_file_chooser_dialog_new (gettext ("Save file"),
02342                                  window->window,
02343                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02344                                  gettext ("_Cancel"),
02345                                  GTK_RESPONSE_CANCEL,
02346                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02347   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02348   buffer = g_build_filename (input->directory, input->name, NULL);
02349   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02350   g_free (buffer);
02351
02352   // If OK response then saving
02353   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02354     {
02355
02356       // Adding properties to the root XML node
02357       input->simulator = gtk_file_chooser_get_filename
```

```
02358         (GTK_FILE_CHOOSER (window->button_simulator));
02359       if (gtk_toggle_button_get_active
02360           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02361         input->evaluator = gtk_file_chooser_get_filename
02362           (GTK_FILE_CHOOSER (window->button_evaluator));
02363       else
02364         input->evaluator = NULL;
02365
02366       // Setting the algorithm
02367       switch (window_get_algorithm ())
02368         {
02369         case ALGORITHM_MONTE_CARLO:
02370           input->algorithm = ALGORITHM_MONTE_CARLO;
02371           input->nsimulations
02372             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02373           input->niterations
02374             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02375           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02376           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02377           break;
02378         case ALGORITHM_SWEEP:
02379           input->algorithm = ALGORITHM_SWEEP;
02380           input->niterations
02381             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02382           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02383           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02384           break;
02385         default:
02386           input->algorithm = ALGORITHM_GENETIC;
02387           input->nsimulations
02388             = gtk_spin_button_get_value_as_int (window->spin_population);
02389           input->niterations
02390             = gtk_spin_button_get_value_as_int (window->spin_generations);
02391           input->mutation_ratio
02392             = gtk_spin_button_get_value (window->spin_mutation);
02393           input->reproduction_ratio
02394             = gtk_spin_button_get_value (window->spin_reproduction);
02395           input->adaptation_ratio
02396             = gtk_spin_button_get_value (window->spin_adaptation);
02397           break;
02398         }
02399
02400       // Saving the XML file
02401       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02402       input_save (buffer);
02403
02404       // Closing and freeing memory
02405       g_free (buffer);
02406       gtk_widget_destroy (GTK_WIDGET (dlg));
02407 #if DEBUG
02408       fprintf (stderr, "window_save: end\n");
02409 #endif
02410       return 1;
02411     }
02412
02413   // Closing and freeing memory
02414   gtk_widget_destroy (GTK_WIDGET (dlg));
02415 #if DEBUG
02416   fprintf (stderr, "window_save: end\n");
02417 #endif
02418   return 0;
02419 }
```

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 2902 of file calibrator.c.

```
02903 {
02904   unsigned int i, j;
02905   char *buffer;
02906   GFile *file1, *file2;
02907 #if DEBUG
02908   fprintf (stderr, "window_template_experiment: start\n");
02909 #endif
02910   i = (size_t) data;
02911   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02912   file1
02913     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02914   file2 = g_file_new_for_path (input->directory);
02915   buffer = g_file_get_relative_path (file2, file1);
02916   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02917   g_free (buffer);
02918   g_object_unref (file2);
02919   g_object_unref (file1);
02920 #if DEBUG
02921   fprintf (stderr, "window_template_experiment: end\n");
02922 #endif
02923 }
```

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 325 of file calibrator.c.

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

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

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

Parameters

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

Returns

Integer number value.

Definition at line 263 of file calibrator.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 294 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 392 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 354 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 373 of file calibrator.c.

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

5.1.3 Variable Documentation

5.1.3.1 const char* format[NPRECISIONS]

Initial value:

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

Array of C-strings with variable formats.

Definition at line 107 of file calibrator.c.

5.1.3.2 const double precision[NPRECISIONS]

Initial value:

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

Array of variable precisions.

Definition at line 112 of file calibrator.c.

5.1.3.3 const xmlChar* template[MAX_NINPUTS]

Initial value:

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

Array of xmlChar strings with template labels.

Definition at line 100 of file calibrator.c.

5.2 calibrator.c

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

```
00051 #elif (!__BSD_VISIBLE)
00052 #include <alloca.h>
00053 #endif
00054 #if HAVE_MPI
00055 #include <mpi.h>
00056 #endif
00057 #include "genetic/genetic.h"
00058 #include "calibrator.h"
00059 #if HAVE_GTK
00060 #include <gio/gio.h>
00061 #include <gtk/gtk.h>
00062 #include "interface.h"
00063 #endif
00064
00065 #define DEBUG 0
00066
00076 #if HAVE_GTK
00077 #define ERROR_TYPE GTK_MESSAGE_ERROR
00078 #define INFO_TYPE GTK_MESSAGE_INFO
00079 #else
00080 #define ERROR_TYPE 0
00081 #define INFO_TYPE 0
00082 #endif
00083 #ifdef G_OS_WIN32
00084 #define INPUT_FILE "test-ga-win.xml"
00085 #define RM "del"
00086 #else
00087 #define INPUT_FILE "test-ga.xml"
00088 #define RM "rm"
00089 #endif
00090
00091 int ntasks;
00092 unsigned int nthreads;
00093 GMutex mutex[1];
00094 void (*calibrate_step) ();
00096 Input input[1];
00098 Calibrate calibrate[1];
00099
00100 const xmlChar *template[MAX_NINPUTS] = {
00101   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00102   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
00103 };
00104
00106
00107 const char *format[NPRECISIONS] = {
00108   "%.1lg", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00109   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00110 };
00111
00112 const double precision[NPRECISIONS] = {
00113   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00114   1e-13, 1e-14
00115 };
00116
00117 const char *logo[] = {
00118   "32 32 3 1",
00119   "     c None",
00120   ".    c #0000FF",
00121   "+    c #FF0000",
00122   "                                ",
00123   "                                ",
00124   "                                ",
00125   "     .      .      .      .     ",
00126   "     .      .      .      .     ",
00127   "     .      .      .      .     ",
00128   "     .      .      .      .     ",
00129   "     .      .     +++     .     ",
00130   "     .      .    +++++    .     ",
00131   "     .      .    +++++    .     ",
00132   "     .      .    +++++    .     ",
00133   "    +++     .     +++    +++    ",
00134   "   +++++    .      .    +++++   ",
00135   "   +++++    .      .    +++++   ",
00136   "   +++++    .      .    +++++   ",
00137   "    +++     .      .     +++    ",
00138   "     .      .      .      .     ",
00139   "     .     +++     .      .     ",
00140   "     .    +++++    .      .     ",
00141   "     .    +++++    .      .     ",
00142   "     .    +++++    .      .     ",
00143   "     .     +++     .      .     ",
00144   "     .      .      .      .     ",
00145   "     .      .      .      .     ",
00146   "     .      .      .      .     ",
00147   "     .      .      .      .     ",
```

```
00148   "     .      .      .      .     ",
00149   "     .      .      .      .     ",
00150   "     .      .      .      .     ",
00151   "                                ",
00152   "                                ",
00153   "                                "
00154 };
00155
00156 /*
00157 const char * logo[] = {
00158 "32 32 3 1",
00159 "     c #FFFFFFFFFFFF",
00160 ".    c #00000000FFFF",
00161 "X    c #FFFF00000000",
00162 "                                ",
00163 "                                ",
00164 "                                ",
00165 "     .      .      .      .     ",
00166 "     .      .      .      .     ",
00167 "     .      .      .      .     ",
00168 "     .      .      .      .     ",
00169 "     .      .     XXX     .     ",
00170 "     .      .    XXXXX    .     ",
00171 "     .      .    XXXXX    .     ",
00172 "     .      .    XXXXX    .     ",
00173 "    XXX     .     XXX    XXX    ",
00174 "   XXXXX    .      .    XXXXX   ",
00175 "   XXXXX    .      .    XXXXX   ",
00176 "   XXXXX    .      .    XXXXX   ",
00177 "    XXX     .      .     XXX    ",
00178 "     .      .      .      .     ",
00179 "     .     XXX     .      .     ",
00180 "     .    XXXXX    .      .     ",
00181 "     .    XXXXX    .      .     ",
00182 "     .    XXXXX    .      .     ",
00183 "     .     XXX     .      .     ",
00184 "     .      .      .      .     ",
00185 "     .      .      .      .     ",
00186 "     .      .      .      .     ",
00187 "     .      .      .      .     ",
00188 "     .      .      .      .     ",
00189 "     .      .      .      .     ",
00190 "     .      .      .      .     ",
00191 "                                ",
00192 "                                ",
00193 "                                "};
00194 */
00195
00196 #if HAVE_GTK
00197 Options options[1];
00199 Running running[1];
00201 Window window[1];
00203 #endif
00204
00215 void
00216 show_message (char *title, char *msg, int type)
00217 {
00218 #if HAVE_GTK
00219   GtkMessageDialog *dlg;
00220
00221   // Creating the dialog
00222   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00223     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00224
00225   // Setting the dialog title
00226   gtk_window_set_title (GTK_WINDOW (dlg), title);
00227
00228   // Showing the dialog and waiting response
00229   gtk_dialog_run (GTK_DIALOG (dlg));
00230
00231   // Closing and freeing memory
00232   gtk_widget_destroy (GTK_WIDGET (dlg));
00233
00234 #else
00235   printf ("%s: %s\n", title, msg);
00236 #endif
00237 }
00238
00245 void
00246 show_error (char *msg)
00247 {
00248   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00249 }
00250
00262 int
00263 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00264 {
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
01918   g_free (calibrate->error_old);
01919   g_free (calibrate->value_old);
01920   g_free (calibrate->value);
01921   g_free (calibrate->genetic_variable);
01922   g_free (calibrate->rangemax);
01923   g_free (calibrate->rangemin);
01924 #if DEBUG
01925   fprintf (stderr, "calibrate_free: end\n");
01926 #endif
01927 }
01928
01933 void
01934 calibrate_new ()
01935 {
01936   unsigned int i, j, *nbits;
01937
01938 #if DEBUG
01939   fprintf (stderr, "calibrate_new: start\n");
01940 #endif
01941
01942   // Obtaining and initing the pseudo-random numbers generator seed
01943   calibrate->seed = input->seed;
01944   gsl_rng_set (calibrate->rng, calibrate->seed);
01945
01946   // Replacing the working dir
01947   g_chdir (input->directory);
01948
01949   // Obtaining the simulator file
01950   calibrate->simulator = input->simulator;
01951
01952   // Obtaining the evaluator file
01953   calibrate->evaluator = input->evaluator;
01954
01955   // Reading the algorithm
01956   calibrate->algorithm = input->algorithm;
01957   switch (calibrate->algorithm)
01958     {
01959     case ALGORITHM_MONTE_CARLO:
01960       calibrate_step = calibrate_MonteCarlo;
01961       break;
01962     case ALGORITHM_SWEEP:
01963       calibrate_step = calibrate_sweep;
01964       break;
01965     default:
01966       calibrate_step = calibrate_genetic;
01967       calibrate->mutation_ratio = input->mutation_ratio;
01968       calibrate->reproduction_ratio = input->
      reproduction_ratio;
01969       calibrate->adaptation_ratio = input->adaptation_ratio;
01970     }
01971   calibrate->nsimulations = input->nsimulations;
01972   calibrate->niterations = input->niterations;
01973   calibrate->nbest = input->nbest;
01974   calibrate->tolerance = input->tolerance;
01975
01976   calibrate->simulation_best
01977     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
01978   calibrate->error_best
01979     = (double *) alloca (calibrate->nbest * sizeof (double));
01980
01981   // Reading the experimental data
01982 #if DEBUG
01983   fprintf (stderr, "calibrate_new: current directory=%s\n",
01984            g_get_current_dir ());
01985 #endif
01986   calibrate->nexperiments = input->nexperiments;
01987   calibrate->ninputs = input->ninputs;
01988   calibrate->experiment = input->experiment;
01989   calibrate->weight = input->weight;
01990   for (i = 0; i < input->ninputs; ++i)
01991     {
01992       calibrate->template[i] = input->template[i];
01993       calibrate->file[i]
01994         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
01995     }
01996   for (i = 0; i < input->nexperiments; ++i)
01997     {
01998 #if DEBUG
01999       fprintf (stderr, "calibrate_new: i=%u\n", i);
02000       fprintf (stderr, "calibrate_new: experiment=%s\n",
02001                calibrate->experiment[i]);
02002       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
02003 #endif
02004       for (j = 0; j < input->ninputs; ++j)
02005         {
02006 #if DEBUG
02007           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
```

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

```
02091           calibrate->nend);
02092 #endif
02093
02094   // Calculating simulations to perform on each thread
02095   calibrate->thread
02096     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02097   for (i = 0; i <= nthreads; ++i)
02098     {
02099       calibrate->thread[i] = calibrate->nstart
02100         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02101 #if DEBUG
02102       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02103                calibrate->thread[i]);
02104 #endif
02105     }
02106
02107   // Opening result files
02108   calibrate->file_result = g_fopen ("result", "w");
02109   calibrate->file_variables = g_fopen ("variables", "w");
02110
02111   // Performing the algorithm
02112   switch (calibrate->algorithm)
02113     {
02114       // Genetic algorithm
02115     case ALGORITHM_GENETIC:
02116       calibrate_genetic ();
02117       break;
02118
02119       // Iterative algorithm
02120     default:
02121       calibrate_iterate ();
02122     }
02123
02124   // Closing result files
02125   fclose (calibrate->file_variables);
02126   fclose (calibrate->file_result);
02127
02128 #if DEBUG
02129   fprintf (stderr, "calibrate_new: end\n");
02130 #endif
02131 }
02132
02133 #if HAVE_GTK
02134
02141 void
02142 input_save (char *filename)
02143 {
02144   unsigned int i, j;
02145   char *buffer;
02146   xmlDoc *doc;
02147   xmlNode *node, *child;
02148   GFile *file, *file2;
02149
02150   // Getting the input file directory
02151   input->name = g_path_get_basename (filename);
02152   input->directory = g_path_get_dirname (filename);
02153   file = g_file_new_for_path (input->directory);
02154
02155   // Opening the input file
02156   doc = xmlNewDoc ((const xmlChar *) "1.0");
02157
02158   // Setting root XML node
02159   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02160   xmlDocSetRootElement (doc, node);
02161
02162   // Adding properties to the root XML node
02163   file2 = g_file_new_for_path (input->simulator);
02164   buffer = g_file_get_relative_path (file, file2);
02165   g_object_unref (file2);
02166   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02167   g_free (buffer);
02168   if (input->evaluator)
02169     {
02170       file2 = g_file_new_for_path (input->evaluator);
02171       buffer = g_file_get_relative_path (file, file2);
02172       g_object_unref (file2);
02173       if (xmlStrlen ((xmlChar *) buffer))
02174         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02175       g_free (buffer);
02176     }
02177   if (input->seed != DEFAULT_RANDOM_SEED)
02178     xml_node_set_uint (node, XML_SEED, input->seed);
02179
02180   // Setting the algorithm
02181   buffer = (char *) g_malloc (64);
02182   switch (input->algorithm)
02183     {
```

```
02184     case ALGORITHM_MONTE_CARLO:
02185       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02186       snprintf (buffer, 64, "%u", input->nsimulations);
02187       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02188       snprintf (buffer, 64, "%u", input->niterations);
02189       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%.3lg", input->tolerance);
02191       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%u", input->nbest);
02193       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02194       break;
02195     case ALGORITHM_SWEEP:
02196       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02197       snprintf (buffer, 64, "%u", input->niterations);
02198       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02199       snprintf (buffer, 64, "%.3lg", input->tolerance);
02200       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%u", input->nbest);
02202       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02203       break;
02204     default:
02205       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02206       snprintf (buffer, 64, "%u", input->nsimulations);
02207       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02208       snprintf (buffer, 64, "%u", input->niterations);
02209       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02211       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02213       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02215       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02216       break;
02217     }
02218   g_free (buffer);
02219
02220   // Setting the experimental data
02221   for (i = 0; i < input->nexperiments; ++i)
02222     {
02223       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02224       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02225       if (input->weight[i] != 1.)
02226         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02227       for (j = 0; j < input->ninputs; ++j)
02228         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02229     }
02230
02231   // Setting the variables data
02232   for (i = 0; i < input->nvariables; ++i)
02233     {
02234       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02235       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02236       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02237       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02238         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02239       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02240       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02241         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02242       if (input->precision[i] != DEFAULT_PRECISION)
02243         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02244       if (input->algorithm == ALGORITHM_SWEEP)
02245         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02246       else if (input->algorithm == ALGORITHM_GENETIC)
02247         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02248     }
02249
02250   // Saving the XML file
02251   xmlSaveFormatFile (filename, doc, 1);
02252
02253   // Freeing memory
02254   xmlFreeDoc (doc);
02255 }
02256
02261 void
02262 options_new ()
02263 {
02264   options->label_processors
02265     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02266   options->spin_processors
```

```
02267     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02268   gtk_spin_button_set_value (options->spin_processors, (gdouble)
    nthreads);
02269   options->label_seed = (GtkLabel *)
02270     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02271   options->spin_seed = (GtkSpinButton *)
02272     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02273   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02274   options->grid = (GtkGrid *) gtk_grid_new ();
02275   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02276                    0, 0, 1, 1);
02277   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02278                    1, 0, 1, 1);
02279   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 1, 1, 1);
02280   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 1, 1, 1);
02281   gtk_widget_show_all (GTK_WIDGET (options->grid));
02282   options->dialog = (GtkDialog *)
02283     gtk_dialog_new_with_buttons (gettext ("Options"),
02284                                  window->window,
02285                                  GTK_DIALOG_MODAL,
02286                                  gettext ("_OK"), GTK_RESPONSE_OK,
02287                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02288                                  NULL);
02289   gtk_container_add
02290     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02291      GTK_WIDGET (options->grid));
02292   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02293     {
02294       nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02295       input->seed
02296         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02297     }
02298   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02299 }
02300
02305 void
02306 running_new ()
02307 {
02308 #if DEBUG
02309   fprintf (stderr, "running_new: start\n");
02310 #endif
02311   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02312   running->dialog = (GtkDialog *)
02313     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02314                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02315   gtk_container_add
02316     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02317      GTK_WIDGET (running->label));
02318   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02319 #if DEBUG
02320   fprintf (stderr, "running_new: end\n");
02321 #endif
02322 }
02323
02329 int
02330 window_save ()
02331 {
02332   char *buffer;
02333   GtkFileChooserDialog *dlg;
02334
02335 #if DEBUG
02336   fprintf (stderr, "window_save: start\n");
02337 #endif
02338
02339   // Opening the saving dialog
02340   dlg = (GtkFileChooserDialog *)
02341     gtk_file_chooser_dialog_new (gettext ("Save file"),
02342                                  window->window,
02343                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02344                                  gettext ("_Cancel"),
02345                                  GTK_RESPONSE_CANCEL,
02346                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02347   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02348   buffer = g_build_filename (input->directory, input->name, NULL);
02349   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02350   g_free (buffer);
02351
02352   // If OK response then saving
02353   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02354     {
02355
02356       // Adding properties to the root XML node
02357       input->simulator = gtk_file_chooser_get_filename
02358         (GTK_FILE_CHOOSER (window->button_simulator));
02359       if (gtk_toggle_button_get_active
02360           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02361         input->evaluator = gtk_file_chooser_get_filename
```

```
02362           (GTK_FILE_CHOOSER (window->button_evaluator));
02363       else
02364         input->evaluator = NULL;
02365
02366       // Setting the algorithm
02367       switch (window_get_algorithm ())
02368         {
02369         case ALGORITHM_MONTE_CARLO:
02370           input->algorithm = ALGORITHM_MONTE_CARLO;
02371           input->nsimulations
02372             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02373           input->niterations
02374             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02375           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02376           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02377           break;
02378         case ALGORITHM_SWEEP:
02379           input->algorithm = ALGORITHM_SWEEP;
02380           input->niterations
02381             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02382           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02383           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02384           break;
02385         default:
02386           input->algorithm = ALGORITHM_GENETIC;
02387           input->nsimulations
02388             = gtk_spin_button_get_value_as_int (window->spin_population);
02389           input->niterations
02390             = gtk_spin_button_get_value_as_int (window->spin_generations);
02391           input->mutation_ratio
02392             = gtk_spin_button_get_value (window->spin_mutation);
02393           input->reproduction_ratio
02394             = gtk_spin_button_get_value (window->spin_reproduction);
02395           input->adaptation_ratio
02396             = gtk_spin_button_get_value (window->spin_adaptation);
02397           break;
02398         }
02399
02400       // Saving the XML file
02401       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02402       input_save (buffer);
02403
02404       // Closing and freeing memory
02405       g_free (buffer);
02406       gtk_widget_destroy (GTK_WIDGET (dlg));
02407 #if DEBUG
02408       fprintf (stderr, "window_save: end\n");
02409 #endif
02410       return 1;
02411     }
02412
02413   // Closing and freeing memory
02414   gtk_widget_destroy (GTK_WIDGET (dlg));
02415 #if DEBUG
02416   fprintf (stderr, "window_save: end\n");
02417 #endif
02418   return 0;
02419 }
02420
02425 void
02426 window_run ()
02427 {
02428   unsigned int i;
02429   char *msg, *msg2, buffer[64], buffer2[64];
02430 #if DEBUG
02431   fprintf (stderr, "window_run: start\n");
02432 #endif
02433   if (!window_save ())
02434     {
02435 #if DEBUG
02436       fprintf (stderr, "window_run: end\n");
02437 #endif
02438       return;
02439     }
02440   running_new ();
02441   while (gtk_events_pending ())
02442     gtk_main_iteration ();
02443   calibrate_new ();
02444   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02445   snprintf (buffer, 64, "error=%.15le\n", calibrate->error_old[0]);
02446   msg2 = g_strdup (buffer);
02447   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02448     {
```

```
02449       snprintf (buffer, 64, "%s=%s\n",
02450                 calibrate->label[i], format[calibrate->precision[i]]);
02451       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02452       msg = g_strconcat (msg2, buffer2, NULL);
02453       g_free (msg2);
02454     }
02455   show_message (gettext ("Best result"), msg2, INFO_TYPE);
02456   g_free (msg2);
02457   calibrate_free ();
02458 #if DEBUG
02459   fprintf (stderr, "window_run: end\n");
02460 #endif
02461 }
02462
02467 void
02468 window_help ()
02469 {
02470   char *buffer, *buffer2;
02471   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
02472                               gettext ("user-manual.pdf"), NULL);
02473   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02474   g_free (buffer2);
02475   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02476   g_free (buffer);
02477 }
02478
02483 void
02484 window_about ()
02485 {
02486   gchar *authors[] = {
02487     "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02488     "Borja Latorre Garcés (borja.latorre@csic.es)",
02489     NULL
02490   };
02491   gtk_show_about_dialog (window->window,
02492                          "program_name",
02493                          "Calibrator",
02494                          "comments",
02495                          gettext ("A software to make calibrations of "
02496                                   "empirical parameters"),
02497                          "authors", authors,
02498                          "translator-credits",
02499                          "Javier Burguete Tolosa (jburguete@eead.csic.es)",
02500                          "version", "1.0.2",
02501                          "copyright",
02502                          "Copyright 2012-2015 Javier Burguete Tolosa",
02503                          "logo", window->logo,
02504                          "website-label", gettext ("Website"),
02505                          "website",
02506                          "https://github.com/jburguete/calibrator", NULL);
02507 }
02508
02514 int
02515 window_get_algorithm ()
02516 {
02517   unsigned int i;
02518   for (i = 0; i < NALGORITHMS; ++i)
02519     if (gtk_toggle_button_get_active
02520         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02521       break;
02522   return i;
02523 }
02524
02529 void
02530 window_update ()
02531 {
02532   unsigned int i;
02533   gtk_widget_set_sensitive
02534     (GTK_WIDGET (window->button_evaluator),
02535      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02536                                    (window->check_evaluator)));
02537   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02538   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02539   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02540   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02541   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02542   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
02543   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02544   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02545   gtk_widget_hide (GTK_WIDGET (window->label_population));
02546   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02547   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02548   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02549   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02550   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
02551   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02552   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
```

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

```
02637           gtk_toggle_button_get_active
02638           GTK_TOGGLE_BUTTON (window->check_template[i]));
02639         g_signal_handler_block
02640           (window->check_template[i], window->id_template[i]);
02641         g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02642         gtk_toggle_button_set_active
02643           (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02644         g_signal_handler_unblock
02645           (window->button_template[i], window->id_input[i]);
02646         g_signal_handler_unblock
02647           (window->check_template[i], window->id_template[i]);
02648       }
02649     while (++i < MAX_NINPUTS)
02650       {
02651         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02652         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02653       }
02654     gtk_widget_set_sensitive
02655       (GTK_WIDGET (window->spin_minabs),
02656        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02657     gtk_widget_set_sensitive
02658       (GTK_WIDGET (window->spin_maxabs),
02659        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02660 }
02661
02666 void
02667 window_set_algorithm ()
02668 {
02669   int i;
02670 #if DEBUG
02671   fprintf (stderr, "window_set_algorithm: start\n");
02672 #endif
02673   i = window_get_algorithm ();
02674   switch (i)
02675     {
02676     case ALGORITHM_SWEEP:
02677       input->nsweeps = (unsigned int *) g_realloc
02678         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02679       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02680       if (i < 0)
02681         i = 0;
02682       gtk_spin_button_set_value (window->spin_sweeps,
02683                                  (gdouble) input->nsweeps[i]);
02684       break;
02685     case ALGORITHM_GENETIC:
02686       input->nbits = (unsigned int *) g_realloc
02687         (input->nbits, input->nvariables * sizeof (unsigned int));
02688       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02689       if (i < 0)
02690         i = 0;
02691       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02692     }
02693   window_update ();
02694 #if DEBUG
02695   fprintf (stderr, "window_set_algorithm: end\n");
02696 #endif
02697 }
02698
02703 void
02704 window_set_experiment ()
02705 {
02706   unsigned int i, j;
02707   char *buffer1, *buffer2;
02708 #if DEBUG
02709   fprintf (stderr, "window_set_experiment: start\n");
02710 #endif
02711   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02712   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02713   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02714   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02715   g_free (buffer1);
02716   g_signal_handler_block
02717     (window->button_experiment, window->id_experiment_name);
02718   gtk_file_chooser_set_filename
02719     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
02720   g_signal_handler_unblock
02721     (window->button_experiment, window->id_experiment_name);
02722   g_free (buffer2);
02723   for (j = 0; j < input->ninputs; ++j)
02724     {
02725       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02726       buffer2
02727         = g_build_filename (input->directory, input->template[j][i], NULL);
02728       gtk_file_chooser_set_filename
```

```
02729         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02730       g_free (buffer2);
02731       g_signal_handler_unblock
02732         (window->button_template[j], window->id_input[j]);
02733     }
02734 #if DEBUG
02735   fprintf (stderr, "window_set_experiment: end\n");
02736 #endif
02737 }
02738
02743 void
02744 window_remove_experiment ()
02745 {
02746   unsigned int i, j;
02747   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02748   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02749   gtk_combo_box_text_remove (window->combo_experiment, i);
02750   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02751   xmlFree (input->experiment[i]);
02752   --input->nexperiments;
02753   for (j = i; j < input->nexperiments; ++j)
02754     {
02755       input->experiment[j] = input->experiment[j + 1];
02756       input->weight[j] = input->weight[j + 1];
02757     }
02758   j = input->nexperiments - 1;
02759   if (i > j)
02760     i = j;
02761   for (j = 0; j < input->ninputs; ++j)
02762     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02763   g_signal_handler_block
02764     (window->button_experiment, window->id_experiment_name);
02765   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02766   g_signal_handler_unblock
02767     (window->button_experiment, window->id_experiment_name);
02768   for (j = 0; j < input->ninputs; ++j)
02769     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02770   window_update ();
02771 }
02772
02777 void
02778 window_add_experiment ()
02779 {
02780   unsigned int i, j;
02781   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02782   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02783   gtk_combo_box_text_insert_text
02784     (window->combo_experiment, i, input->experiment[i]);
02785   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02786   input->experiment = (char **) g_realloc
02787     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02788   input->weight = (double *) g_realloc
02789     (input->weight, (input->nexperiments + 1) * sizeof (double));
02790   for (j = input->nexperiments - 1; j > i; --j)
02791     {
02792       input->experiment[j + 1] = input->experiment[j];
02793       input->weight[j + 1] = input->weight[j];
02794     }
02795   input->experiment[j + 1]
02796     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02797   input->weight[j + 1] = input->weight[j];
02798   ++input->nexperiments;
02799   for (j = 0; j < input->ninputs; ++j)
02800     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02801   g_signal_handler_block
02802     (window->button_experiment, window->id_experiment_name);
02803   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02804   g_signal_handler_unblock
02805     (window->button_experiment, window->id_experiment_name);
02806   for (j = 0; j < input->ninputs; ++j)
02807     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02808   window_update ();
02809 }
02810
02815 void
02816 window_name_experiment ()
02817 {
02818   unsigned int i;
02819   char *buffer;
```

```
02820   GFile *file1, *file2;
02821 #if DEBUG
02822   fprintf (stderr, "window_name_experiment: start\n");
02823 #endif
02824   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02825   file1
02826     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02827   file2 = g_file_new_for_path (input->directory);
02828   buffer = g_file_get_relative_path (file2, file1);
02829   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02830   gtk_combo_box_text_remove (window->combo_experiment, i);
02831   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02832   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02833   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02834   g_free (buffer);
02835   g_object_unref (file2);
02836   g_object_unref (file1);
02837 #if DEBUG
02838   fprintf (stderr, "window_name_experiment: end\n");
02839 #endif
02840 }
02841
02846 void
02847 window_weight_experiment ()
02848 {
02849   unsigned int i;
02850 #if DEBUG
02851   fprintf (stderr, "window_weight_experiment: start\n");
02852 #endif
02853   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02854   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02855 #if DEBUG
02856   fprintf (stderr, "window_weight_experiment: end\n");
02857 #endif
02858 }
02859
02865 void
02866 window_inputs_experiment ()
02867 {
02868   unsigned int j;
02869 #if DEBUG
02870   fprintf (stderr, "window_inputs_experiment: start\n");
02871 #endif
02872   j = input->ninputs - 1;
02873   if (j
02874       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02875                                         (window->check_template[j])))
02876     --input->ninputs;
02877   if (input->ninputs < MAX_NINPUTS
02878       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02879                                        (window->check_template[j])))
02880     {
02881       ++input->ninputs;
02882       for (j = 0; j < input->ninputs; ++j)
02883         {
02884           input->template[j] = (char **)
02885             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02886         }
02887     }
02888   window_update ();
02889 #if DEBUG
02890   fprintf (stderr, "window_inputs_experiment: end\n");
02891 #endif
02892 }
02893
02901 void
02902 window_template_experiment (void *data)
02903 {
02904   unsigned int i, j;
02905   char *buffer;
02906   GFile *file1, *file2;
02907 #if DEBUG
02908   fprintf (stderr, "window_template_experiment: start\n");
02909 #endif
02910   i = (size_t) data;
02911   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02912   file1
02913     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02914   file2 = g_file_new_for_path (input->directory);
02915   buffer = g_file_get_relative_path (file2, file1);
02916   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02917   g_free (buffer);
02918   g_object_unref (file2);
02919   g_object_unref (file1);
02920 #if DEBUG
```

```
02921   fprintf (stderr, "window_template_experiment: end\n");
02922 #endif
02923 }
02924
02929 void
02930 window_set_variable ()
02931 {
02932   unsigned int i;
02933 #if DEBUG
02934   fprintf (stderr, "window_set_variable: start\n");
02935 #endif
02936   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02937   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
02938   gtk_entry_set_text (window->entry_variable, input->label[i]);
02939   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
02940   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02941   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
02942   if (input->rangeminabs[i] != -G_MAXDOUBLE)
02943     {
02944       gtk_spin_button_set_value (window->spin_minabs, input->
      rangeminabs[i]);
02945       gtk_toggle_button_set_active
02946         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
02947     }
02948   else
02949     {
02950       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
02951       gtk_toggle_button_set_active
02952         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
02953     }
02954   if (input->rangemaxabs[i] != G_MAXDOUBLE)
02955     {
02956       gtk_spin_button_set_value (window->spin_maxabs, input->
      rangemaxabs[i]);
02957       gtk_toggle_button_set_active
02958         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
02959     }
02960   else
02961     {
02962       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
02963       gtk_toggle_button_set_active
02964         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
02965     }
02966   gtk_spin_button_set_value (window->spin_precision, input->
      precision[i]);
02967 #if DEBUG
02968   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
02969            input->precision[i]);
02970 #endif
02971   switch (window_get_algorithm ())
02972     {
02973     case ALGORITHM_SWEEP:
02974       gtk_spin_button_set_value (window->spin_sweeps,
02975                                  (gdouble) input->nsweeps[i]);
02976 #if DEBUG
02977       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
02978                input->nsweeps[i]);
02979 #endif
02980       break;
02981     case ALGORITHM_GENETIC:
02982       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02983 #if DEBUG
02984       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
02985                input->nbits[i]);
02986 #endif
02987       break;
02988     }
02989   window_update ();
02990 #if DEBUG
02991   fprintf (stderr, "window_set_variable: end\n");
02992 #endif
02993 }
02994
02999 void
03000 window_remove_variable ()
03001 {
03002   unsigned int i, j;
03003   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03004   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03005   gtk_combo_box_text_remove (window->combo_variable, i);
03006   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03007   xmlFree (input->label[i]);
```

```
03008   --input->nvariables;
03009   for (j = i; j < input->nvariables; ++j)
03010     {
03011       input->label[j] = input->label[j + 1];
03012       input->rangemin[j] = input->rangemin[j + 1];
03013       input->rangemax[j] = input->rangemax[j + 1];
03014       input->rangeminabs[j] = input->rangeminabs[j + 1];
03015       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03016       input->precision[j] = input->precision[j + 1];
03017       switch (window_get_algorithm ())
03018         {
03019         case ALGORITHM_SWEEP:
03020           input->nsweeps[j] = input->nsweeps[j + 1];
03021           break;
03022         case ALGORITHM_GENETIC:
03023           input->nbits[j] = input->nbits[j + 1];
03024         }
03025     }
03026   j = input->nvariables - 1;
03027   if (i > j)
03028     i = j;
03029   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03030   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03031   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03032   window_update ();
03033 }
03034
03039 void
03040 window_add_variable ()
03041 {
03042   unsigned int i, j;
03043 #if DEBUG
03044   fprintf (stderr, "window_add_variable: start\n");
03045 #endif
03046   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03047   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03048   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03049   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03050   input->label = (char **) g_realloc
03051     (input->label, (input->nvariables + 1) * sizeof (char *));
03052   input->rangemin = (double *) g_realloc
03053     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03054   input->rangemax = (double *) g_realloc
03055     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03056   input->rangeminabs = (double *) g_realloc
03057     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03058   input->rangemaxabs = (double *) g_realloc
03059     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03060   input->precision = (unsigned int *) g_realloc
03061     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03062   for (j = input->nvariables - 1; j > i; --j)
03063     {
03064       input->label[j + 1] = input->label[j];
03065       input->rangemin[j + 1] = input->rangemin[j];
03066       input->rangemax[j + 1] = input->rangemax[j];
03067       input->rangeminabs[j + 1] = input->rangeminabs[j];
03068       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03069       input->precision[j + 1] = input->precision[j];
03070     }
03071   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03072   input->rangemin[j + 1] = input->rangemin[j];
03073   input->rangemax[j + 1] = input->rangemax[j];
03074   input->rangeminabs[j + 1] = input->rangeminabs[j];
03075   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03076   input->precision[j + 1] = input->precision[j];
03077   switch (window_get_algorithm ())
03078     {
03079     case ALGORITHM_SWEEP:
03080       input->nsweeps = (unsigned int *) g_realloc
03081         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03082       for (j = input->nvariables - 1; j > i; --j)
03083         input->nsweeps[j + 1] = input->nsweeps[j];
03084       input->nsweeps[j + 1] = input->nsweeps[j];
03085       break;
03086     case ALGORITHM_GENETIC:
03087       input->nbits = (unsigned int *) g_realloc
03088         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03089       for (j = input->nvariables - 1; j > i; --j)
03090         input->nbits[j + 1] = input->nbits[j];
03091       input->nbits[j + 1] = input->nbits[j];
03092     }
03093   ++input->nvariables;
```

```
03094   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03095   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03096   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03097   window_update ();
03098 #if DEBUG
03099   fprintf (stderr, "window_add_variable: end\n");
03100 #endif
03101 }
03102
03107 void
03108 window_label_variable ()
03109 {
03110   unsigned int i;
03111   const char *buffer;
03112 #if DEBUG
03113   fprintf (stderr, "window_label_variable: start\n");
03114 #endif
03115   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03116   buffer = gtk_entry_get_text (window->entry_variable);
03117   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03118   gtk_combo_box_text_remove (window->combo_variable, i);
03119   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03120   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03121   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03122 #if DEBUG
03123   fprintf (stderr, "window_label_variable: end\n");
03124 #endif
03125 }
03126
03131 void
03132 window_precision_variable ()
03133 {
03134   unsigned int i;
03135 #if DEBUG
03136   fprintf (stderr, "window_precision_variable: start\n");
03137 #endif
03138   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03139   input->precision[i]
03140     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03141   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03142   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03143   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03144   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03145 #if DEBUG
03146   fprintf (stderr, "window_precision_variable: end\n");
03147 #endif
03148 }
03149
03154 void
03155 window_rangemin_variable ()
03156 {
03157   unsigned int i;
03158 #if DEBUG
03159   fprintf (stderr, "window_rangemin_variable: start\n");
03160 #endif
03161   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03162   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03163 #if DEBUG
03164   fprintf (stderr, "window_rangemin_variable: end\n");
03165 #endif
03166 }
03167
03172 void
03173 window_rangemax_variable ()
03174 {
03175   unsigned int i;
03176 #if DEBUG
03177   fprintf (stderr, "window_rangemax_variable: start\n");
03178 #endif
03179   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03180   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03181 #if DEBUG
03182   fprintf (stderr, "window_rangemax_variable: end\n");
03183 #endif
03184 }
03185
03190 void
03191 window_rangeminabs_variable ()
03192 {
03193   unsigned int i;
03194 #if DEBUG
03195   fprintf (stderr, "window_rangeminabs_variable: start\n");
03196 #endif
```

```
03197   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03198   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03199 #if DEBUG
03200   fprintf (stderr, "window_rangeminabs_variable: end\n");
03201 #endif
03202 }
03203
03208 void
03209 window_rangemaxabs_variable ()
03210 {
03211   unsigned int i;
03212 #if DEBUG
03213   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03214 #endif
03215   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03216   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03217 #if DEBUG
03218   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03219 #endif
03220 }
03221
03226 void
03227 window_update_variable ()
03228 {
03229   int i;
03230 #if DEBUG
03231   fprintf (stderr, "window_update_variable: start\n");
03232 #endif
03233   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03234   if (i < 0)
03235     i = 0;
03236   switch (window_get_algorithm ())
03237     {
03238     case ALGORITHM_SWEEP:
03239       input->nsweeps[i]
03240         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03241 #if DEBUG
03242       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03243                input->nsweeps[i]);
03244 #endif
03245       break;
03246     case ALGORITHM_GENETIC:
03247       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03248 #if DEBUG
03249       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
03250                input->nbits[i]);
03251 #endif
03252     }
03253 #if DEBUG
03254   fprintf (stderr, "window_update_variable: end\n");
03255 #endif
03256 }
03257
03265 int
03266 window_read (char *filename)
03267 {
03268   unsigned int i;
03269   char *buffer;
03270 #if DEBUG
03271   fprintf (stderr, "window_read: start\n");
03272 #endif
03273
03274   // Reading new input file
03275   input_free ();
03276   if (!input_open (filename))
03277     return 0;
03278
03279   // Setting GTK+ widgets data
03280   buffer = g_build_filename (input->directory, input->simulator, NULL);
03281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03282                                  (window->button_simulator), buffer);
03283   g_free (buffer);
03284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03285                                 (size_t) input->evaluator);
03286   if (input->evaluator)
03287     {
03288       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03290                                      (window->button_evaluator), buffer);
03291       g_free (buffer);
03292     }
03293   gtk_toggle_button_set_active
03294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03295   switch (input->algorithm)
```

```
03296     {
03297     case ALGORITHM_MONTE_CARLO:
03298       gtk_spin_button_set_value (window->spin_simulations,
03299                                  (gdouble) input->nsimulations);
03300     case ALGORITHM_SWEEP:
03301       gtk_spin_button_set_value (window->spin_iterations,
03302                                  (gdouble) input->niterations);
03303       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03304       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03305       break;
03306     default:
03307       gtk_spin_button_set_value (window->spin_population,
03308                                  (gdouble) input->nsimulations);
03309       gtk_spin_button_set_value (window->spin_generations,
03310                                  (gdouble) input->niterations);
03311       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03312       gtk_spin_button_set_value (window->spin_reproduction,
03313                                  input->reproduction_ratio);
03314       gtk_spin_button_set_value (window->spin_adaptation,
03315                                  input->adaptation_ratio);
03316     }
03317   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03318   g_signal_handler_block (window->button_experiment,
03319                           window->id_experiment_name);
03320   gtk_combo_box_text_remove_all (window->combo_experiment);
03321   for (i = 0; i < input->nexperiments; ++i)
03322     gtk_combo_box_text_append_text (window->combo_experiment,
03323                                     input->experiment[i]);
03324   g_signal_handler_unblock
03325     (window->button_experiment, window->id_experiment_name);
03326   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03327   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03328   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03329   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03330   gtk_combo_box_text_remove_all (window->combo_variable);
03331   for (i = 0; i < input->nvariables; ++i)
03332     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03333   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03334   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03335   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03336   window_set_variable ();
03337   window_update ();
03338
03339 #if DEBUG
03340   fprintf (stderr, "window_read: end\n");
03341 #endif
03342   return 1;
03343 }
03344
03349 void
03350 window_open ()
03351 {
03352   char *buffer, *directory, *name;
03353   GtkFileChooserDialog *dlg;
03354
03355 #if DEBUG
03356   fprintf (stderr, "window_open: start\n");
03357 #endif
03358
03359   // Saving a backup of the current input file
03360   directory = g_strdup (input->directory);
03361   name = g_strdup (input->name);
03362
03363   // Opening dialog
03364   dlg = (GtkFileChooserDialog *)
03365     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03366                                  window->window,
03367                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03368                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03369                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03370   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03371     {
03372
03373       // Traying to open the input file
03374       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03375       if (!window_read (buffer))
03376         {
```

```
03377 #if DEBUG
03378           fprintf (stderr, "window_open: error reading input file\n");
03379 #endif
03380
03381           // Reading backup file on error
03382           buffer = g_build_filename (directory, name, NULL);
03383           if (!input_open (buffer))
03384             {
03385
03386               // Closing on backup file reading error
03387 #if DEBUG
03388               fprintf (stderr, "window_read: error reading backup file\n");
03389 #endif
03390               g_free (buffer);
03391               g_free (name);
03392               g_free (directory);
03393 #if DEBUG
03394               fprintf (stderr, "window_open: end\n");
03395 #endif
03396               gtk_main_quit ();
03397             }
03398           g_free (buffer);
03399         }
03400       else
03401         break;
03402     }
03403
03404   // Freeing and closing
03405   g_free (name);
03406   g_free (directory);
03407   gtk_widget_destroy (GTK_WIDGET (dlg));
03408 #if DEBUG
03409   fprintf (stderr, "window_open: end\n");
03410 #endif
03411 }
03412
03417 void
03418 window_new ()
03419 {
03420   unsigned int i;
03421   char *buffer, *buffer2, buffer3[64];
03422   GtkViewport *viewport;
03423   char *label_algorithm[NALGORITHMS] = {
03424     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03425   };
03426
03427   // Creating the window
03428   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
03429
03430   // Finish when closing the window
03431   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
03432
03433   // Setting the window title
03434   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03435
03436   // Creating the open button
03437   window->button_open = (GtkToolButton *) gtk_tool_button_new
03438     (gtk_image_new_from_icon_name ("document-open",
03439                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03440      gettext ("Open"));
03441   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03442
03443   // Creating the save button
03444   window->button_save = (GtkToolButton *) gtk_tool_button_new
03445     (gtk_image_new_from_icon_name ("document-save",
03446                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03447      gettext ("Save"));
03448   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
03449                     NULL);
03450
03451   // Creating the run button
03452   window->button_run = (GtkToolButton *) gtk_tool_button_new
03453     (gtk_image_new_from_icon_name ("system-run",
03454                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03455      gettext ("Run"));
03456   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03457
03458   // Creating the options button
03459   window->button_options = (GtkToolButton *) gtk_tool_button_new
03460     (gtk_image_new_from_icon_name ("preferences-system",
03461                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03462      gettext ("Options"));
03463   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03464
03465   // Creating the help button
03466   window->button_help = (GtkToolButton *) gtk_tool_button_new
```

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

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

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

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

```
03809   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03810   gtk_grid_attach (window->grid_experiment,
03811                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03812   gtk_grid_attach (window->grid_experiment,
03813                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03814   gtk_grid_attach (window->grid_experiment,
03815                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03816   gtk_grid_attach (window->grid_experiment,
03817                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03818   gtk_grid_attach (window->grid_experiment,
03819                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03820   gtk_grid_attach (window->grid_experiment,
03821                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03822   gtk_grid_attach (window->grid_experiment,
03823                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03824   for (i = 0; i < MAX_NINPUTS; ++i)
03825     {
03826       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03827       window->check_template[i] = (GtkCheckButton *)
03828         gtk_check_button_new_with_label (buffer3);
03829       window->id_template[i]
03830         = g_signal_connect (window->check_template[i], "toggled",
03831                             window_inputs_experiment, NULL);
03832       gtk_grid_attach (window->grid_experiment,
03833                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03834       window->button_template[i] = (GtkFileChooserButton *)
03835         gtk_file_chooser_button_new (gettext ("Input template"),
03836                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03837       gtk_widget_set_tooltip_text
03838         (GTK_WIDGET (window->button_template[i]),
03839          gettext ("Experimental input template file"));
03840       window->id_input[i]
03841         = g_signal_connect_swapped (window->button_template[i],
03842                                     "selection-changed",
03843                                     (void (*)) window_template_experiment,
03844                                     (void *) (size_t) i);
03845       gtk_grid_attach (window->grid_experiment,
03846                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03847     }
03848   window->frame_experiment
03849     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03850   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
03851                      GTK_WIDGET (window->grid_experiment));
03852
03853   // Creating the grid and attaching the widgets to the grid
03854   window->grid = (GtkGrid *) gtk_grid_new ();
03855   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 6, 1);
03856   gtk_grid_attach (window->grid,
03857                    GTK_WIDGET (window->label_simulator), 0, 1, 1, 1);
03858   gtk_grid_attach (window->grid,
03859                    GTK_WIDGET (window->button_simulator), 1, 1, 1, 1);
03860   gtk_grid_attach (window->grid,
03861                    GTK_WIDGET (window->check_evaluator), 2, 1, 1, 1);
03862   gtk_grid_attach (window->grid,
03863                    GTK_WIDGET (window->button_evaluator), 3, 1, 1, 1);
03864   gtk_grid_attach (window->grid,
03865                    GTK_WIDGET (window->frame_algorithm), 0, 2, 2, 1);
03866   gtk_grid_attach (window->grid,
03867                    GTK_WIDGET (window->frame_variable), 2, 2, 2, 1);
03868   gtk_grid_attach (window->grid,
03869                    GTK_WIDGET (window->frame_experiment), 4, 2, 2, 1);
03870   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
03871
03872   // Setting the window logo
03873   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
03874   gtk_window_set_icon (window->window, window->logo);
03875
03876   // Showing the window
03877   gtk_widget_show_all (GTK_WIDGET (window->window));
03878
03879   // In Windows the default scrolled size is wrong
03880 #ifdef G_OS_WIN32
03881   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
03882   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
03883   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
03884   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
03885 #endif
03886
03887   // Reading initial example
03888   input_new ();
03889   buffer2 = g_get_current_dir ();
03890   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
03891   g_free (buffer2);
03892   window_read (buffer);
03893   g_free (buffer);
03894 }
```

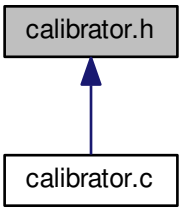
```
03895
03896 #endif
03897
03903 int
03904 cores_number ()
03905 {
03906 #ifdef G_OS_WIN32
03907   SYSTEM_INFO sysinfo;
03908   GetSystemInfo (&sysinfo);
03909   return sysinfo.dwNumberOfProcessors;
03910 #else
03911   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03912 #endif
03913 }
03914
03924 int
03925 main (int argn, char **argc)
03926 {
03927   // Starting pseudo-random numbers generator
03928   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
03929   calibrate->seed = DEFAULT_RANDOM_SEED;
03930
03931   // Allowing spaces in the XML data file
03932   xmlKeepBlanksDefault (0);
03933
03934   // Starting MPI
03935 #if HAVE_MPI
03936   MPI_Init (&argn, &argc);
03937   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
03938   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
03939   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
03940 #else
03941   ntasks = 1;
03942 #endif
03943
03944 #if HAVE_GTK
03945
03946   // Getting threads number
03947   nthreads = cores_number ();
03948
03949   // Setting local language and international floating point numbers notation
03950   setlocale (LC_ALL, "");
03951   setlocale (LC_NUMERIC, "C");
03952   window->application_directory = g_get_current_dir ();
03953   bindtextdomain (PROGRAM_INTERFACE,
03954                   g_build_filename (window->application_directory,
03955                                     LOCALE_DIR, NULL));
03956   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
03957   textdomain (PROGRAM_INTERFACE);
03958
03959   // Initing GTK+
03960   gtk_disable_setlocale ();
03961   gtk_init (&argn, &argc);
03962
03963   // Opening the main window
03964   window_new ();
03965   gtk_main ();
03966
03967   // Freeing memory
03968   gtk_widget_destroy (GTK_WIDGET (window->window));
03969   g_free (window->application_directory);
03970
03971 #else
03972
03973   // Checking syntax
03974   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
03975     {
03976       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
03977       return 1;
03978     }
03979
03980   // Getting threads number
03981   if (argn == 2)
03982     nthreads = cores_number ();
03983   else
03984     nthreads = atoi (argc[2]);
03985   printf ("nthreads=%u\n", nthreads);
03986
03987   // Making calibration
03988   input_new ();
03989   if (input_open (argc[argn - 1]))
03990     calibrate_new ();
03991
03992   // Freeing memory
03993   calibrate_free ();
03994
03995 #endif
```

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

5.3 calibrator.h File Reference

Header file of the calibrator.

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



Data Structures

- struct Input

  Struct to define the calibration input file.
- struct Calibrate

  Struct to define the calibration data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

  Enum to define the algorithms.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to refine the search ranges of the variables in iterative algorithms.

- void calibrate_iterate ()

  Function to iterate the algorithm.
- void calibrate_new ()

  Function to open and perform a calibration.

5.3.1 Detailed Description

Header file of the calibrator.

Authors

Javier Burguete.

Copyright

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 43 of file calibrator.h.

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

5.3.3 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1330 of file calibrator.c.

```
01331 {
01332   unsigned int i, j;
01333   double e;
01334 #if DEBUG
01335   fprintf (stderr, "calibrate_best_sequential: start\n");
01336 #endif
01337   if (calibrate->nsaveds < calibrate->nbest
```

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

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

Function to save the best simulations of a thread.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1285 of file calibrator.c.

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

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

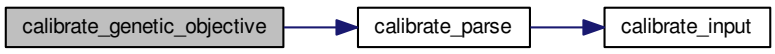
Returns

objective function value.

Definition at line 1639 of file calibrator.c.

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

Here is the call graph for this function:



5.3.3.4 void calibrate_input (unsigned int *simulation*, char * *input*, GMappedFile * *template*)

Function to write the simulation input file.

Parameters

simulation	Simulation number.
input	Input file name.
template	Template of the input file name.

Definition at line 1034 of file calibrator.c.

```
01035 {
01036   unsigned int i;
01037   char buffer[32], value[32], *buffer2, *buffer3, *content;
01038   FILE *file;
01039   gsize length;
01040   GRegex *regex;
01041
```

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

5.3.3.5 void calibrate_merge (unsigned int *nsaveds*, unsigned int * *simulation_best*, double * *error_best*)

Function to merge the 2 calibration results.

Parameters

nsaveds	Number of saved results.

simulation_best	Array of best simulation numbers.
error_best	Array of best objective function values.

Definition at line 1446 of file calibrator.c.

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

5.3.3.6 double calibrate_parse (unsigned int *simulation*, unsigned int *experiment*)

Function to parse input files, simulating and calculating the \ objective function.

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 1121 of file calibrator.c.

```
01122 {
01123   unsigned int i;
01124   double e;
01125   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
```

```
01126     *buffer3, *buffer4;
01127   FILE *file_result;
01128
01129 #if DEBUG
01130   fprintf (stderr, "calibrate_parse: start\n");
01131   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01132            experiment);
01133 #endif
01134
01135   // Opening input files
01136   for (i = 0; i < calibrate->ninputs; ++i)
01137     {
01138       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01139 #if DEBUG
01140       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01141 #endif
01142       calibrate_input (simulation, &input[i][0],
01143                        calibrate->file[i][experiment]);
01144     }
01145   for (; i < MAX_NINPUTS; ++i)
01146     strcpy (&input[i][0], "");
01147 #if DEBUG
01148   fprintf (stderr, "calibrate_parse: parsing end\n");
01149 #endif
01150
01151   // Performing the simulation
01152   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01153   buffer2 = g_path_get_dirname (calibrate->simulator);
01154   buffer3 = g_path_get_basename (calibrate->simulator);
01155   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01156   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01157             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01158             input[6], input[7], output);
01159   g_free (buffer4);
01160   g_free (buffer3);
01161   g_free (buffer2);
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01164 #endif
01165   system (buffer);
01166
01167   // Checking the objective value function
01168   if (calibrate->evaluator)
01169     {
01170       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01171       buffer2 = g_path_get_dirname (calibrate->evaluator);
01172       buffer3 = g_path_get_basename (calibrate->evaluator);
01173       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01174       snprintf (buffer, 512, "\"%s\" %s %s %s",
01175                 buffer4, output, calibrate->experiment[experiment], result);
01176       g_free (buffer4);
01177       g_free (buffer3);
01178       g_free (buffer2);
01179 #if DEBUG
01180       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01181 #endif
01182       system (buffer);
01183       file_result = g_fopen (result, "r");
01184       e = atof (fgets (buffer, 512, file_result));
01185       fclose (file_result);
01186     }
01187   else
01188     {
01189       strcpy (result, "");
01190       file_result = g_fopen (output, "r");
01191       e = atof (fgets (buffer, 512, file_result));
01192       fclose (file_result);
01193     }
01194
01195   // Removing files
01196 #if !DEBUG
01197   for (i = 0; i < calibrate->ninputs; ++i)
01198     {
01199       if (calibrate->file[i][0])
01200         {
01201           snprintf (buffer, 512, RM " %s", &input[i][0]);
01202           system (buffer);
01203         }
01204     }
01205   snprintf (buffer, 512, RM " %s %s", output, result);
01206   system (buffer);
01207 #endif
01208
01209 #if DEBUG
01210   fprintf (stderr, "calibrate_parse: end\n");
01211 #endif
01212
```

```
01213   // Returning the objective function
01214   return e * calibrate->weight[experiment];
01215 }
```

Here is the call graph for this function:

5.3.3.7 void calibrate_save_variables (unsigned int *simulation*, double *error*)

Function to save in a file the variables and the error.

Parameters

simulation	Simulation number.
error	Error value.

Definition at line 1257 of file calibrator.c.

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

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

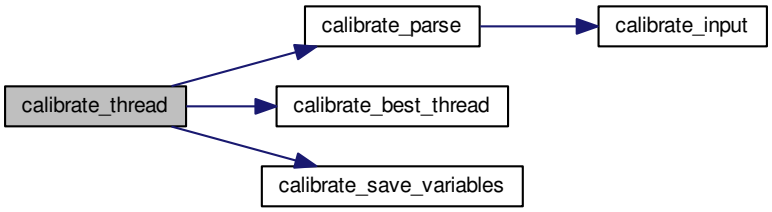
NULL

Definition at line 1372 of file calibrator.c.

```
01373 {
01374   unsigned int i, j, thread;
01375   double e;
01376 #if DEBUG
```

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

Here is the call graph for this function:



5.3.3.9 int input_open (char * *filename*)

Function to open the input file.

Parameters

filename	Input data file name.

Returns

1 on success, 0 on error.

Definition at line 472 of file calibrator.c.

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

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

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

```
00658
00659       // Obtaining iterations number
00660       input->niterations
00661         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00662       if (error_code == 1)
00663         input->niterations = 1;
00664       else if (error_code)
00665         {
00666           msg = gettext ("Bad iterations number");
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining best number
00671       if (xmlHasProp (node, XML_NBEST))
00672         {
00673           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00674           if (error_code || !input->nbest)
00675             {
00676               msg = gettext ("Invalid best number");
00677               goto exit_on_error;
00678             }
00679         }
00680       else
00681         input->nbest = 1;
00682
00683       // Obtaining tolerance
00684       if (xmlHasProp (node, XML_TOLERANCE))
00685         {
00686           input->tolerance
00687             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00688           if (error_code || input->tolerance < 0.)
00689             {
00690               msg = gettext ("Invalid tolerance");
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         input->tolerance = 0.;
00696     }
00697
00698   // Reading the experimental data
00699   for (child = node->children; child; child = child->next)
00700     {
00701       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00702         break;
00703 #if DEBUG
00704       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00705 #endif
00706       if (xmlHasProp (child, XML_NAME))
00707         {
00708           input->experiment
00709             = g_realloc (input->experiment,
00710                          (1 + input->nexperiments) * sizeof (char *));
00711           input->experiment[input->nexperiments]
00712             = (char *) xmlGetProp (child, XML_NAME);
00713         }
00714       else
00715         {
00716           snprintf (buffer2, 64, "%s %u: %s",
00717                     gettext ("Experiment"),
00718                     input->nexperiments + 1, gettext ("no data file name"));
00719           msg = buffer2;
00720           goto exit_on_error;
00721         }
00722 #if DEBUG
00723       fprintf (stderr, "input_open: experiment=%s\n",
00724                input->experiment[input->nexperiments]);
00725 #endif
00726       input->weight = g_realloc (input->weight,
00727                                  (1 + input->nexperiments) * sizeof (double));
00728       if (xmlHasProp (child, XML_WEIGHT))
00729         {
00730           input->weight[input->nexperiments]
00731             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00732           if (error_code)
00733             {
00734               snprintf (buffer2, 64, "%s %u: %s",
00735                         gettext ("Experiment"),
00736                         input->nexperiments + 1, gettext ("bad weight"));
00737               msg = buffer2;
00738               goto exit_on_error;
00739             }
00740         }
00741       else
00742         input->weight[input->nexperiments] = 1.;
00743 #if DEBUG
```

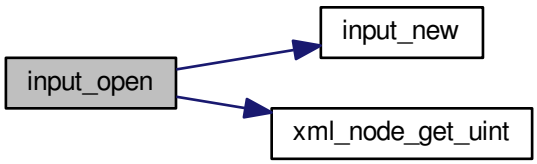
```
00744       fprintf (stderr, "input_open: weight=%lg\n",
00745                input->weight[input->nexperiments]);
00746 #endif
00747       if (!input->nexperiments)
00748         input->ninputs = 0;
00749 #if DEBUG
00750       fprintf (stderr, "input_open: template[0]\n");
00751 #endif
00752   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

```
00831     }
00832
00833   // Reading the variables data
00834   for (; child; child = child->next)
00835     {
00836       if (xmlStrcmp (child->name, XML_VARIABLE))
00837         {
00838           snprintf (buffer2, 64, "%s %u: %s",
00839                     gettext ("Variable"),
00840                     input->nvariables + 1, gettext ("bad XML node"));
00841           msg = buffer2;
00842           goto exit_on_error;
00843         }
00844       if (xmlHasProp (child, XML_NAME))
00845         {
00846           input->label = g_realloc
00847             (input->label, (1 + input->nvariables) * sizeof (char *));
00848           input->label[input->nvariables]
00849             = (char *) xmlGetProp (child, XML_NAME);
00850         }
00851       else
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("no name"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_MINIMUM))
00860         {
00861           input->rangemin = g_realloc
00862             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00863           input->rangeminabs = g_realloc
00864             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00865           input->rangemin[input->nvariables]
00866             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00867           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00868             {
00869               input->rangeminabs[input->nvariables]
00870                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00871             }
00872           else
00873             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00874           if (input->rangemin[input->nvariables]
00875               < input->rangeminabs[input->nvariables])
00876             {
00877               snprintf (buffer2, 64, "%s %u: %s",
00878                         gettext ("Variable"),
00879                         input->nvariables + 1,
00880                         gettext ("minimum range not allowed"));
00881               msg = buffer2;
00882               goto exit_on_error;
00883             }
00884         }
00885       else
00886         {
00887           snprintf (buffer2, 64, "%s %u: %s",
00888                     gettext ("Variable"),
00889                     input->nvariables + 1, gettext ("no minimum range"));
00890           msg = buffer2;
00891           goto exit_on_error;
00892         }
00893       if (xmlHasProp (child, XML_MAXIMUM))
00894         {
00895           input->rangemax = g_realloc
00896             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00897           input->rangemaxabs = g_realloc
00898             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00899           input->rangemax[input->nvariables]
00900             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00901           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00902             input->rangemaxabs[input->nvariables]
00903               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00904           else
00905             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00906           if (input->rangemax[input->nvariables]
00907               > input->rangemaxabs[input->nvariables])
00908             {
00909               snprintf (buffer2, 64, "%s %u: %s",
00910                         gettext ("Variable"),
00911                         input->nvariables + 1,
00912                         gettext ("maximum range not allowed"));
00913               msg = buffer2;
00914               goto exit_on_error;
00915             }
```

```
00916         }
00917       else
00918         {
00919           snprintf (buffer2, 64, "%s %u: %s",
00920                     gettext ("Variable"),
00921                     input->nvariables + 1, gettext ("no maximum range"));
00922           msg = buffer2;
00923           goto exit_on_error;
00924         }
00925       if (input->rangemax[input->nvariables]
00926           < input->rangemin[input->nvariables])
00927         {
00928           snprintf (buffer2, 64, "%s %u: %s",
00929                     gettext ("Variable"),
00930                     input->nvariables + 1, gettext ("bad range"));
00931           msg = buffer2;
00932           goto exit_on_error;
00933         }
00934       input->precision = g_realloc
00935         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00936       if (xmlHasProp (child, XML_PRECISION))
00937         input->precision[input->nvariables]
00938           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00939       else
00940         input->precision[input->nvariables] =
    DEFAULT_PRECISION;
00941       if (input->algorithm == ALGORITHM_SWEEP)
00942         {
00943           if (xmlHasProp (child, XML_NSWEEPS))
00944             {
00945               input->nsweeps = (unsigned int *)
00946                 g_realloc (input->nsweeps,
00947                            (1 + input->nvariables) * sizeof (unsigned int));
00948               input->nsweeps[input->nvariables]
00949                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00950             }
00951           else
00952             {
00953               snprintf (buffer2, 64, "%s %u: %s",
00954                         gettext ("Variable"),
00955                         input->nvariables + 1, gettext ("no sweeps number"));
00956               msg = buffer2;
00957               goto exit_on_error;
00958             }
00959 #if DEBUG
00960           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00961                    input->nsweeps[input->nvariables],
    input->nsimulations);
00962 #endif
00963         }
00964       if (input->algorithm == ALGORITHM_GENETIC)
00965         {
00966           // Obtaining bits representing each variable
00967           if (xmlHasProp (child, XML_NBITS))
00968             {
00969               input->nbits = (unsigned int *)
00970                 g_realloc (input->nbits,
00971                            (1 + input->nvariables) * sizeof (unsigned int));
00972               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00973               if (error_code || !i)
00974                 {
00975                   snprintf (buffer2, 64, "%s %u: %s",
00976                             gettext ("Variable"),
00977                             input->nvariables + 1,
00978                             gettext ("invalid bits number"));
00979                   msg = buffer2;
00980                   goto exit_on_error;
00981                 }
00982               input->nbits[input->nvariables] = i;
00983             }
00984           else
00985             {
00986               snprintf (buffer2, 64, "%s %u: %s",
00987                         gettext ("Variable"),
00988                         input->nvariables + 1, gettext ("no bits number"));
00989               msg = buffer2;
00990               goto exit_on_error;
00991             }
00992         }
00993       ++input->nvariables;
00994     }
00995   if (!input->nvariables)
00996     {
00997       msg = gettext ("No calibration variables");
00998       goto exit_on_error;
00999     }
01000
```

```
01001   // Getting the working directory
01002   input->directory = g_path_get_dirname (filename);
01003   input->name = g_path_get_basename (filename);
01004
01005   // Closing the XML document
01006   xmlFreeDoc (doc);
01007
01008 #if DEBUG
01009   fprintf (stderr, "input_open: end\n");
01010 #endif
01011   return 1;
01012
01013 exit_on_error:
01014   show_error (msg);
01015   input_free ();
01016 #if DEBUG
01017   fprintf (stderr, "input_open: end\n");
01018 #endif
01019   return 0;
01020 }
```

Here is the call graph for this function:



5.3.3.10 void show_error (char * *msg*)

Function to show a dialog with an error message.

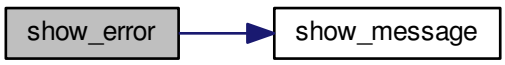
Parameters

msg	Error message.

Definition at line 246 of file calibrator.c.

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

Here is the call graph for this function:



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

Function to show a dialog with a message.

Parameters

title	Title.
msg	Message.
type	Message type.

Definition at line 216 of file calibrator.c.

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

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

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

Parameters

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

Returns

Floating point number value.

Definition at line 325 of file calibrator.c.

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

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

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

Parameters

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

Returns

Integer number value.

Definition at line 263 of file calibrator.c.

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

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 294 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 392 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 354 of file calibrator.c.

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

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

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

Parameters

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

Definition at line 373 of file calibrator.c.

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

5.4 calibrator.h

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
```

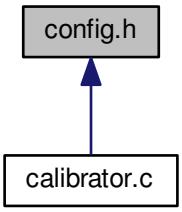
```
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef CALIBRATOR__H
00037 #define CALIBRATOR__H 1
00038
00043 enum Algorithm
00044 {
00045   ALGORITHM_MONTE_CARLO = 0,
00046   ALGORITHM_SWEEP = 1,
00047   ALGORITHM_GENETIC = 2
00048 };
00049
00054 typedef struct
00055 {
00056   char *simulator;
00057   char *evaluator;
00059   char **experiment;
00060   char **template[MAX_NINPUTS];
00061   char **label;
00062   char *directory;
00063   char *name;
00064   double *rangemin;
00065   double *rangemax;
00066   double *rangeminabs;
00067   double *rangemaxabs;
00068   double *weight;
00069   double tolerance;
00070   double mutation_ratio;
00071   double reproduction_ratio;
00072   double adaptation_ratio;
00073   unsigned long int seed;
00075   unsigned int nvariables;
00076   unsigned int nexperiments;
00077   unsigned int ninputs;
00078   unsigned int nsimulations;
00079   unsigned int algorithm;
00080   unsigned int *precision;
00081   unsigned int *nsweeps;
00082   unsigned int *nbits;
00084   unsigned int niterations;
00085   unsigned int nbest;
00086 } Input;
00087
00092 typedef struct
00093 {
00094   char *simulator;
00095   char *evaluator;
00097   char **experiment;
00098   char **template[MAX_NINPUTS];
00099   char **label;
00100   unsigned int nvariables;
00101   unsigned int nexperiments;
00102   unsigned int ninputs;
00103   unsigned int nsimulations;
00104   unsigned int algorithm;
00105   unsigned int *precision;
00106   unsigned int *nsweeps;
00107   unsigned int nstart;
00108   unsigned int nend;
00109   unsigned int *thread;
00111   unsigned int niterations;
00112   unsigned int nbest;
00113   unsigned int nsaveds;
00114   unsigned int *simulation_best;
00115   unsigned long int seed;
00117   double *value;
00118   double *rangemin;
00119   double *rangemax;
00120   double *rangeminabs;
00121   double *rangemaxabs;
00122   double *error_best;
00123   double *weight;
00124   double *value_old;
```

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

5.5 config.h File Reference

Configuration header file.

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



Macros

- #define MAX_NINPUTS 8

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

  Number of algorithms.
- #define NPRECISIONS 15

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

  Default precision digits.
- #define DEFAULT_RANDOM_SEED 7007

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

  Locales directory.
- #define PROGRAM_INTERFACE "calibrator"

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

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

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

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

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

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

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

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

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

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

maximum XML label.

- #define XML_MONTE_CARLO (const xmlChar*)"Monte-Carlo"

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  template6 XML label.
- #define XML_TEMPLATE7 (const xmlChar*)"template7"

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

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

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

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

  weight XML label.

5.5.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file config.h.

5.6 config.h

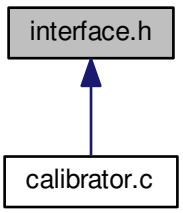
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary forms, with or without modification,
00010 are permitted provided that the following conditions are met:
00011
00012     1. Redistributions of source code must retain the above copyright notice,
00013        this list of conditions and the following disclaimer.
00014
00015     2. Redistributions in binary form must reproduce the above copyright notice,
00016        this list of conditions and the following disclaimer in the
00017        documentation and/or other materials provided with the distribution.
00018
00019 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00020 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00021 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00022 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00023 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00024 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00025 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00026 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00027 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00028 OF SUCH DAMAGE.
00029 */
00030
00037 #ifndef CONFIG__H
00038 #define CONFIG__H 1
00039
00040 // Array sizes
00041
00042 #define MAX_NINPUTS 8
00043 #define NALGORITHMS 3
00045 #define NPRECISIONS 15
00046
00047 // Default choices
00048
00049 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00050 #define DEFAULT_RANDOM_SEED 7007
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "calibrator"
00056
00057 // XML labels
00058
00059 #define XML_ABSOLUTE_MINIMUM (const xmlChar*)"absolute_minimum"
00060 #define XML_ABSOLUTE_MAXIMUM (const xmlChar*)"absolute_maximum"
00062 #define XML_ADAPTATION (const xmlChar*)"adaptation"
00064 #define XML_ALGORITHM (const xmlChar*)"algorithm"
00066 #define XML_CALIBRATE (const xmlChar*)"calibrate"
00068 #define XML_EVALUATOR (const xmlChar*)"evaluator"
00070 #define XML_EXPERIMENT (const xmlChar*)"experiment"
00072 #define XML_GENETIC (const xmlChar*)"genetic"
00074 #define XML_MINIMUM (const xmlChar*)"minimum"
00075 #define XML_MAXIMUM (const xmlChar*)"maximum"
```

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

5.7 interface.h File Reference

Header file of the interface.

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



Data Structures

- struct Experiment

  Struct to define experiment data.
- struct Variable

  Struct to define variable data.
- struct Options

  Struct to define the options dialog.
- struct Running

  Struct to define the running dialog.
- struct Window

  Struct to define the main window.

Macros

- #define MAX_LENGTH (DEFAULT_PRECISION + 8)

  Max length of texts allowed in GtkSpinButtons.

Functions

- void input_save (char *filename)

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

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

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

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

  Function to run a calibration.
- void window_help ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to update the variable rangemin in the main window.

- void window_rangemax_variable ()

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

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

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

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

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

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

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

  Function to obtain the cores number.

5.7.1 Detailed Description

Header file of the interface.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.7.2 Function Documentation

5.7.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 3904 of file calibrator.c.

```
03905 {
03906 #ifdef G_OS_WIN32
03907   SYSTEM_INFO sysinfo;
03908   GetSystemInfo (&sysinfo);
03909   return sysinfo.dwNumberOfProcessors;
03910 #else
03911   return (int) sysconf (_SC_NPROCESSORS_ONLN);
03912 #endif
03913 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

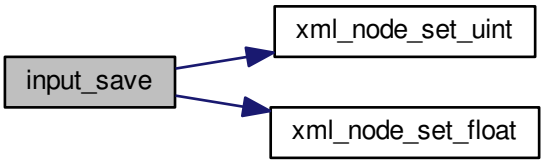
filename	Input file name.

Definition at line 2142 of file calibrator.c.

```
02143 {
02144   unsigned int i, j;
02145   char *buffer;
02146   xmlDoc *doc;
02147   xmlNode *node, *child;
02148   GFile *file, *file2;
02149
02150   // Getting the input file directory
02151   input->name = g_path_get_basename (filename);
02152   input->directory = g_path_get_dirname (filename);
02153   file = g_file_new_for_path (input->directory);
02154
02155   // Opening the input file
02156   doc = xmlNewDoc ((const xmlChar *) "1.0");
02157
02158   // Setting root XML node
02159   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02160   xmlDocSetRootElement (doc, node);
02161
02162   // Adding properties to the root XML node
02163   file2 = g_file_new_for_path (input->simulator);
02164   buffer = g_file_get_relative_path (file, file2);
02165   g_object_unref (file2);
02166   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02167   g_free (buffer);
02168   if (input->evaluator)
02169     {
02170       file2 = g_file_new_for_path (input->evaluator);
02171       buffer = g_file_get_relative_path (file, file2);
02172       g_object_unref (file2);
02173       if (xmlStrlen ((xmlChar *) buffer))
02174         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02175       g_free (buffer);
02176     }
02177   if (input->seed != DEFAULT_RANDOM_SEED)
02178     xml_node_set_uint (node, XML_SEED, input->seed);
02179
02180   // Setting the algorithm
02181   buffer = (char *) g_malloc (64);
02182   switch (input->algorithm)
02183     {
02184     case ALGORITHM_MONTE_CARLO:
02185       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02186       snprintf (buffer, 64, "%u", input->nsimulations);
02187       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02188       snprintf (buffer, 64, "%u", input->niterations);
02189       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02190       snprintf (buffer, 64, "%.3lg", input->tolerance);
02191       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02192       snprintf (buffer, 64, "%u", input->nbest);
02193       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02194       break;
02195     case ALGORITHM_SWEEP:
02196       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02197       snprintf (buffer, 64, "%u", input->niterations);
02198       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02199       snprintf (buffer, 64, "%.3lg", input->tolerance);
02200       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02201       snprintf (buffer, 64, "%u", input->nbest);
02202       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02203       break;
02204     default:
02205       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02206       snprintf (buffer, 64, "%u", input->nsimulations);
02207       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02208       snprintf (buffer, 64, "%u", input->niterations);
02209       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02210       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02211       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02212       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02213       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02214       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02215       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02216       break;
02217     }
02218   g_free (buffer);
02219
02220   // Setting the experimental data
02221   for (i = 0; i < input->nexperiments; ++i)
02222     {
```

```
02223       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02224       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02225       if (input->weight[i] != 1.)
02226         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02227       for (j = 0; j < input->ninputs; ++j)
02228         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02229     }
02230
02231   // Setting the variables data
02232   for (i = 0; i < input->nvariables; ++i)
02233     {
02234       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02235       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02236       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02237       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02238         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02239       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02240       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02241         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02242       if (input->precision[i] != DEFAULT_PRECISION)
02243         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02244       if (input->algorithm == ALGORITHM_SWEEP)
02245         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02246       else if (input->algorithm == ALGORITHM_GENETIC)
02247         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02248     }
02249
02250   // Saving the XML file
02251   xmlSaveFormatFile (filename, doc, 1);
02252
02253   // Freeing memory
02254   xmlFreeDoc (doc);
02255 }
```

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 2515 of file calibrator.c.

```
02516 {
02517   unsigned int i;
02518   for (i = 0; i < NALGORITHMS; ++i)
```

```
02519     if (gtk_toggle_button_get_active
02520         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02521       break;
02522   return i;
02523 }
```

5.7.2.4 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

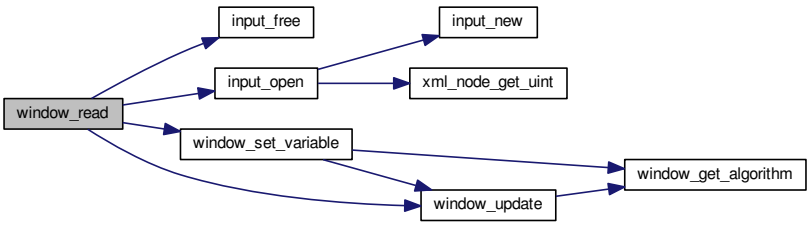
1 on succes, 0 on error.

Definition at line 3266 of file calibrator.c.

```
03267 {
03268   unsigned int i;
03269   char *buffer;
03270 #if DEBUG
03271   fprintf (stderr, "window_read: start\n");
03272 #endif
03273
03274   // Reading new input file
03275   input_free ();
03276   if (!input_open (filename))
03277     return 0;
03278
03279   // Setting GTK+ widgets data
03280   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03281   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03282                                  (window->button_simulator), buffer);
03283   g_free (buffer);
03284   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03285                                 (size_t) input->evaluator);
03286   if (input->evaluator)
03287     {
03288       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03289       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03290                                      (window->button_evaluator), buffer);
03291       g_free (buffer);
03292     }
03293   gtk_toggle_button_set_active
03294     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03295   switch (input->algorithm)
03296     {
03297     case ALGORITHM_MONTE_CARLO:
03298       gtk_spin_button_set_value (window->spin_simulations,
03299                                  (gdouble) input->nsimulations);
03300     case ALGORITHM_SWEEP:
03301       gtk_spin_button_set_value (window->spin_iterations,
03302                                  (gdouble) input->niterations);
03303       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03304       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03305       break;
03306     default:
03307       gtk_spin_button_set_value (window->spin_population,
03308                                  (gdouble) input->nsimulations);
03309       gtk_spin_button_set_value (window->spin_generations,
03310                                  (gdouble) input->niterations);
03311       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03312       gtk_spin_button_set_value (window->spin_reproduction,
03313                                  input->reproduction_ratio);
03314       gtk_spin_button_set_value (window->spin_adaptation,
03315                                  input->adaptation_ratio);
03316     }
03317   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
```

```
03318   g_signal_handler_block (window->button_experiment,
03319                           window->id_experiment_name);
03320   gtk_combo_box_text_remove_all (window->combo_experiment);
03321   for (i = 0; i < input->nexperiments; ++i)
03322     gtk_combo_box_text_append_text (window->combo_experiment,
03323                                     input->experiment[i]);
03324   g_signal_handler_unblock
03325     (window->button_experiment, window->
      id_experiment_name);
03326   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03327   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03328   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03329   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03330   gtk_combo_box_text_remove_all (window->combo_variable);
03331   for (i = 0; i < input->nvariables; ++i)
03332     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03333   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03334   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03335   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03336   window_set_variable ();
03337   window_update ();
03338
03339 #if DEBUG
03340   fprintf (stderr, "window_read: end\n");
03341 #endif
03342   return 1;
03343 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

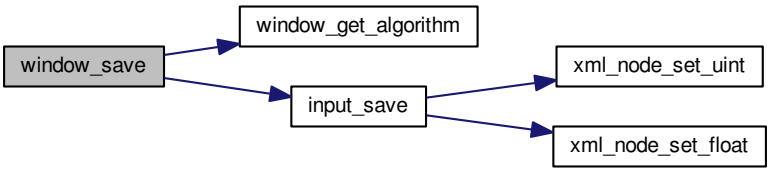
1 on OK, 0 on Cancel.

Definition at line 2330 of file calibrator.c.

```
02331 {
02332   char *buffer;
02333   GtkFileChooserDialog *dlg;
02334
02335 #if DEBUG
02336   fprintf (stderr, "window_save: start\n");
02337 #endif
02338
02339   // Opening the saving dialog
02340   dlg = (GtkFileChooserDialog *)
02341     gtk_file_chooser_dialog_new (gettext ("Save file"),
02342                                  window->window,
```

```
02343                                 GTK_FILE_CHOOSER_ACTION_SAVE,
02344                                 gettext ("_Cancel"),
02345                                 GTK_RESPONSE_CANCEL,
02346                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02347   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02348   buffer = g_build_filename (input->directory, input->name, NULL);
02349   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02350   g_free (buffer);
02351
02352   // If OK response then saving
02353   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02354     {
02355
02356       // Adding properties to the root XML node
02357       input->simulator = gtk_file_chooser_get_filename
02358         (GTK_FILE_CHOOSER (window->button_simulator));
02359       if (gtk_toggle_button_get_active
02360           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02361         input->evaluator = gtk_file_chooser_get_filename
02362           (GTK_FILE_CHOOSER (window->button_evaluator));
02363       else
02364         input->evaluator = NULL;
02365
02366       // Setting the algorithm
02367       switch (window_get_algorithm ())
02368         {
02369         case ALGORITHM_MONTE_CARLO:
02370           input->algorithm = ALGORITHM_MONTE_CARLO;
02371           input->nsimulations
02372             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02373           input->niterations
02374             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02375           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02376           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02377           break;
02378         case ALGORITHM_SWEEP:
02379           input->algorithm = ALGORITHM_SWEEP;
02380           input->niterations
02381             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02382           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02383           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02384           break;
02385         default:
02386           input->algorithm = ALGORITHM_GENETIC;
02387           input->nsimulations
02388             = gtk_spin_button_get_value_as_int (window->spin_population);
02389           input->niterations
02390             = gtk_spin_button_get_value_as_int (window->spin_generations);
02391           input->mutation_ratio
02392             = gtk_spin_button_get_value (window->spin_mutation);
02393           input->reproduction_ratio
02394             = gtk_spin_button_get_value (window->spin_reproduction);
02395           input->adaptation_ratio
02396             = gtk_spin_button_get_value (window->spin_adaptation);
02397           break;
02398         }
02399
02400       // Saving the XML file
02401       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02402       input_save (buffer);
02403
02404       // Closing and freeing memory
02405       g_free (buffer);
02406       gtk_widget_destroy (GTK_WIDGET (dlg));
02407 #if DEBUG
02408       fprintf (stderr, "window_save: end\n");
02409 #endif
02410       return 1;
02411     }
02412
02413   // Closing and freeing memory
02414   gtk_widget_destroy (GTK_WIDGET (dlg));
02415 #if DEBUG
02416   fprintf (stderr, "window_save: end\n");
02417 #endif
02418   return 0;
02419 }
```

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 2902 of file calibrator.c.

```
02903 {
02904   unsigned int i, j;
02905   char *buffer;
02906   GFile *file1, *file2;
02907 #if DEBUG
02908   fprintf (stderr, "window_template_experiment: start\n");
02909 #endif
02910   i = (size_t) data;
02911   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02912   file1
02913     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02914   file2 = g_file_new_for_path (input->directory);
02915   buffer = g_file_get_relative_path (file2, file1);
02916   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02917   g_free (buffer);
02918   g_object_unref (file2);
02919   g_object_unref (file1);
02920 #if DEBUG
02921   fprintf (stderr, "window_template_experiment: end\n");
02922 #endif
02923 }
```

5.8 interface.h

```
00001 /*
00002 Calibrator: a software to make calibrations of empirical parameters.
00003
00004 AUTHORS: Javier Burguete and Borja Latorre.
00005
00006 Copyright 2012-2015, AUTHORS.
00007
00008 Redistribution and use in source and binary forms, with or without modification,
00009 are permitted provided that the following conditions are met:
00010
00011     1. Redistributions of source code must retain the above copyright notice,
00012        this list of conditions and the following disclaimer.
00013
00014     2. Redistributions in binary form must reproduce the above copyright notice,
00015        this list of conditions and the following disclaimer in the
00016        documentation and/or other materials provided with the distribution.
00017
00018 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00019 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00020 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00021 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00022 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
```

```
00023 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00024 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00025 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00026 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00027 OF SUCH DAMAGE.
00028 */
00029
00036 #ifndef INTERFACE__H
00037 #define INTERFACE__H 1
00038
00039 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00040
00046 typedef struct
00047 {
00048   char *template[MAX_NINPUTS];
00049   char *name;
00050   double weight;
00052 } Experiment;
00053
00058 typedef struct
00059 {
00060   char *label;
00061   double rangemin;
00062   double rangemax;
00063   double rangeminabs;
00064   double rangemaxabs;
00065   unsigned int precision;
00066   unsigned int nsweeps;
00067   unsigned int nbits;
00068 } Variable;
00069
00074 typedef struct
00075 {
00076   GtkDialog *dialog;
00077   GtkGrid *grid;
00078   GtkLabel *label_processors;
00079   GtkSpinButton *spin_processors;
00080   GtkLabel *label_seed;
00082   GtkSpinButton *spin_seed;
00084 } Options;
00085
00090 typedef struct
00091 {
00092   GtkDialog *dialog;
00093   GtkLabel *label;
00094 } Running;
00095
00100 typedef struct
00101 {
00102   GtkWindow *window;
00103   GtkGrid *grid;
00104   GtkToolbar *bar_buttons;
00105   GtkToolButton *button_open;
00106   GtkToolButton *button_save;
00107   GtkToolButton *button_run;
00108   GtkToolButton *button_options;
00109   GtkToolButton *button_help;
00110   GtkToolButton *button_about;
00111   GtkToolButton *button_exit;
00112   GtkLabel *label_simulator;
00113   GtkFileChooserButton *button_simulator;
00115   GtkCheckButton *check_evaluator;
00116   GtkFileChooserButton *button_evaluator;
00118   GtkFrame *frame_algorithm;
00119   GtkGrid *grid_algorithm;
00120   GtkRadioButton *button_algorithm[NALGORITHMS];
00122   GtkLabel *label_simulations;
00123   GtkSpinButton *spin_simulations;
00125   GtkLabel *label_iterations;
00126   GtkSpinButton *spin_iterations;
00128   GtkLabel *label_tolerance;
00129   GtkSpinButton *spin_tolerance;
00130   GtkLabel *label_bests;
00131   GtkSpinButton *spin_bests;
00132   GtkLabel *label_population;
00133   GtkSpinButton *spin_population;
00135   GtkLabel *label_generations;
00136   GtkSpinButton *spin_generations;
00138   GtkLabel *label_mutation;
00139   GtkSpinButton *spin_mutation;
00140   GtkLabel *label_reproduction;
00141   GtkSpinButton *spin_reproduction;
00143   GtkLabel *label_adaptation;
00144   GtkSpinButton *spin_adaptation;
00146   GtkFrame *frame_variable;
00147   GtkGrid *grid_variable;
00148   GtkComboBoxText *combo_variable;
```

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

Index