Calibrator

1.0.6

Generated by Doxygen 1.8.9.1

Thu Nov 19 2015 06:19:56

Contents

Chapter 1

CALIBRATOR

A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 1.0.6: Stable and recommended version.
- 1.1.39: 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/1.0.6
$ 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 * result
  Name of the result file.
- char * variables
  Name of the variables file.
- 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.
- double calculation_time

  Calculation time.
- 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 94 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 * result
  Name of the result file.
- char * variables
  Name of the variables file.

- 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 152 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:

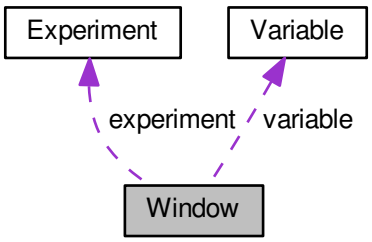
- interface.h

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.
- GtkGrid * grid_files

  Files GtkGrid.
- GtkLabel * label_simulator

  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator

  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator

  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator

  Evaluator program GtkFileChooserButton.
- GtkLabel * label_result

  Result file GtkLabel.
- GtkEntry * entry_result

  Result file GtkEntry.
- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- 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

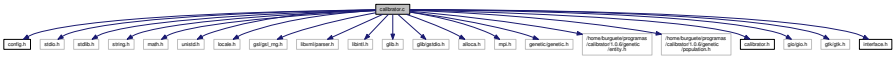
File Documentation

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 * result_name = (xmlChar *) "result"

  Name of the result file.
- const xmlChar * variables_name = (xmlChar *) "variables"

  Name of the variables file.

- 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 1341 of file calibrator.c.

```
01342 {
01343   unsigned int i, j;
01344   double e;
01345 #if DEBUG
01346   fprintf (stderr, "calibrate_best_sequential: start\n");
01347 #endif
01348   if (calibrate->nsaveds < calibrate->nbest
01349       || value < calibrate->error_best[calibrate->nsaveds - 1])
01350     {
01351       if (calibrate->nsaveds < calibrate->nbest)
01352         ++calibrate->nsaveds;
01353       calibrate->error_best[calibrate->nsaveds - 1] = value;
01354       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01355       for (i = calibrate->nsaveds; --i;)
01356         {
01357           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01358             {
01359               j = calibrate->simulation_best[i];
```

```
01360               e = calibrate->error_best[i];
01361               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01362               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01363               calibrate->simulation_best[i - 1] = j;
01364               calibrate->error_best[i - 1] = e;
01365             }
01366           else
01367             break;
01368         }
01369     }
01370 #if DEBUG
01371   fprintf (stderr, "calibrate_best_sequential: end\n");
01372 #endif
01373 }
```

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 1296 of file calibrator.c.

```
01297 {
01298   unsigned int i, j;
01299   double e;
01300 #if DEBUG
01301   fprintf (stderr, "calibrate_best_thread: start\n");
01302 #endif
01303   if (calibrate->nsaveds < calibrate->nbest
01304       || value < calibrate->error_best[calibrate->nsaveds - 1])
01305     {
01306       g_mutex_lock (mutex);
01307       if (calibrate->nsaveds < calibrate->nbest)
01308         ++calibrate->nsaveds;
01309       calibrate->error_best[calibrate->nsaveds - 1] = value;
01310       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01311       for (i = calibrate->nsaveds; --i;)
01312         {
01313           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01314             {
01315               j = calibrate->simulation_best[i];
01316               e = calibrate->error_best[i];
01317               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01318               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01319               calibrate->simulation_best[i - 1] = j;
01320               calibrate->error_best[i - 1] = e;
01321             }
01322           else
01323             break;
01324         }
01325       g_mutex_unlock (mutex);
01326     }
01327 #if DEBUG
01328   fprintf (stderr, "calibrate_best_thread: end\n");
01329 #endif
01330 }
```

5.1.2.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

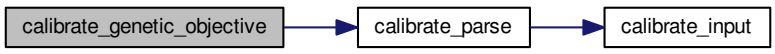
Returns

objective function value.

Definition at line 1650 of file calibrator.c.

```
01651 {
01652   unsigned int j;
01653   double objective;
01654   char buffer[64];
01655 #if DEBUG
01656   fprintf (stderr, "calibrate_genetic_objective: start\n");
01657 #endif
01658   for (j = 0; j < calibrate->nvariables; ++j)
01659     {
01660       calibrate->value[entity->id * calibrate->nvariables + j]
01661         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01662     }
01663   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01664     objective += calibrate_parse (entity->id, j);
01665   g_mutex_lock (mutex);
01666   for (j = 0; j < calibrate->nvariables; ++j)
01667     {
01668       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01669       fprintf (calibrate->file_variables, buffer,
01670                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01671     }
01672   fprintf (calibrate->file_variables, "%.14le\n", objective);
01673   g_mutex_unlock (mutex);
01674 #if DEBUG
01675   fprintf (stderr, "calibrate_genetic_objective: end\n");
01676 #endif
01677   return objective;
01678 }
```

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 1049 of file calibrator.c.

```
01050 {
01051   unsigned int i;
01052   char buffer[32], value[32], *buffer2, *buffer3, *content;
01053   FILE *file;
01054   gsize length;
01055   GRegex *regex;
01056
```

```
01057 #if DEBUG
01058   fprintf (stderr, "calibrate_input: start\n");
01059 #endif
01060
01061   // Checking the file
01062   if (!template)
01063     goto calibrate_input_end;
01064
01065   // Opening template
01066   content = g_mapped_file_get_contents (template);
01067   length = g_mapped_file_get_length (template);
01068 #if DEBUG
01069   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01070            content);
01071 #endif
01072   file = g_fopen (input, "w");
01073
01074   // Parsing template
01075   for (i = 0; i < calibrate->nvariables; ++i)
01076     {
01077 #if DEBUG
01078       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01079 #endif
01080       snprintf (buffer, 32, "@variable%u@", i + 1);
01081       regex = g_regex_new (buffer, 0, 0, NULL);
01082       if (i == 0)
01083         {
01084           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01085                                              calibrate->label[i], 0, NULL);
01086 #if DEBUG
01087           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01088 #endif
01089         }
01090       else
01091         {
01092           length = strlen (buffer3);
01093           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01094                                              calibrate->label[i], 0, NULL);
01095           g_free (buffer3);
01096         }
01097       g_regex_unref (regex);
01098       length = strlen (buffer2);
01099       snprintf (buffer, 32, "@value%u@", i + 1);
01100       regex = g_regex_new (buffer, 0, 0, NULL);
01101       snprintf (value, 32, format[calibrate->precision[i]],
01102                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01103
01104 #if DEBUG
01105       fprintf (stderr, "calibrate_input: value=%s\n", value);
01106 #endif
01107       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01108                                          0, NULL);
01109       g_free (buffer2);
01110       g_regex_unref (regex);
01111     }
01112
01113   // Saving input file
01114   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01115   g_free (buffer3);
01116   fclose (file);
01117
01118 calibrate_input_end:
01119 #if DEBUG
01120   fprintf (stderr, "calibrate_input: end\n");
01121 #endif
01122   return;
01123 }
```

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 1457 of file calibrator.c.

```
01459 {
01460   unsigned int i, j, k, s[calibrate->nbest];
01461   double e[calibrate->nbest];
01462 #if DEBUG
01463   fprintf (stderr, "calibrate_merge: start\n");
01464 #endif
01465   i = j = k = 0;
01466   do
01467     {
01468       if (i == calibrate->nsaveds)
01469         {
01470           s[k] = simulation_best[j];
01471           e[k] = error_best[j];
01472           ++j;
01473           ++k;
01474           if (j == nsaveds)
01475             break;
01476         }
01477       else if (j == nsaveds)
01478         {
01479           s[k] = calibrate->simulation_best[i];
01480           e[k] = calibrate->error_best[i];
01481           ++i;
01482           ++k;
01483           if (i == calibrate->nsaveds)
01484             break;
01485         }
01486       else if (calibrate->error_best[i] > error_best[j])
01487         {
01488           s[k] = simulation_best[j];
01489           e[k] = error_best[j];
01490           ++j;
01491           ++k;
01492         }
01493       else
01494         {
01495           s[k] = calibrate->simulation_best[i];
01496           e[k] = calibrate->error_best[i];
01497           ++i;
01498           ++k;
01499         }
01500     }
01501   while (k < calibrate->nbest);
01502   calibrate->nsaveds = k;
01503   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01504   memcpy (calibrate->error_best, e, k * sizeof (double));
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_merge: end\n");
01507 #endif
01508 }
```

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 1136 of file calibrator.c.

```
01137 {
01138   unsigned int i;
01139   double e;
01140   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
```

```
01141     *buffer3, *buffer4;
01142   FILE *file_result;
01143
01144 #if DEBUG
01145   fprintf (stderr, "calibrate_parse: start\n");
01146   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01147            experiment);
01148 #endif
01149
01150   // Opening input files
01151   for (i = 0; i < calibrate->ninputs; ++i)
01152     {
01153       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01154 #if DEBUG
01155       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01156 #endif
01157       calibrate_input (simulation, &input[i][0],
01158                        calibrate->file[i][experiment]);
01159     }
01160   for (; i < MAX_NINPUTS; ++i)
01161     strcpy (&input[i][0], "");
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: parsing end\n");
01164 #endif
01165
01166   // Performing the simulation
01167   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01168   buffer2 = g_path_get_dirname (calibrate->simulator);
01169   buffer3 = g_path_get_basename (calibrate->simulator);
01170   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01171   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01172             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01173             input[6], input[7], output);
01174   g_free (buffer4);
01175   g_free (buffer3);
01176   g_free (buffer2);
01177 #if DEBUG
01178   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01179 #endif
01180   system (buffer);
01181
01182   // Checking the objective value function
01183   if (calibrate->evaluator)
01184     {
01185       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01186       buffer2 = g_path_get_dirname (calibrate->evaluator);
01187       buffer3 = g_path_get_basename (calibrate->evaluator);
01188       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01189       snprintf (buffer, 512, "\"%s\" %s %s %s",
01190                 buffer4, output, calibrate->experiment[experiment], result);
01191       g_free (buffer4);
01192       g_free (buffer3);
01193       g_free (buffer2);
01194 #if DEBUG
01195       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01196 #endif
01197       system (buffer);
01198       file_result = g_fopen (result, "r");
01199       e = atof (fgets (buffer, 512, file_result));
01200       fclose (file_result);
01201     }
01202   else
01203     {
01204       strcpy (result, "");
01205       file_result = g_fopen (output, "r");
01206       e = atof (fgets (buffer, 512, file_result));
01207       fclose (file_result);
01208     }
01209
01210   // Removing files
01211 #if !DEBUG
01212   for (i = 0; i < calibrate->ninputs; ++i)
01213     {
01214       if (calibrate->file[i][0])
01215         {
01216           snprintf (buffer, 512, RM " %s", &input[i][0]);
01217           system (buffer);
01218         }
01219     }
01220   snprintf (buffer, 512, RM " %s %s", output, result);
01221   system (buffer);
01222 #endif
01223
01224 #if DEBUG
01225   fprintf (stderr, "calibrate_parse: end\n");
01226 #endif
01227
```

```
01228   // Returning the objective function
01229   return e * calibrate->weight[experiment];
01230 }
```

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 1268 of file calibrator.c.

```
01269 {
01270   unsigned int i;
01271   char buffer[64];
01272 #if DEBUG
01273   fprintf (stderr, "calibrate_save_variables: start\n");
01274 #endif
01275   for (i = 0; i < calibrate->nvariables; ++i)
01276     {
01277       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01278       fprintf (calibrate->file_variables, buffer,
01279                calibrate->value[simulation * calibrate->
      nvariables + i]);
01280     }
01281   fprintf (calibrate->file_variables, "%.14le\n", error);
01282 #if DEBUG
01283   fprintf (stderr, "calibrate_save_variables: end\n");
01284 #endif
01285 }
```

5.1.2.8 void * calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

Returns

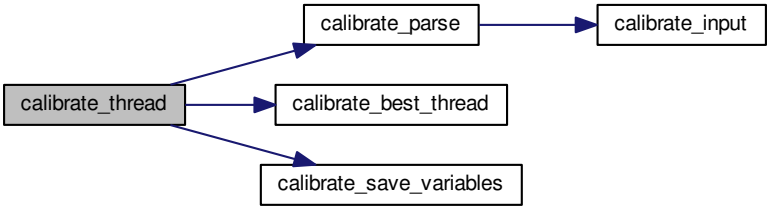
NULL

Definition at line 1383 of file calibrator.c.

```
01384 {
01385   unsigned int i, j, thread;
01386   double e;
01387 #if DEBUG
```

```
01388   fprintf (stderr, "calibrate_thread: start\n");
01389 #endif
01390   thread = data->thread;
01391 #if DEBUG
01392   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01393            calibrate->thread[thread], calibrate->thread[thread + 1]);
01394 #endif
01395   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01396     {
01397       e = 0.;
01398       for (j = 0; j < calibrate->nexperiments; ++j)
01399         e += calibrate_parse (i, j);
01400       calibrate_best_thread (i, e);
01401       g_mutex_lock (mutex);
01402       calibrate_save_variables (i, e);
01403       g_mutex_unlock (mutex);
01404 #if DEBUG
01405       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01406 #endif
01407     }
01408 #if DEBUG
01409   fprintf (stderr, "calibrate_thread: end\n");
01410 #endif
01411   g_thread_exit (NULL);
01412   return NULL;
01413 }
```

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 4046 of file calibrator.c.

```
04047 {
04048 #ifdef G_OS_WIN32
04049   SYSTEM_INFO sysinfo;
04050   GetSystemInfo (&sysinfo);
04051   return sysinfo.dwNumberOfProcessors;
04052 #else
04053   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04054 #endif
04055 }
```

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 479 of file calibrator.c.

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

```
00553       if (error_code)
00554         {
00555           msg = gettext ("Bad simulations number");
00556           goto exit_on_error;
00557         }
00558     }
00559   else if (!xmlStrcmp (buffer, XML_SWEEP))
00560     input->algorithm = ALGORITHM_SWEEP;
00561   else if (!xmlStrcmp (buffer, XML_GENETIC))
00562     {
00563       input->algorithm = ALGORITHM_GENETIC;
00564
00565       // Obtaining population
00566       if (xmlHasProp (node, XML_NPOPULATION))
00567         {
00568           input->nsimulations
00569             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00570           if (error_code || input->nsimulations < 3)
00571             {
00572               msg = gettext ("Invalid population number");
00573               goto exit_on_error;
00574             }
00575         }
00576       else
00577         {
00578           msg = gettext ("No population number");
00579           goto exit_on_error;
00580         }
00581
00582       // Obtaining generations
00583       if (xmlHasProp (node, XML_NGENERATIONS))
00584         {
00585           input->niterations
00586             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00587           if (error_code || !input->niterations)
00588             {
00589               msg = gettext ("Invalid generations number");
00590               goto exit_on_error;
00591             }
00592         }
00593       else
00594         {
00595           msg = gettext ("No generations number");
00596           goto exit_on_error;
00597         }
00598
00599       // Obtaining mutation probability
00600       if (xmlHasProp (node, XML_MUTATION))
00601         {
00602           input->mutation_ratio
00603             = xml_node_get_float (node, XML_MUTATION, &error_code);
00604           if (error_code || input->mutation_ratio < 0.
00605               || input->mutation_ratio >= 1.)
00606             {
00607               msg = gettext ("Invalid mutation probability");
00608               goto exit_on_error;
00609             }
00610         }
00611       else
00612         {
00613           msg = gettext ("No mutation probability");
00614           goto exit_on_error;
00615         }
00616
00617       // Obtaining reproduction probability
00618       if (xmlHasProp (node, XML_REPRODUCTION))
00619         {
00620           input->reproduction_ratio
00621             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00622           if (error_code || input->reproduction_ratio < 0.
00623               || input->reproduction_ratio >= 1.0)
00624             {
00625               msg = gettext ("Invalid reproduction probability");
00626               goto exit_on_error;
00627             }
00628         }
00629       else
00630         {
00631           msg = gettext ("No reproduction probability");
00632           goto exit_on_error;
00633         }
00634
00635       // Obtaining adaptation probability
00636       if (xmlHasProp (node, XML_ADAPTATION))
00637         {
00638           input->adaptation_ratio
00639             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
```

```
00640           if (error_code || input->adaptation_ratio < 0.
00641               || input->adaptation_ratio >= 1.)
00642             {
00643               msg = gettext ("Invalid adaptation probability");
00644               goto exit_on_error;
00645             }
00646         }
00647       else
00648         {
00649           msg = gettext ("No adaptation probability");
00650           goto exit_on_error;
00651         }
00652
00653       // Checking survivals
00654       i = input->mutation_ratio * input->nsimulations;
00655       i += input->reproduction_ratio * input->
      nsimulations;
00656       i += input->adaptation_ratio * input->
      nsimulations;
00657       if (i > input->nsimulations - 2)
00658         {
00659           msg = gettext
00660             ("No enough survival entities to reproduce the population");
00661           goto exit_on_error;
00662         }
00663     }
00664   else
00665     {
00666       msg = gettext ("Unknown algorithm");
00667       goto exit_on_error;
00668     }
00669
00670   if (input->algorithm == ALGORITHM_MONTE_CARLO
00671       || input->algorithm == ALGORITHM_SWEEP)
00672     {
00673
00674       // Obtaining iterations number
00675       input->niterations
00676         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00677       if (error_code == 1)
00678         input->niterations = 1;
00679       else if (error_code)
00680         {
00681           msg = gettext ("Bad iterations number");
00682           goto exit_on_error;
00683         }
00684
00685       // Obtaining best number
00686       if (xmlHasProp (node, XML_NBEST))
00687         {
00688           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00689           if (error_code || !input->nbest)
00690             {
00691               msg = gettext ("Invalid best number");
00692               goto exit_on_error;
00693             }
00694         }
00695       else
00696         input->nbest = 1;
00697
00698       // Obtaining tolerance
00699       if (xmlHasProp (node, XML_TOLERANCE))
00700         {
00701           input->tolerance
00702             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00703           if (error_code || input->tolerance < 0.)
00704             {
00705               msg = gettext ("Invalid tolerance");
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         input->tolerance = 0.;
00711     }
00712
00713   // Reading the experimental data
00714   for (child = node->children; child; child = child->next)
00715     {
00716       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00717         break;
00718 #if DEBUG
00719       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00720 #endif
00721       if (xmlHasProp (child, XML_NAME))
00722         {
00723           input->experiment
```

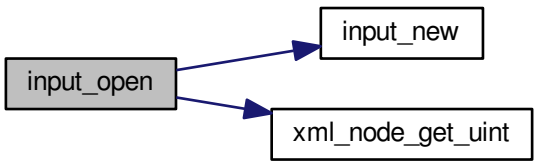
```
00724             = g_realloc (input->experiment,
00725                          (1 + input->nexperiments) * sizeof (char *));
00726           input->experiment[input->nexperiments]
00727             = (char *) xmlGetProp (child, XML_NAME);
00728         }
00729       else
00730         {
00731           snprintf (buffer2, 64, "%s %u: %s",
00732                     gettext ("Experiment"),
00733                     input->nexperiments + 1, gettext ("no data file name"));
00734           msg = buffer2;
00735           goto exit_on_error;
00736         }
00737 #if DEBUG
00738       fprintf (stderr, "input_open: experiment=%s\n",
00739                input->experiment[input->nexperiments]);
00740 #endif
00741       input->weight = g_realloc (input->weight,
00742                                  (1 + input->nexperiments) * sizeof (double));
00743       if (xmlHasProp (child, XML_WEIGHT))
00744         {
00745           input->weight[input->nexperiments]
00746             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00747           if (error_code)
00748             {
00749               snprintf (buffer2, 64, "%s %u: %s",
00750                         gettext ("Experiment"),
00751                         input->nexperiments + 1, gettext ("bad weight"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755         }
00756       else
00757         input->weight[input->nexperiments] = 1.;
00758 #if DEBUG
00759       fprintf (stderr, "input_open: weight=%lg\n",
00760                input->weight[input->nexperiments]);
00761 #endif
00762       if (!input->nexperiments)
00763         input->ninputs = 0;
00764 #if DEBUG
00765       fprintf (stderr, "input_open: template[0]\n");
00766 #endif
00767       if (xmlHasProp (child, XML_TEMPLATE1))
00768         {
00769           input->template[0]
00770             = (char **) g_realloc (input->template[0],
00771                                    (1 + input->nexperiments) * sizeof (char *));
00772           input->template[0][input->nexperiments]
00773             = (char *) xmlGetProp (child, template[0]);
00774 #if DEBUG
00775           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00776                    input->nexperiments,
00777                    input->template[0][input->nexperiments]);
00778 #endif
00779           if (!input->nexperiments)
00780             ++input->ninputs;
00781 #if DEBUG
00782           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00783 #endif
00784         }
00785       else
00786         {
00787           snprintf (buffer2, 64, "%s %u: %s",
00788                     gettext ("Experiment"),
00789                     input->nexperiments + 1, gettext ("no template"));
00790           msg = buffer2;
00791           goto exit_on_error;
00792         }
00793       for (i = 1; i < MAX_NINPUTS; ++i)
00794         {
00795 #if DEBUG
00796           fprintf (stderr, "input_open: template%u\n", i + 1);
00797 #endif
00798           if (xmlHasProp (child, template[i]))
00799             {
00800               if (input->nexperiments && input->ninputs <= i)
00801                 {
00802                   snprintf (buffer2, 64, "%s %u: %s",
00803                             gettext ("Experiment"),
00804                             input->nexperiments + 1,
00805                             gettext ("bad templates number"));
00806                   msg = buffer2;
00807                   goto exit_on_error;
00808                 }
00809               input->template[i] = (char **)
00810                 g_realloc (input->template[i],
```

```
00811                           (1 + input->nexperiments) * sizeof (char *));
00812               input->template[i][input->nexperiments]
00813                 = (char *) xmlGetProp (child, template[i]);
00814 #if DEBUG
00815               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00816                        input->nexperiments, i + 1,
00817                        input->template[i][input->nexperiments]);
00818 #endif
00819               if (!input->nexperiments)
00820                 ++input->ninputs;
00821 #if DEBUG
00822               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00823 #endif
00824             }
00825           else if (input->nexperiments && input->ninputs >= i)
00826             {
00827               snprintf (buffer2, 64, "%s %u: %s%u",
00828                         gettext ("Experiment"),
00829                         input->nexperiments + 1,
00830                         gettext ("no template"), i + 1);
00831               msg = buffer2;
00832               goto exit_on_error;
00833             }
00834           else
00835             break;
00836         }
00837       ++input->nexperiments;
00838 #if DEBUG
00839       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00840 #endif
00841     }
00842   if (!input->nexperiments)
00843     {
00844       msg = gettext ("No calibration experiments");
00845       goto exit_on_error;
00846     }
00847
00848   // Reading the variables data
00849   for (; child; child = child->next)
00850     {
00851       if (xmlStrcmp (child->name, XML_VARIABLE))
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("bad XML node"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_NAME))
00860         {
00861           input->label = g_realloc
00862             (input->label, (1 + input->nvariables) * sizeof (char *));
00863           input->label[input->nvariables]
00864             = (char *) xmlGetProp (child, XML_NAME);
00865         }
00866       else
00867         {
00868           snprintf (buffer2, 64, "%s %u: %s",
00869                     gettext ("Variable"),
00870                     input->nvariables + 1, gettext ("no name"));
00871           msg = buffer2;
00872           goto exit_on_error;
00873         }
00874       if (xmlHasProp (child, XML_MINIMUM))
00875         {
00876           input->rangemin = g_realloc
00877             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00878           input->rangeminabs = g_realloc
00879             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00880           input->rangemin[input->nvariables]
00881             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00882           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00883             {
00884               input->rangeminabs[input->nvariables]
00885                 = xml_node_get_float (child,
    XML_ABSOLUTE_MINIMUM, &error_code);
00886             }
00887           else
00888             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00889           if (input->rangemin[input->nvariables]
00890               < input->rangeminabs[input->nvariables])
00891             {
00892               snprintf (buffer2, 64, "%s %u: %s",
00893                         gettext ("Variable"),
00894                         input->nvariables + 1,
00895                         gettext ("minimum range not allowed"));
00896               msg = buffer2;
```

```
00897               goto exit_on_error;
00898             }
00899         }
00900       else
00901         {
00902           snprintf (buffer2, 64, "%s %u: %s",
00903                     gettext ("Variable"),
00904                     input->nvariables + 1, gettext ("no minimum range"));
00905           msg = buffer2;
00906           goto exit_on_error;
00907         }
00908       if (xmlHasProp (child, XML_MAXIMUM))
00909         {
00910           input->rangemax = g_realloc
00911             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00912           input->rangemaxabs = g_realloc
00913             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00914           input->rangemax[input->nvariables]
00915             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00916           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00917             input->rangemaxabs[input->nvariables]
00918               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00919           else
00920             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00921           if (input->rangemax[input->nvariables]
00922               > input->rangemaxabs[input->nvariables])
00923             {
00924               snprintf (buffer2, 64, "%s %u: %s",
00925                         gettext ("Variable"),
00926                         input->nvariables + 1,
00927                         gettext ("maximum range not allowed"));
00928               msg = buffer2;
00929               goto exit_on_error;
00930             }
00931         }
00932       else
00933         {
00934           snprintf (buffer2, 64, "%s %u: %s",
00935                     gettext ("Variable"),
00936                     input->nvariables + 1, gettext ("no maximum range"));
00937           msg = buffer2;
00938           goto exit_on_error;
00939         }
00940       if (input->rangemax[input->nvariables]
00941           < input->rangemin[input->nvariables])
00942         {
00943           snprintf (buffer2, 64, "%s %u: %s",
00944                     gettext ("Variable"),
00945                     input->nvariables + 1, gettext ("bad range"));
00946           msg = buffer2;
00947           goto exit_on_error;
00948         }
00949       input->precision = g_realloc
00950         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00951       if (xmlHasProp (child, XML_PRECISION))
00952         input->precision[input->nvariables]
00953           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00954       else
00955         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00956       if (input->algorithm == ALGORITHM_SWEEP)
00957         {
00958           if (xmlHasProp (child, XML_NSWEEPS))
00959             {
00960               input->nsweeps = (unsigned int *)
00961                 g_realloc (input->nsweeps,
00962                            (1 + input->nvariables) * sizeof (unsigned int));
00963               input->nsweeps[input->nvariables]
00964                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00965             }
00966           else
00967             {
00968               snprintf (buffer2, 64, "%s %u: %s",
00969                         gettext ("Variable"),
00970                         input->nvariables + 1, gettext ("no sweeps number"));
00971               msg = buffer2;
00972               goto exit_on_error;
00973             }
00974 #if DEBUG
00975           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00976                    input->nsweeps[input->nvariables],
      input->nsimulations);
00977 #endif
00978         }
00979       if (input->algorithm == ALGORITHM_GENETIC)
00980         {
```

```
00981           // Obtaining bits representing each variable
00982           if (xmlHasProp (child, XML_NBITS))
00983             {
00984               input->nbits = (unsigned int *)
00985                 g_realloc (input->nbits,
00986                            (1 + input->nvariables) * sizeof (unsigned int));
00987               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00988               if (error_code || !i)
00989                 {
00990                   snprintf (buffer2, 64, "%s %u: %s",
00991                             gettext ("Variable"),
00992                             input->nvariables + 1,
00993                             gettext ("invalid bits number"));
00994                   msg = buffer2;
00995                   goto exit_on_error;
00996                 }
00997               input->nbits[input->nvariables] = i;
00998             }
00999           else
01000             {
01001               snprintf (buffer2, 64, "%s %u: %s",
01002                         gettext ("Variable"),
01003                         input->nvariables + 1, gettext ("no bits number"));
01004               msg = buffer2;
01005               goto exit_on_error;
01006             }
01007         }
01008       ++input->nvariables;
01009     }
01010   if (!input->nvariables)
01011     {
01012       msg = gettext ("No calibration variables");
01013       goto exit_on_error;
01014     }
01015
01016   // Getting the working directory
01017   input->directory = g_path_get_dirname (filename);
01018   input->name = g_path_get_basename (filename);
01019
01020   // Closing the XML document
01021   xmlFreeDoc (doc);
01022
01023 #if DEBUG
01024   fprintf (stderr, "input_open: end\n");
01025 #endif
01026   return 1;
01027
01028 exit_on_error:
01029   show_error (msg);
01030   input_free ();
01031 #if DEBUG
01032   fprintf (stderr, "input_open: end\n");
01033 #endif
01034   return 0;
01035 }
```

Here is the call graph for this function:



5.1.2.11 void input_save (char * *filename*)

Function to save the input file.

Parameters

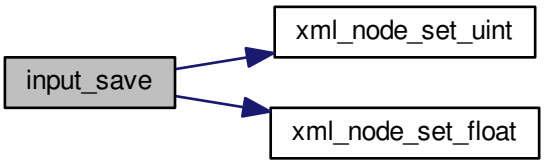
filename	Input file name.

Definition at line 2183 of file calibrator.c.

```
02184 {
02185   unsigned int i, j;
02186   char *buffer;
02187   xmlDoc *doc;
02188   xmlNode *node, *child;
02189   GFile *file, *file2;
02190
02191   // Getting the input file directory
02192   input->name = g_path_get_basename (filename);
02193   input->directory = g_path_get_dirname (filename);
02194   file = g_file_new_for_path (input->directory);
02195
02196   // Opening the input file
02197   doc = xmlNewDoc ((const xmlChar *) "1.0");
02198
02199   // Setting root XML node
02200   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02201   xmlDocSetRootElement (doc, node);
02202
02203   // Adding properties to the root XML node
02204   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02205     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02206   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02207     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02208   file2 = g_file_new_for_path (input->simulator);
02209   buffer = g_file_get_relative_path (file, file2);
02210   g_object_unref (file2);
02211   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02212   g_free (buffer);
02213   if (input->evaluator)
02214     {
02215       file2 = g_file_new_for_path (input->evaluator);
02216       buffer = g_file_get_relative_path (file, file2);
02217       g_object_unref (file2);
02218       if (xmlStrlen ((xmlChar *) buffer))
02219         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02220       g_free (buffer);
02221     }
02222   if (input->seed != DEFAULT_RANDOM_SEED)
02223     xml_node_set_uint (node, XML_SEED, input->seed);
02224
02225   // Setting the algorithm
02226   buffer = (char *) g_malloc (64);
02227   switch (input->algorithm)
02228     {
02229     case ALGORITHM_MONTE_CARLO:
02230       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02231       snprintf (buffer, 64, "%u", input->nsimulations);
02232       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02233       snprintf (buffer, 64, "%u", input->niterations);
02234       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02235       snprintf (buffer, 64, "%.3lg", input->tolerance);
02236       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02237       snprintf (buffer, 64, "%u", input->nbest);
02238       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02239       break;
02240     case ALGORITHM_SWEEP:
02241       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02242       snprintf (buffer, 64, "%u", input->niterations);
02243       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02244       snprintf (buffer, 64, "%.3lg", input->tolerance);
02245       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02246       snprintf (buffer, 64, "%u", input->nbest);
02247       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02248       break;
02249     default:
02250       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02251       snprintf (buffer, 64, "%u", input->nsimulations);
02252       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02253       snprintf (buffer, 64, "%u", input->niterations);
02254       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02255       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02256       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02257       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02258       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02259       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02260       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02261       break;
```

```
02262     }
02263   g_free (buffer);
02264
02265   // Setting the experimental data
02266   for (i = 0; i < input->nexperiments; ++i)
02267     {
02268       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02269       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02270       if (input->weight[i] != 1.)
02271         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02272       for (j = 0; j < input->ninputs; ++j)
02273         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02274     }
02275
02276   // Setting the variables data
02277   for (i = 0; i < input->nvariables; ++i)
02278     {
02279       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02280       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02281       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02282       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02283         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02284       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02285       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02286         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02287       if (input->precision[i] != DEFAULT_PRECISION)
02288         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02289       if (input->algorithm == ALGORITHM_SWEEP)
02290         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02291       else if (input->algorithm == ALGORITHM_GENETIC)
02292         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02293     }
02294
02295   // Saving the XML file
02296   xmlSaveFormatFile (filename, doc, 1);
02297
02298   // Freeing memory
02299   xmlFreeDoc (doc);
02300 }
```

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

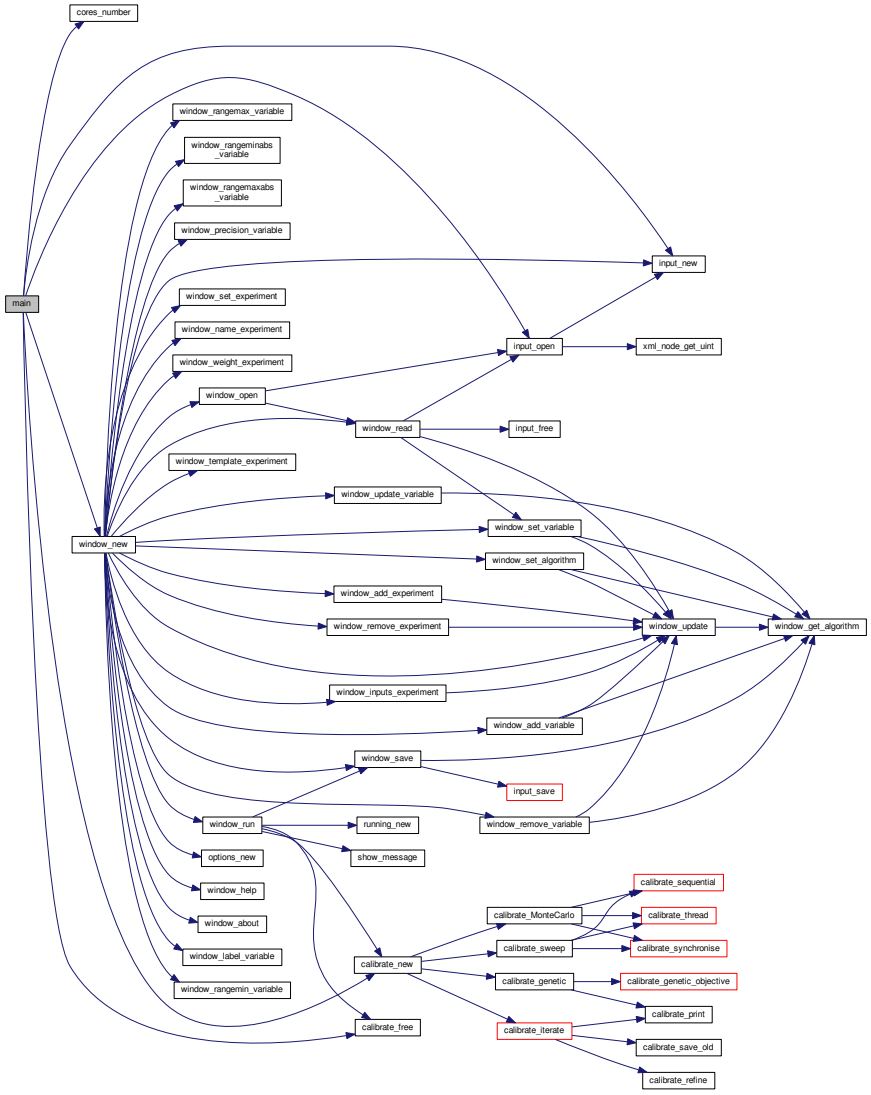
0 on success, >0 on error.

Definition at line 4067 of file calibrator.c.

```
04068 {
04069   // Starting pseudo-random numbers generator
04070   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04071   calibrate->seed = DEFAULT_RANDOM_SEED;
04072
04073   // Allowing spaces in the XML data file
04074   xmlKeepBlanksDefault (0);
04075
04076   // Starting MPI
04077 #if HAVE_MPI
04078   MPI_Init (&argn, &argc);
04079   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04080   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04081   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04082 #else
04083   ntasks = 1;
04084 #endif
04085
04086 #if HAVE_GTK
04087
04088   // Getting threads number
04089   nthreads = cores_number ();
04090
04091   // Setting local language and international floating point numbers notation
04092   setlocale (LC_ALL, "");
04093   setlocale (LC_NUMERIC, "C");
04094   window->application_directory = g_get_current_dir ();
04095   bindtextdomain (PROGRAM_INTERFACE,
04096                   g_build_filename (window->application_directory,
04097                                     LOCALE_DIR, NULL));
04098   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04099   textdomain (PROGRAM_INTERFACE);
04100
04101   // Initing GTK+
04102   gtk_disable_setlocale ();
04103   gtk_init (&argn, &argc);
04104
04105   // Opening the main window
04106   window_new ();
04107   gtk_main ();
04108
04109   // Freeing memory
04110   gtk_widget_destroy (GTK_WIDGET (window->window));
04111   g_free (window->application_directory);
04112
04113 #else
04114
04115   // Checking syntax
04116   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04117     {
04118       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
04119       return 1;
04120     }
04121
04122   // Getting threads number
04123   if (argn == 2)
04124     nthreads = cores_number ();
04125   else
04126     nthreads = atoi (argc[2]);
04127   printf ("nthreads=%u\n", nthreads);
04128
04129   // Making calibration
04130   input_new ();
04131   if (input_open (argc[argn - 1]))
04132     calibrate_new ();
04133
04134   // Freeing memory
04135   calibrate_free ();
04136
04137 #endif
04138
```

```
04139   // Closing MPI
04140 #if HAVE_MPI
04141   MPI_Finalize ();
04142 #endif
04143
04144   // Freeing memory
04145   gsl_rng_free (calibrate->rng);
04146
04147   // Closing
04148   return 0;
04149 }
```

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.

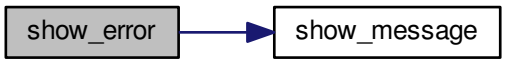
Parameters

msg	Error message.

Definition at line 251 of file calibrator.c.

```
00252 {
00253   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00254 }
```

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 221 of file calibrator.c.

```
00222 {
00223 #if HAVE_GTK
00224   GtkMessageDialog *dlg;
00225
00226   // Creating the dialog
00227   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00228     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00229
00230   // Setting the dialog title
00231   gtk_window_set_title (GTK_WINDOW (dlg), title);
00232
00233   // Showing the dialog and waiting response
00234   gtk_dialog_run (GTK_DIALOG (dlg));
00235
00236   // Closing and freeing memory
00237   gtk_widget_destroy (GTK_WIDGET (dlg));
00238
00239 #else
00240   printf ("%s: %s\n", title, msg);
00241 #endif
00242 }
```

5.1.2.15 int window_get_algorithm ( )

Function to get the algorithm number.

Returns

Algorithm number.

Definition at line 2573 of file calibrator.c.

```
02574 {
02575   unsigned int i;
02576   for (i = 0; i < NALGORITHMS; ++i)
02577     if (gtk_toggle_button_get_active
02578         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02579       break;
02580   return i;
02581 }
```

5.1.2.16 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

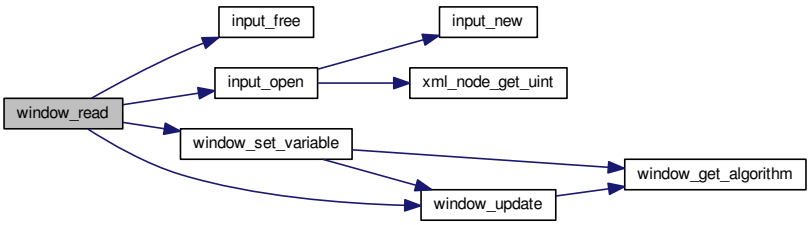
1 on succes, 0 on error.

Definition at line 3324 of file calibrator.c.

```
03325 {
03326   unsigned int i;
03327   char *buffer;
03328 #if DEBUG
03329   fprintf (stderr, "window_read: start\n");
03330 #endif
03331
03332   // Reading new input file
03333   input_free ();
03334   if (!input_open (filename))
03335     return 0;
03336
03337   // Setting GTK+ widgets data
03338   gtk_entry_set_text (window->entry_result, input->result);
03339   gtk_entry_set_text (window->entry_variables, input->
      variables);
03340   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03341   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03342                                  (window->button_simulator), buffer);
03343   g_free (buffer);
03344   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03345                                 (size_t) input->evaluator);
03346   if (input->evaluator)
03347     {
03348       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03349       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03350                                      (window->button_evaluator), buffer);
03351       g_free (buffer);
03352     }
03353   gtk_toggle_button_set_active
03354     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03355   switch (input->algorithm)
03356     {
03357     case ALGORITHM_MONTE_CARLO:
03358       gtk_spin_button_set_value (window->spin_simulations,
03359                                  (gdouble) input->nsimulations);
03360     case ALGORITHM_SWEEP:
03361       gtk_spin_button_set_value (window->spin_iterations,
03362                                  (gdouble) input->niterations);
03363       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03364       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03365       break;
```

```
03366     default:
03367       gtk_spin_button_set_value (window->spin_population,
03368                                  (gdouble) input->nsimulations);
03369       gtk_spin_button_set_value (window->spin_generations,
03370                                  (gdouble) input->niterations);
03371       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03372       gtk_spin_button_set_value (window->spin_reproduction,
03373                                  input->reproduction_ratio);
03374       gtk_spin_button_set_value (window->spin_adaptation,
03375                                  input->adaptation_ratio);
03376     }
03377   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03378   g_signal_handler_block (window->button_experiment,
03379                           window->id_experiment_name);
03380   gtk_combo_box_text_remove_all (window->combo_experiment);
03381   for (i = 0; i < input->nexperiments; ++i)
03382     gtk_combo_box_text_append_text (window->combo_experiment,
03383                                     input->experiment[i]);
03384   g_signal_handler_unblock
03385     (window->button_experiment, window->
      id_experiment_name);
03386   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03387   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03388   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03389   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03390   gtk_combo_box_text_remove_all (window->combo_variable);
03391   for (i = 0; i < input->nvariables; ++i)
03392     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03393   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03394   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03395   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03396   window_set_variable ();
03397   window_update ();
03398
03399 #if DEBUG
03400   fprintf (stderr, "window_read: end\n");
03401 #endif
03402   return 1;
03403 }
```

Here is the call graph for this function:



5.1.2.17 int window_save ( )

Function to save the input file.

Returns

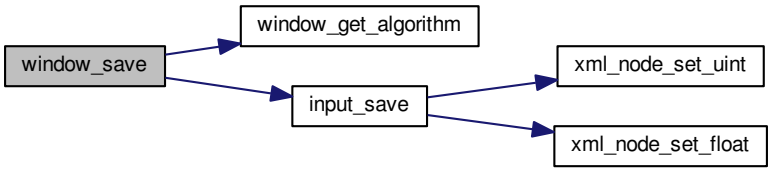
1 on OK, 0 on Cancel.

Definition at line 2381 of file calibrator.c.

```
02382 {
02383   char *buffer;
02384   GtkFileChooserDialog *dlg;
02385
02386 #if DEBUG
02387   fprintf (stderr, "window_save: start\n");
02388 #endif
02389
02390   // Opening the saving dialog
02391   dlg = (GtkFileChooserDialog *)
02392     gtk_file_chooser_dialog_new (gettext ("Save file"),
02393                                  window->window,
02394                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02395                                  gettext ("_Cancel"),
02396                                  GTK_RESPONSE_CANCEL,
02397                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02398   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02399   buffer = g_build_filename (input->directory, input->name, NULL);
02400   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02401   g_free (buffer);
02402
02403   // If OK response then saving
02404   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02405     {
02406
02407       // Adding properties to the root XML node
02408       input->simulator = gtk_file_chooser_get_filename
02409         (GTK_FILE_CHOOSER (window->button_simulator));
02410       if (gtk_toggle_button_get_active
02411           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02412         input->evaluator = gtk_file_chooser_get_filename
02413           (GTK_FILE_CHOOSER (window->button_evaluator));
02414       else
02415         input->evaluator = NULL;
02416       input->result
02417         = (char *) xmlStrdup ((const xmlChar *)
02418                               gtk_entry_get_text (window->entry_result));
02419       input->variables
02420         = (char *) xmlStrdup ((const xmlChar *)
02421                               gtk_entry_get_text (window->entry_variables));
02422
02423       // Setting the algorithm
02424       switch (window_get_algorithm ())
02425         {
02426         case ALGORITHM_MONTE_CARLO:
02427           input->algorithm = ALGORITHM_MONTE_CARLO;
02428           input->nsimulations
02429             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02430           input->niterations
02431             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02432           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02433           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02434           break;
02435         case ALGORITHM_SWEEP:
02436           input->algorithm = ALGORITHM_SWEEP;
02437           input->niterations
02438             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02439           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
02440           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
02441           break;
02442         default:
02443           input->algorithm = ALGORITHM_GENETIC;
02444           input->nsimulations
02445             = gtk_spin_button_get_value_as_int (window->spin_population);
02446           input->niterations
02447             = gtk_spin_button_get_value_as_int (window->spin_generations);
02448           input->mutation_ratio
02449             = gtk_spin_button_get_value (window->spin_mutation);
02450           input->reproduction_ratio
02451             = gtk_spin_button_get_value (window->spin_reproduction);
02452           input->adaptation_ratio
02453             = gtk_spin_button_get_value (window->spin_adaptation);
02454           break;
02455         }
02456
02457       // Saving the XML file
02458       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02459       input_save (buffer);
02460
02461       // Closing and freeing memory
02462       g_free (buffer);
02463       gtk_widget_destroy (GTK_WIDGET (dlg));
02464 #if DEBUG
```

```
02465       fprintf (stderr, "window_save: end\n");
02466 #endif
02467       return 1;
02468     }
02469
02470   // Closing and freeing memory
02471   gtk_widget_destroy (GTK_WIDGET (dlg));
02472 #if DEBUG
02473   fprintf (stderr, "window_save: end\n");
02474 #endif
02475   return 0;
02476 }
```

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 2960 of file calibrator.c.

```
02961 {
02962   unsigned int i, j;
02963   char *buffer;
02964   GFile *file1, *file2;
02965 #if DEBUG
02966   fprintf (stderr, "window_template_experiment: start\n");
02967 #endif
02968   i = (size_t) data;
02969   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02970   file1
02971     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02972   file2 = g_file_new_for_path (input->directory);
02973   buffer = g_file_get_relative_path (file2, file1);
02974   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02975   g_free (buffer);
02976   g_object_unref (file2);
02977   g_object_unref (file1);
02978 #if DEBUG
02979   fprintf (stderr, "window_template_experiment: end\n");
02980 #endif
02981 }
```

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 330 of file calibrator.c.

```
00331 {
00332   double x = 0.;
00333   xmlChar *buffer;
00334   buffer = xmlGetProp (node, prop);
00335   if (!buffer)
00336     *error_code = 1;
00337   else
00338     {
00339       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343       xmlFree (buffer);
00344     }
00345   return x;
00346 }
```

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 268 of file calibrator.c.

```
00269 {
00270   int i = 0;
00271   xmlChar *buffer;
00272   buffer = xmlGetProp (node, prop);
00273   if (!buffer)
00274     *error_code = 1;
00275   else
00276     {
00277       if (sscanf ((char *) buffer, "%d", &i) != 1)
00278         *error_code = 2;
00279       else
00280         *error_code = 0;
00281       xmlFree (buffer);
00282     }
00283   return i;
00284 }
```

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 299 of file calibrator.c.

```
00300 {
00301   unsigned int i = 0;
00302   xmlChar *buffer;
00303   buffer = xmlGetProp (node, prop);
00304   if (!buffer)
00305     *error_code = 1;
00306   else
00307     {
00308       if (sscanf ((char *) buffer, "%u", &i) != 1)
00309         *error_code = 2;
00310       else
00311         *error_code = 0;
00312       xmlFree (buffer);
00313     }
00314   return i;
00315 }
```

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 397 of file calibrator.c.

```
00398 {
00399   xmlChar buffer[64];
00400   snprintf ((char *) buffer, 64, "%.14lg", value);
00401   xmlSetProp (node, prop, buffer);
00402 }
```

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 359 of file calibrator.c.

```
00360 {
00361   xmlChar buffer[64];
00362   snprintf ((char *) buffer, 64, "%d", value);
00363   xmlSetProp (node, prop, buffer);
00364 }
```

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 378 of file calibrator.c.

```
00379 {
00380   xmlChar buffer[64];
00381   snprintf ((char *) buffer, 64, "%u", value);
00382   xmlSetProp (node, prop, buffer);
00383 }
```

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 112 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 117 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 105 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 *result_name = (xmlChar *) "result";
00102 const xmlChar *variables_name = (xmlChar *) "variables";
00104
00105 const xmlChar *template[MAX_NINPUTS] = {
00106   XML_TEMPLATE1, XML_TEMPLATE2, XML_TEMPLATE3,
      XML_TEMPLATE4,
00107   XML_TEMPLATE5, XML_TEMPLATE6, XML_TEMPLATE7,
      XML_TEMPLATE8
```

```
00108 };
00109
00111
00112 const char *format[NPRECISIONS] = {
00113   "%.11g", "%.2lg", "%.3lg", "%.4lg", "%.5lg", "%.6lg", "%.7lg", "%.8lg",
00114   "%.9lg", "%.10lg", "%.11lg", "%.12lg", "%.13lg", "%.14lg", "%.15lg"
00115 };
00116
00117 const double precision[NPRECISIONS] = {
00118   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,
00119   1e-13, 1e-14
00120 };
00121
00122 const char *logo[] = {
00123   "32 32 3 1",
00124   "     c None",
00125   ".    c #0000FF",
00126   "+    c #FF0000",
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   "                                ",
00158   "                                "
00159 };
00160
00161 /*
00162 const char * logo[] = {
00163 "32 32 3 1",
00164 "     c #FFFFFFFFFFFF",
00165 ".    c #00000000FFFF",
00166 "X    c #FFFF00000000",
00167 "                                ",
00168 "                                ",
00169 "                                ",
00170 "     .      .      .      .     ",
00171 "     .      .      .      .     ",
00172 "     .      .      .      .     ",
00173 "     .      .      .      .     ",
00174 "     .      .     XXX     .     ",
00175 "     .      .    XXXXX    .     ",
00176 "     .      .    XXXXX    .     ",
00177 "     .      .    XXXXX    .     ",
00178 "    XXX     .     XXX    XXX    ",
00179 "   XXXXX    .      .    XXXXX   ",
00180 "   XXXXX    .      .    XXXXX   ",
00181 "   XXXXX    .      .    XXXXX   ",
00182 "    XXX     .      .     XXX    ",
00183 "     .      .      .      .     ",
00184 "     .     XXX     .      .     ",
00185 "     .    XXXXX    .      .     ",
00186 "     .    XXXXX    .      .     ",
00187 "     .    XXXXX    .      .     ",
00188 "     .     XXX     .      .     ",
00189 "     .      .      .      .     ",
00190 "     .      .      .      .     ",
00191 "     .      .      .      .     ",
00192 "     .      .      .      .     ",
00193 "     .      .      .      .     ",
00194 "     .      .      .      .     ",
00195 "     .      .      .      .     ",
```

```
00196 "                                 ",
00197 "                                 ",
00198 "                                 "};
00199 */
00200
00201 #if HAVE_GTK
00202 Options options[1];
00204 Running running[1];
00206 Window window[1];
00208 #endif
00209
00220 void
00221 show_message (char *title, char *msg, int type)
00222 {
00223 #if HAVE_GTK
00224   GtkMessageDialog *dlg;
00225
00226   // Creating the dialog
00227   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00228     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00229
00230   // Setting the dialog title
00231   gtk_window_set_title (GTK_WINDOW (dlg), title);
00232
00233   // Showing the dialog and waiting response
00234   gtk_dialog_run (GTK_DIALOG (dlg));
00235
00236   // Closing and freeing memory
00237   gtk_widget_destroy (GTK_WIDGET (dlg));
00238
00239 #else
00240   printf ("%s: %s\n", title, msg);
00241 #endif
00242 }
00243
00250 void
00251 show_error (char *msg)
00252 {
00253   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00254 }
00255
00267 int
00268 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00269 {
00270   int i = 0;
00271   xmlChar *buffer;
00272   buffer = xmlGetProp (node, prop);
00273   if (!buffer)
00274     *error_code = 1;
00275   else
00276     {
00277       if (sscanf ((char *) buffer, "%d", &i) != 1)
00278         *error_code = 2;
00279       else
00280         *error_code = 0;
00281       xmlFree (buffer);
00282     }
00283   return i;
00284 }
00285
00298 unsigned int
00299 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00300 {
00301   unsigned int i = 0;
00302   xmlChar *buffer;
00303   buffer = xmlGetProp (node, prop);
00304   if (!buffer)
00305     *error_code = 1;
00306   else
00307     {
00308       if (sscanf ((char *) buffer, "%u", &i) != 1)
00309         *error_code = 2;
00310       else
00311         *error_code = 0;
00312       xmlFree (buffer);
00313     }
00314   return i;
00315 }
00316
00329 double
00330 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00331 {
00332   double x = 0.;
00333   xmlChar *buffer;
00334   buffer = xmlGetProp (node, prop);
00335   if (!buffer)
00336     *error_code = 1;
```

```
00337   else
00338     {
00339       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343       xmlFree (buffer);
00344     }
00345   return x;
00346 }
00347
00358 void
00359 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00360 {
00361   xmlChar buffer[64];
00362   snprintf ((char *) buffer, 64, "%d", value);
00363   xmlSetProp (node, prop, buffer);
00364 }
00365
00377 void
00378 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00379 {
00380   xmlChar buffer[64];
00381   snprintf ((char *) buffer, 64, "%u", value);
00382   xmlSetProp (node, prop, buffer);
00383 }
00384
00396 void
00397 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00398 {
00399   xmlChar buffer[64];
00400   snprintf ((char *) buffer, 64, "%.14lg", value);
00401   xmlSetProp (node, prop, buffer);
00402 }
00403
00408 void
00409 input_new ()
00410 {
00411   unsigned int i;
00412 #if DEBUG
00413   fprintf (stderr, "input_init: start\n");
00414 #endif
00415   input->nvariables = input->nexperiments = input->ninputs = 0;
00416   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00417   input->experiment = input->label = NULL;
00418   input->precision = input->nsweeps = input->nbits = NULL;
00419   input->rangemin = input->rangemax = input->rangeminabs = input->
      rangemaxabs
00420     = input->weight = NULL;
00421   for (i = 0; i < MAX_NINPUTS; ++i)
00422     input->template[i] = NULL;
00423 #if DEBUG
00424   fprintf (stderr, "input_init: end\n");
00425 #endif
00426 }
00427
00432 void
00433 input_free ()
00434 {
00435   unsigned int i, j;
00436 #if DEBUG
00437   fprintf (stderr, "input_free: start\n");
00438 #endif
00439   g_free (input->name);
00440   g_free (input->directory);
00441   for (i = 0; i < input->nexperiments; ++i)
00442     {
00443       xmlFree (input->experiment[i]);
00444       for (j = 0; j < input->ninputs; ++j)
00445         xmlFree (input->template[j][i]);
00446     }
00447   g_free (input->experiment);
00448   for (i = 0; i < input->ninputs; ++i)
00449     g_free (input->template[i]);
00450   for (i = 0; i < input->nvariables; ++i)
00451     xmlFree (input->label[i]);
00452   g_free (input->label);
00453   g_free (input->precision);
00454   g_free (input->rangemin);
00455   g_free (input->rangemax);
00456   g_free (input->rangeminabs);
00457   g_free (input->rangemaxabs);
00458   g_free (input->weight);
00459   g_free (input->nsweeps);
00460   g_free (input->nbits);
00461   xmlFree (input->evaluator);
```

```
00462   xmlFree (input->simulator);
00463   xmlFree (input->result);
00464   xmlFree (input->variables);
00465   input->nexperiments = input->ninputs = input->nvariables = 0;
00466 #if DEBUG
00467   fprintf (stderr, "input_free: end\n");
00468 #endif
00469 }
00470
00478 int
00479 input_open (char *filename)
00480 {
00481   char buffer2[64];
00482   xmlDoc *doc;
00483   xmlNode *node, *child;
00484   xmlChar *buffer;
00485   char *msg;
00486   int error_code;
00487   unsigned int i;
00488
00489 #if DEBUG
00490   fprintf (stderr, "input_open: start\n");
00491 #endif
00492
00493   // Resetting input data
00494   input_new ();
00495
00496   // Parsing the input file
00497   doc = xmlParseFile (filename);
00498   if (!doc)
00499     {
00500       msg = gettext ("Unable to parse the input file");
00501       goto exit_on_error;
00502     }
00503
00504   // Getting the root node
00505   node = xmlDocGetRootElement (doc);
00506   if (xmlStrcmp (node->name, XML_CALIBRATE))
00507     {
00508       msg = gettext ("Bad root XML node");
00509       goto exit_on_error;
00510     }
00511
00512   // Getting results file names
00513   input->result = (char *) xmlGetProp (node, XML_RESULT);
00514   if (!input->result)
00515     input->result = (char *) xmlStrdup (result_name);
00516   input->variables = (char *) xmlGetProp (node, XML_VARIABLES);
00517   if (!input->variables)
00518     input->variables = (char *) xmlStrdup (variables_name);
00519
00520   // Opening simulator program name
00521   input->simulator = (char *) xmlGetProp (node, XML_SIMULATOR);
00522   if (!input->simulator)
00523     {
00524       msg = gettext ("Bad simulator program");
00525       goto exit_on_error;
00526     }
00527
00528   // Opening evaluator program name
00529   input->evaluator = (char *) xmlGetProp (node, XML_EVALUATOR);
00530
00531   // Obtaining pseudo-random numbers generator seed
00532   if (!xmlHasProp (node, XML_SEED))
00533     input->seed = DEFAULT_RANDOM_SEED;
00534   else
00535     {
00536       input->seed = xml_node_get_uint (node, XML_SEED, &error_code);
00537       if (error_code)
00538         {
00539           msg = gettext ("Bad pseudo-random numbers generator seed");
00540           goto exit_on_error;
00541         }
00542     }
00543
00544   // Opening algorithm
00545   buffer = xmlGetProp (node, XML_ALGORITHM);
00546   if (!xmlStrcmp (buffer, XML_MONTE_CARLO))
00547     {
00548       input->algorithm = ALGORITHM_MONTE_CARLO;
00549
00550       // Obtaining simulations number
00551       input->nsimulations
00552         = xml_node_get_int (node, XML_NSIMULATIONS, &error_code);
00553       if (error_code)
00554         {
00555           msg = gettext ("Bad simulations number");
```

```
00556           goto exit_on_error;
00557         }
00558     }
00559   else if (!xmlStrcmp (buffer, XML_SWEEP))
00560     input->algorithm = ALGORITHM_SWEEP;
00561   else if (!xmlStrcmp (buffer, XML_GENETIC))
00562     {
00563       input->algorithm = ALGORITHM_GENETIC;
00564
00565       // Obtaining population
00566       if (xmlHasProp (node, XML_NPOPULATION))
00567         {
00568           input->nsimulations
00569             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00570           if (error_code || input->nsimulations < 3)
00571             {
00572               msg = gettext ("Invalid population number");
00573               goto exit_on_error;
00574             }
00575         }
00576       else
00577         {
00578           msg = gettext ("No population number");
00579           goto exit_on_error;
00580         }
00581
00582       // Obtaining generations
00583       if (xmlHasProp (node, XML_NGENERATIONS))
00584         {
00585           input->niterations
00586             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00587           if (error_code || !input->niterations)
00588             {
00589               msg = gettext ("Invalid generations number");
00590               goto exit_on_error;
00591             }
00592         }
00593       else
00594         {
00595           msg = gettext ("No generations number");
00596           goto exit_on_error;
00597         }
00598
00599       // Obtaining mutation probability
00600       if (xmlHasProp (node, XML_MUTATION))
00601         {
00602           input->mutation_ratio
00603             = xml_node_get_float (node, XML_MUTATION, &error_code);
00604           if (error_code || input->mutation_ratio < 0.
00605               || input->mutation_ratio >= 1.)
00606             {
00607               msg = gettext ("Invalid mutation probability");
00608               goto exit_on_error;
00609             }
00610         }
00611       else
00612         {
00613           msg = gettext ("No mutation probability");
00614           goto exit_on_error;
00615         }
00616
00617       // Obtaining reproduction probability
00618       if (xmlHasProp (node, XML_REPRODUCTION))
00619         {
00620           input->reproduction_ratio
00621             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00622           if (error_code || input->reproduction_ratio < 0.
00623               || input->reproduction_ratio >= 1.0)
00624             {
00625               msg = gettext ("Invalid reproduction probability");
00626               goto exit_on_error;
00627             }
00628         }
00629       else
00630         {
00631           msg = gettext ("No reproduction probability");
00632           goto exit_on_error;
00633         }
00634
00635       // Obtaining adaptation probability
00636       if (xmlHasProp (node, XML_ADAPTATION))
00637         {
00638           input->adaptation_ratio
00639             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00640           if (error_code || input->adaptation_ratio < 0.
00641               || input->adaptation_ratio >= 1.)
00642             {
```

```
00643               msg = gettext ("Invalid adaptation probability");
00644               goto exit_on_error;
00645             }
00646         }
00647       else
00648         {
00649           msg = gettext ("No adaptation probability");
00650           goto exit_on_error;
00651         }
00652
00653       // Checking survivals
00654       i = input->mutation_ratio * input->nsimulations;
00655       i += input->reproduction_ratio * input->nsimulations;
00656       i += input->adaptation_ratio * input->nsimulations;
00657       if (i > input->nsimulations - 2)
00658         {
00659           msg = gettext
00660             ("No enough survival entities to reproduce the population");
00661           goto exit_on_error;
00662         }
00663     }
00664   else
00665     {
00666       msg = gettext ("Unknown algorithm");
00667       goto exit_on_error;
00668     }
00669
00670   if (input->algorithm == ALGORITHM_MONTE_CARLO
00671       || input->algorithm == ALGORITHM_SWEEP)
00672     {
00673
00674       // Obtaining iterations number
00675       input->niterations
00676         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00677       if (error_code == 1)
00678         input->niterations = 1;
00679       else if (error_code)
00680         {
00681           msg = gettext ("Bad iterations number");
00682           goto exit_on_error;
00683         }
00684
00685       // Obtaining best number
00686       if (xmlHasProp (node, XML_NBEST))
00687         {
00688           input->nbest = xml_node_get_uint (node,
    XML_NBEST, &error_code);
00689           if (error_code || !input->nbest)
00690             {
00691               msg = gettext ("Invalid best number");
00692               goto exit_on_error;
00693             }
00694         }
00695       else
00696         input->nbest = 1;
00697
00698       // Obtaining tolerance
00699       if (xmlHasProp (node, XML_TOLERANCE))
00700         {
00701           input->tolerance
00702             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00703           if (error_code || input->tolerance < 0.)
00704             {
00705               msg = gettext ("Invalid tolerance");
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         input->tolerance = 0.;
00711     }
00712
00713   // Reading the experimental data
00714   for (child = node->children; child; child = child->next)
00715     {
00716       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00717         break;
00718 #if DEBUG
00719       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00720 #endif
00721       if (xmlHasProp (child, XML_NAME))
00722         {
00723           input->experiment
00724             = g_realloc (input->experiment,
00725                          (1 + input->nexperiments) * sizeof (char *));
00726           input->experiment[input->nexperiments]
00727             = (char *) xmlGetProp (child, XML_NAME);
00728         }
```

```
00729       else
00730         {
00731           snprintf (buffer2, 64, "%s %u: %s",
00732                     gettext ("Experiment"),
00733                     input->nexperiments + 1, gettext ("no data file name"));
00734           msg = buffer2;
00735           goto exit_on_error;
00736         }
00737 #if DEBUG
00738       fprintf (stderr, "input_open: experiment=%s\n",
00739                input->experiment[input->nexperiments]);
00740 #endif
00741       input->weight = g_realloc (input->weight,
00742                                  (1 + input->nexperiments) * sizeof (double));
00743       if (xmlHasProp (child, XML_WEIGHT))
00744         {
00745           input->weight[input->nexperiments]
00746             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00747           if (error_code)
00748             {
00749               snprintf (buffer2, 64, "%s %u: %s",
00750                         gettext ("Experiment"),
00751                         input->nexperiments + 1, gettext ("bad weight"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755         }
00756       else
00757         input->weight[input->nexperiments] = 1.;
00758 #if DEBUG
00759       fprintf (stderr, "input_open: weight=%lg\n",
00760                input->weight[input->nexperiments]);
00761 #endif
00762       if (!input->nexperiments)
00763         input->ninputs = 0;
00764 #if DEBUG
00765       fprintf (stderr, "input_open: template[0]\n");
00766 #endif
00767       if (xmlHasProp (child, XML_TEMPLATE1))
00768         {
00769           input->template[0]
00770             = (char **) g_realloc (input->template[0],
00771                                    (1 + input->nexperiments) * sizeof (char *));
00772           input->template[0][input->nexperiments]
00773             = (char *) xmlGetProp (child, template[0]);
00774 #if DEBUG
00775           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00776                    input->nexperiments,
00777                    input->template[0][input->nexperiments]);
00778 #endif
00779           if (!input->nexperiments)
00780             ++input->ninputs;
00781 #if DEBUG
00782           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00783 #endif
00784         }
00785       else
00786         {
00787           snprintf (buffer2, 64, "%s %u: %s",
00788                     gettext ("Experiment"),
00789                     input->nexperiments + 1, gettext ("no template"));
00790           msg = buffer2;
00791           goto exit_on_error;
00792         }
00793       for (i = 1; i < MAX_NINPUTS; ++i)
00794         {
00795 #if DEBUG
00796           fprintf (stderr, "input_open: template%u\n", i + 1);
00797 #endif
00798           if (xmlHasProp (child, template[i]))
00799             {
00800               if (input->nexperiments && input->ninputs <= i)
00801                 {
00802                   snprintf (buffer2, 64, "%s %u: %s",
00803                             gettext ("Experiment"),
00804                             input->nexperiments + 1,
00805                             gettext ("bad templates number"));
00806                   msg = buffer2;
00807                   goto exit_on_error;
00808                 }
00809               input->template[i] = (char **)
00810                 g_realloc (input->template[i],
00811                            (1 + input->nexperiments) * sizeof (char *));
00812               input->template[i][input->nexperiments]
00813                 = (char *) xmlGetProp (child, template[i]);
00814 #if DEBUG
00815               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
```

```
00816                         input->nexperiments, i + 1,
00817                         input->template[i][input->nexperiments]);
00818 #endif
00819               if (!input->nexperiments)
00820                 ++input->ninputs;
00821 #if DEBUG
00822               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00823 #endif
00824             }
00825           else if (input->nexperiments && input->ninputs >= i)
00826             {
00827               snprintf (buffer2, 64, "%s %u: %s%u",
00828                         gettext ("Experiment"),
00829                         input->nexperiments + 1,
00830                         gettext ("no template"), i + 1);
00831               msg = buffer2;
00832               goto exit_on_error;
00833             }
00834           else
00835             break;
00836         }
00837       ++input->nexperiments;
00838 #if DEBUG
00839       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00840 #endif
00841     }
00842   if (!input->nexperiments)
00843     {
00844       msg = gettext ("No calibration experiments");
00845       goto exit_on_error;
00846     }
00847
00848   // Reading the variables data
00849   for (; child; child = child->next)
00850     {
00851       if (xmlStrcmp (child->name, XML_VARIABLE))
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("bad XML node"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_NAME))
00860         {
00861           input->label = g_realloc
00862             (input->label, (1 + input->nvariables) * sizeof (char *));
00863           input->label[input->nvariables]
00864             = (char *) xmlGetProp (child, XML_NAME);
00865         }
00866       else
00867         {
00868           snprintf (buffer2, 64, "%s %u: %s",
00869                     gettext ("Variable"),
00870                     input->nvariables + 1, gettext ("no name"));
00871           msg = buffer2;
00872           goto exit_on_error;
00873         }
00874       if (xmlHasProp (child, XML_MINIMUM))
00875         {
00876           input->rangemin = g_realloc
00877             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00878           input->rangeminabs = g_realloc
00879             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00880           input->rangemin[input->nvariables]
00881             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00882           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00883             {
00884               input->rangeminabs[input->nvariables]
00885                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00886             }
00887           else
00888             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00889           if (input->rangemin[input->nvariables]
00890               < input->rangeminabs[input->nvariables])
00891             {
00892               snprintf (buffer2, 64, "%s %u: %s",
00893                         gettext ("Variable"),
00894                         input->nvariables + 1,
00895                         gettext ("minimum range not allowed"));
00896               msg = buffer2;
00897               goto exit_on_error;
00898             }
00899         }
00900       else
00901         {
```

```
00902           snprintf (buffer2, 64, "%s %u: %s",
00903                     gettext ("Variable"),
00904                     input->nvariables + 1, gettext ("no minimum range"));
00905           msg = buffer2;
00906           goto exit_on_error;
00907         }
00908       if (xmlHasProp (child, XML_MAXIMUM))
00909         {
00910           input->rangemax = g_realloc
00911             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00912           input->rangemaxabs = g_realloc
00913             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00914           input->rangemax[input->nvariables]
00915             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00916           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00917             input->rangemaxabs[input->nvariables]
00918               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00919           else
00920             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00921           if (input->rangemax[input->nvariables]
00922               > input->rangemaxabs[input->nvariables])
00923             {
00924               snprintf (buffer2, 64, "%s %u: %s",
00925                         gettext ("Variable"),
00926                         input->nvariables + 1,
00927                         gettext ("maximum range not allowed"));
00928               msg = buffer2;
00929               goto exit_on_error;
00930             }
00931         }
00932       else
00933         {
00934           snprintf (buffer2, 64, "%s %u: %s",
00935                     gettext ("Variable"),
00936                     input->nvariables + 1, gettext ("no maximum range"));
00937           msg = buffer2;
00938           goto exit_on_error;
00939         }
00940       if (input->rangemax[input->nvariables]
00941           < input->rangemin[input->nvariables])
00942         {
00943           snprintf (buffer2, 64, "%s %u: %s",
00944                     gettext ("Variable"),
00945                     input->nvariables + 1, gettext ("bad range"));
00946           msg = buffer2;
00947           goto exit_on_error;
00948         }
00949       input->precision = g_realloc
00950         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00951       if (xmlHasProp (child, XML_PRECISION))
00952         input->precision[input->nvariables]
00953           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00954       else
00955         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00956       if (input->algorithm == ALGORITHM_SWEEP)
00957         {
00958           if (xmlHasProp (child, XML_NSWEEPS))
00959             {
00960               input->nsweeps = (unsigned int *)
00961                 g_realloc (input->nsweeps,
00962                            (1 + input->nvariables) * sizeof (unsigned int));
00963               input->nsweeps[input->nvariables]
00964                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00965             }
00966           else
00967             {
00968               snprintf (buffer2, 64, "%s %u: %s",
00969                         gettext ("Variable"),
00970                         input->nvariables + 1, gettext ("no sweeps number"));
00971               msg = buffer2;
00972               goto exit_on_error;
00973             }
00974 #if DEBUG
00975           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00976                    input->nsweeps[input->nvariables], input->
      nsimulations);
00977 #endif
00978         }
00979       if (input->algorithm == ALGORITHM_GENETIC)
00980         {
00981           // Obtaining bits representing each variable
00982           if (xmlHasProp (child, XML_NBITS))
00983             {
00984               input->nbits = (unsigned int *)
00985                 g_realloc (input->nbits,
```

```
00986                           (1 + input->nvariables) * sizeof (unsigned int));
00987               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00988               if (error_code || !i)
00989                 {
00990                   snprintf (buffer2, 64, "%s %u: %s",
00991                             gettext ("Variable"),
00992                             input->nvariables + 1,
00993                             gettext ("invalid bits number"));
00994                   msg = buffer2;
00995                   goto exit_on_error;
00996                 }
00997               input->nbits[input->nvariables] = i;
00998             }
00999           else
01000             {
01001               snprintf (buffer2, 64, "%s %u: %s",
01002                         gettext ("Variable"),
01003                         input->nvariables + 1, gettext ("no bits number"));
01004               msg = buffer2;
01005               goto exit_on_error;
01006             }
01007         }
01008       ++input->nvariables;
01009     }
01010   if (!input->nvariables)
01011     {
01012       msg = gettext ("No calibration variables");
01013       goto exit_on_error;
01014     }
01015
01016   // Getting the working directory
01017   input->directory = g_path_get_dirname (filename);
01018   input->name = g_path_get_basename (filename);
01019
01020   // Closing the XML document
01021   xmlFreeDoc (doc);
01022
01023 #if DEBUG
01024   fprintf (stderr, "input_open: end\n");
01025 #endif
01026   return 1;
01027
01028 exit_on_error:
01029   show_error (msg);
01030   input_free ();
01031 #if DEBUG
01032   fprintf (stderr, "input_open: end\n");
01033 #endif
01034   return 0;
01035 }
01036
01048 void
01049 calibrate_input (unsigned int simulation, char *input, GMappedFile * template)
01050 {
01051   unsigned int i;
01052   char buffer[32], value[32], *buffer2, *buffer3, *content;
01053   FILE *file;
01054   gsize length;
01055   GRegex *regex;
01056
01057 #if DEBUG
01058   fprintf (stderr, "calibrate_input: start\n");
01059 #endif
01060
01061   // Checking the file
01062   if (!template)
01063     goto calibrate_input_end;
01064
01065   // Opening template
01066   content = g_mapped_file_get_contents (template);
01067   length = g_mapped_file_get_length (template);
01068 #if DEBUG
01069   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01070            content);
01071 #endif
01072   file = g_fopen (input, "w");
01073
01074   // Parsing template
01075   for (i = 0; i < calibrate->nvariables; ++i)
01076     {
01077 #if DEBUG
01078       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01079 #endif
01080       snprintf (buffer, 32, "@variable%u@", i + 1);
01081       regex = g_regex_new (buffer, 0, 0, NULL);
01082       if (i == 0)
01083         {
```

```
01084           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01085                                              calibrate->label[i], 0, NULL);
01086 #if DEBUG
01087           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01088 #endif
01089         }
01090       else
01091         {
01092           length = strlen (buffer3);
01093           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01094                                              calibrate->label[i], 0, NULL);
01095           g_free (buffer3);
01096         }
01097       g_regex_unref (regex);
01098       length = strlen (buffer2);
01099       snprintf (buffer, 32, "@value%u@", i + 1);
01100       regex = g_regex_new (buffer, 0, 0, NULL);
01101       snprintf (value, 32, format[calibrate->precision[i]],
01102                 calibrate->value[simulation * calibrate->nvariables + i]);
01103
01104 #if DEBUG
01105       fprintf (stderr, "calibrate_input: value=%s\n", value);
01106 #endif
01107       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01108                                          0, NULL);
01109       g_free (buffer2);
01110       g_regex_unref (regex);
01111     }
01112
01113   // Saving input file
01114   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01115   g_free (buffer3);
01116   fclose (file);
01117
01118 calibrate_input_end:
01119 #if DEBUG
01120   fprintf (stderr, "calibrate_input: end\n");
01121 #endif
01122   return;
01123 }
01124
01135 double
01136 calibrate_parse (unsigned int simulation, unsigned int experiment)
01137 {
01138   unsigned int i;
01139   double e;
01140   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
01141     *buffer3, *buffer4;
01142   FILE *file_result;
01143
01144 #if DEBUG
01145   fprintf (stderr, "calibrate_parse: start\n");
01146   fprintf (stderr, "calibrate_parse: simulation=%u experiment=%u\n", simulation,
01147            experiment);
01148 #endif
01149
01150   // Opening input files
01151   for (i = 0; i < calibrate->ninputs; ++i)
01152     {
01153       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
01154 #if DEBUG
01155       fprintf (stderr, "calibrate_parse: i=%u input=%s\n", i, &input[i][0]);
01156 #endif
01157       calibrate_input (simulation, &input[i][0],
01158                        calibrate->file[i][experiment]);
01159     }
01160   for (; i < MAX_NINPUTS; ++i)
01161     strcpy (&input[i][0], "");
01162 #if DEBUG
01163   fprintf (stderr, "calibrate_parse: parsing end\n");
01164 #endif
01165
01166   // Performing the simulation
01167   snprintf (output, 32, "output-%u-%u", simulation, experiment);
01168   buffer2 = g_path_get_dirname (calibrate->simulator);
01169   buffer3 = g_path_get_basename (calibrate->simulator);
01170   buffer4 = g_build_filename (buffer2, buffer3, NULL);
01171   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
01172             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
01173             input[6], input[7], output);
01174   g_free (buffer4);
01175   g_free (buffer3);
01176   g_free (buffer2);
01177 #if DEBUG
01178   fprintf (stderr, "calibrate_parse: %s\n", buffer);
01179 #endif
01180   system (buffer);
```

```
01181
01182   // Checking the objective value function
01183   if (calibrate->evaluator)
01184     {
01185       snprintf (result, 32, "result-%u-%u", simulation, experiment);
01186       buffer2 = g_path_get_dirname (calibrate->evaluator);
01187       buffer3 = g_path_get_basename (calibrate->evaluator);
01188       buffer4 = g_build_filename (buffer2, buffer3, NULL);
01189       snprintf (buffer, 512, "\"%s\" %s %s %s",
01190                 buffer4, output, calibrate->experiment[experiment], result);
01191       g_free (buffer4);
01192       g_free (buffer3);
01193       g_free (buffer2);
01194 #if DEBUG
01195       fprintf (stderr, "calibrate_parse: %s\n", buffer);
01196 #endif
01197       system (buffer);
01198       file_result = g_fopen (result, "r");
01199       e = atof (fgets (buffer, 512, file_result));
01200       fclose (file_result);
01201     }
01202   else
01203     {
01204       strcpy (result, "");
01205       file_result = g_fopen (output, "r");
01206       e = atof (fgets (buffer, 512, file_result));
01207       fclose (file_result);
01208     }
01209
01210   // Removing files
01211 #if !DEBUG
01212   for (i = 0; i < calibrate->ninputs; ++i)
01213     {
01214       if (calibrate->file[i][0])
01215         {
01216           snprintf (buffer, 512, RM " %s", &input[i][0]);
01217           system (buffer);
01218         }
01219     }
01220   snprintf (buffer, 512, RM " %s %s", output, result);
01221   system (buffer);
01222 #endif
01223
01224 #if DEBUG
01225   fprintf (stderr, "calibrate_parse: end\n");
01226 #endif
01227
01228   // Returning the objective function
01229   return e * calibrate->weight[experiment];
01230 }
01231
01236 void
01237 calibrate_print ()
01238 {
01239   unsigned int i;
01240   char buffer[512];
01241 #if HAVE_MPI
01242   if (calibrate->mpi_rank)
01243     return;
01244 #endif
01245   printf ("%s\n", gettext ("Best result"));
01246   fprintf (calibrate->file_result, "%s\n", gettext ("Best result"));
01247   printf ("error = %.15le\n", calibrate->error_old[0]);
01248   fprintf (calibrate->file_result, "error = %.15le\n", calibrate->
      error_old[0]);
01249   for (i = 0; i < calibrate->nvariables; ++i)
01250     {
01251       snprintf (buffer, 512, "%s = %s\n",
01252                 calibrate->label[i], format[calibrate->precision[i]]);
01253       printf (buffer, calibrate->value_old[i]);
01254       fprintf (calibrate->file_result, buffer, calibrate->value_old[i]);
01255     }
01256   fflush (calibrate->file_result);
01257 }
01258
01267 void
01268 calibrate_save_variables (unsigned int simulation, double error)
01269 {
01270   unsigned int i;
01271   char buffer[64];
01272 #if DEBUG
01273   fprintf (stderr, "calibrate_save_variables: start\n");
01274 #endif
01275   for (i = 0; i < calibrate->nvariables; ++i)
01276     {
01277       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01278       fprintf (calibrate->file_variables, buffer,
```

```
01279                 calibrate->value[simulation * calibrate->nvariables + i]);
01280     }
01281   fprintf (calibrate->file_variables, "%.14le\n", error);
01282 #if DEBUG
01283   fprintf (stderr, "calibrate_save_variables: end\n");
01284 #endif
01285 }
01286
01295 void
01296 calibrate_best_thread (unsigned int simulation, double value)
01297 {
01298   unsigned int i, j;
01299   double e;
01300 #if DEBUG
01301   fprintf (stderr, "calibrate_best_thread: start\n");
01302 #endif
01303   if (calibrate->nsaveds < calibrate->nbest
01304       || value < calibrate->error_best[calibrate->nsaveds - 1])
01305     {
01306       g_mutex_lock (mutex);
01307       if (calibrate->nsaveds < calibrate->nbest)
01308         ++calibrate->nsaveds;
01309       calibrate->error_best[calibrate->nsaveds - 1] = value;
01310       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01311       for (i = calibrate->nsaveds; --i;)
01312         {
01313           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01314             {
01315               j = calibrate->simulation_best[i];
01316               e = calibrate->error_best[i];
01317               calibrate->simulation_best[i] = calibrate->
    simulation_best[i - 1];
01318               calibrate->error_best[i] = calibrate->error_best[i - 1];
01319               calibrate->simulation_best[i - 1] = j;
01320               calibrate->error_best[i - 1] = e;
01321             }
01322           else
01323             break;
01324         }
01325       g_mutex_unlock (mutex);
01326     }
01327 #if DEBUG
01328   fprintf (stderr, "calibrate_best_thread: end\n");
01329 #endif
01330 }
01331
01340 void
01341 calibrate_best_sequential (unsigned int simulation, double value)
01342 {
01343   unsigned int i, j;
01344   double e;
01345 #if DEBUG
01346   fprintf (stderr, "calibrate_best_sequential: start\n");
01347 #endif
01348   if (calibrate->nsaveds < calibrate->nbest
01349       || value < calibrate->error_best[calibrate->nsaveds - 1])
01350     {
01351       if (calibrate->nsaveds < calibrate->nbest)
01352         ++calibrate->nsaveds;
01353       calibrate->error_best[calibrate->nsaveds - 1] = value;
01354       calibrate->simulation_best[calibrate->nsaveds - 1] = simulation;
01355       for (i = calibrate->nsaveds; --i;)
01356         {
01357           if (calibrate->error_best[i] < calibrate->error_best[i - 1])
01358             {
01359               j = calibrate->simulation_best[i];
01360               e = calibrate->error_best[i];
01361               calibrate->simulation_best[i] = calibrate->
    simulation_best[i - 1];
01362               calibrate->error_best[i] = calibrate->error_best[i - 1];
01363               calibrate->simulation_best[i - 1] = j;
01364               calibrate->error_best[i - 1] = e;
01365             }
01366           else
01367             break;
01368         }
01369     }
01370 #if DEBUG
01371   fprintf (stderr, "calibrate_best_sequential: end\n");
01372 #endif
01373 }
01374
01382 void *
01383 calibrate_thread (ParallelData * data)
01384 {
01385   unsigned int i, j, thread;
01386   double e;
```

```
01387 #if DEBUG
01388   fprintf (stderr, "calibrate_thread: start\n");
01389 #endif
01390   thread = data->thread;
01391 #if DEBUG
01392   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01393            calibrate->thread[thread], calibrate->thread[thread + 1]);
01394 #endif
01395   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01396     {
01397       e = 0.;
01398       for (j = 0; j < calibrate->nexperiments; ++j)
01399         e += calibrate_parse (i, j);
01400       calibrate_best_thread (i, e);
01401       g_mutex_lock (mutex);
01402       calibrate_save_variables (i, e);
01403       g_mutex_unlock (mutex);
01404 #if DEBUG
01405       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01406 #endif
01407     }
01408 #if DEBUG
01409   fprintf (stderr, "calibrate_thread: end\n");
01410 #endif
01411   g_thread_exit (NULL);
01412   return NULL;
01413 }
01414
01419 void
01420 calibrate_sequential ()
01421 {
01422   unsigned int i, j;
01423   double e;
01424 #if DEBUG
01425   fprintf (stderr, "calibrate_sequential: start\n");
01426   fprintf (stderr, "calibrate_sequential: nstart=%u nend=%u\n",
01427            calibrate->nstart, calibrate->nend);
01428 #endif
01429   for (i = calibrate->nstart; i < calibrate->nend; ++i)
01430     {
01431       e = 0.;
01432       for (j = 0; j < calibrate->nexperiments; ++j)
01433         e += calibrate_parse (i, j);
01434       calibrate_best_sequential (i, e);
01435       calibrate_save_variables (i, e);
01436 #if DEBUG
01437       fprintf (stderr, "calibrate_sequential: i=%u e=%lg\n", i, e);
01438 #endif
01439     }
01440 #if DEBUG
01441   fprintf (stderr, "calibrate_sequential: end\n");
01442 #endif
01443 }
01444
01456 void
01457 calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
01458                  double *error_best)
01459 {
01460   unsigned int i, j, k, s[calibrate->nbest];
01461   double e[calibrate->nbest];
01462 #if DEBUG
01463   fprintf (stderr, "calibrate_merge: start\n");
01464 #endif
01465   i = j = k = 0;
01466   do
01467     {
01468       if (i == calibrate->nsaveds)
01469         {
01470           s[k] = simulation_best[j];
01471           e[k] = error_best[j];
01472           ++j;
01473           ++k;
01474           if (j == nsaveds)
01475             break;
01476         }
01477       else if (j == nsaveds)
01478         {
01479           s[k] = calibrate->simulation_best[i];
01480           e[k] = calibrate->error_best[i];
01481           ++i;
01482           ++k;
01483           if (i == calibrate->nsaveds)
01484             break;
01485         }
01486       else if (calibrate->error_best[i] > error_best[j])
01487         {
01488           s[k] = simulation_best[j];
```

```
01489           e[k] = error_best[j];
01490           ++j;
01491           ++k;
01492         }
01493       else
01494         {
01495           s[k] = calibrate->simulation_best[i];
01496           e[k] = calibrate->error_best[i];
01497           ++i;
01498           ++k;
01499         }
01500     }
01501   while (k < calibrate->nbest);
01502   calibrate->nsaveds = k;
01503   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01504   memcpy (calibrate->error_best, e, k * sizeof (double));
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_merge: end\n");
01507 #endif
01508 }
01509
01514 #if HAVE_MPI
01515 void
01516 calibrate_synchronise ()
01517 {
01518   unsigned int i, nsaveds, simulation_best[calibrate->nbest];
01519   double error_best[calibrate->nbest];
01520   MPI_Status mpi_stat;
01521 #if DEBUG
01522   fprintf (stderr, "calibrate_synchronise: start\n");
01523 #endif
01524   if (calibrate->mpi_rank == 0)
01525     {
01526       for (i = 1; i < ntasks; ++i)
01527         {
01528           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
01529           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
01530                     MPI_COMM_WORLD, &mpi_stat);
01531           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
01532                     MPI_COMM_WORLD, &mpi_stat);
01533           calibrate_merge (nsaveds, simulation_best, error_best);
01534         }
01535     }
01536   else
01537     {
01538       MPI_Send (&calibrate->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
01539       MPI_Send (calibrate->simulation_best, calibrate->nsaveds, MPI_INT, 0, 1,
01540                 MPI_COMM_WORLD);
01541       MPI_Send (calibrate->error_best, calibrate->nsaveds, MPI_DOUBLE, 0, 1,
01542                 MPI_COMM_WORLD);
01543     }
01544 #if DEBUG
01545   fprintf (stderr, "calibrate_synchronise: end\n");
01546 #endif
01547 }
01548 #endif
01549
01554 void
01555 calibrate_sweep ()
01556 {
01557   unsigned int i, j, k, l;
01558   double e;
01559   GThread *thread[nthreads];
01560   ParallelData data[nthreads];
01561 #if DEBUG
01562   fprintf (stderr, "calibrate_sweep: start\n");
01563 #endif
01564   for (i = 0; i < calibrate->nsimulations; ++i)
01565     {
01566       k = i;
01567       for (j = 0; j < calibrate->nvariables; ++j)
01568         {
01569           l = k % calibrate->nsweeps[j];
01570           k /= calibrate->nsweeps[j];
01571           e = calibrate->rangemin[j];
01572           if (calibrate->nsweeps[j] > 1)
01573             e += l * (calibrate->rangemax[j] - calibrate->rangemin[j])
01574               / (calibrate->nsweeps[j] - 1);
01575           calibrate->value[i * calibrate->nvariables + j] = e;
01576         }
01577     }
01578   calibrate->nsaveds = 0;
01579   if (nthreads <= 1)
01580     calibrate_sequential ();
01581   else
01582     {
01583       for (i = 0; i < nthreads; ++i)
```

```
01584         {
01585           data[i].thread = i;
01586           thread[i]
01587             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01588         }
01589       for (i = 0; i < nthreads; ++i)
01590         g_thread_join (thread[i]);
01591     }
01592 #if HAVE_MPI
01593   // Communicating tasks results
01594   calibrate_synchronise ();
01595 #endif
01596 #if DEBUG
01597   fprintf (stderr, "calibrate_sweep: end\n");
01598 #endif
01599 }
01600
01605 void
01606 calibrate_MonteCarlo ()
01607 {
01608   unsigned int i, j;
01609   GThread *thread[nthreads];
01610   ParallelData data[nthreads];
01611 #if DEBUG
01612   fprintf (stderr, "calibrate_MonteCarlo: start\n");
01613 #endif
01614   for (i = 0; i < calibrate->nsimulations; ++i)
01615     for (j = 0; j < calibrate->nvariables; ++j)
01616       calibrate->value[i * calibrate->nvariables + j]
01617         = calibrate->rangemin[j] + gsl_rng_uniform (calibrate->rng)
01618         * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01619   calibrate->nsaveds = 0;
01620   if (nthreads <= 1)
01621     calibrate_sequential ();
01622   else
01623     {
01624       for (i = 0; i < nthreads; ++i)
01625         {
01626           data[i].thread = i;
01627           thread[i]
01628             = g_thread_new (NULL, (void (*)) calibrate_thread, &data[i]);
01629         }
01630       for (i = 0; i < nthreads; ++i)
01631         g_thread_join (thread[i]);
01632     }
01633 #if HAVE_MPI
01634   // Communicating tasks results
01635   calibrate_synchronise ();
01636 #endif
01637 #if DEBUG
01638   fprintf (stderr, "calibrate_MonteCarlo: end\n");
01639 #endif
01640 }
01641
01649 double
01650 calibrate_genetic_objective (Entity * entity)
01651 {
01652   unsigned int j;
01653   double objective;
01654   char buffer[64];
01655 #if DEBUG
01656   fprintf (stderr, "calibrate_genetic_objective: start\n");
01657 #endif
01658   for (j = 0; j < calibrate->nvariables; ++j)
01659     {
01660       calibrate->value[entity->id * calibrate->nvariables + j]
01661         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01662     }
01663   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01664     objective += calibrate_parse (entity->id, j);
01665   g_mutex_lock (mutex);
01666   for (j = 0; j < calibrate->nvariables; ++j)
01667     {
01668       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01669       fprintf (calibrate->file_variables, buffer,
01670                genetic_get_variable (entity, calibrate->genetic_variable + j));
01671     }
01672   fprintf (calibrate->file_variables, "%.14le\n", objective);
01673   g_mutex_unlock (mutex);
01674 #if DEBUG
01675   fprintf (stderr, "calibrate_genetic_objective: end\n");
01676 #endif
01677   return objective;
01678 }
01679
01684 void
01685 calibrate_genetic ()
```

```
01686 {
01687   char *best_genome;
01688   double best_objective, *best_variable;
01689 #if DEBUG
01690   fprintf (stderr, "calibrate_genetic: start\n");
01691   fprintf (stderr, "calibrate_genetic: ntasks=%u nthreads=%u\n", ntasks,
01692            nthreads);
01693   fprintf (stderr,
01694            "calibrate_genetic: nvariables=%u population=%u generations=%u\n",
01695            calibrate->nvariables, calibrate->nsimulations,
01696            calibrate->niterations);
01697   fprintf (stderr,
01698            "calibrate_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01699            calibrate->mutation_ratio, calibrate->
      reproduction_ratio,
01700            calibrate->adaptation_ratio);
01701 #endif
01702   genetic_algorithm_default (calibrate->nvariables,
01703                              calibrate->genetic_variable,
01704                              calibrate->nsimulations,
01705                              calibrate->niterations,
01706                              calibrate->mutation_ratio,
01707                              calibrate->reproduction_ratio,
01708                              calibrate->adaptation_ratio,
01709                              &calibrate_genetic_objective,
01710                              &best_genome, &best_variable, &best_objective);
01711 #if DEBUG
01712   fprintf (stderr, "calibrate_genetic: the best\n");
01713 #endif
01714   calibrate->error_old = (double *) g_malloc (sizeof (double));
01715   calibrate->value_old
01716     = (double *) g_malloc (calibrate->nvariables * sizeof (double));
01717   calibrate->error_old[0] = best_objective;
01718   memcpy (calibrate->value_old, best_variable,
01719           calibrate->nvariables * sizeof (double));
01720   g_free (best_genome);
01721   g_free (best_variable);
01722   calibrate_print ();
01723 #if DEBUG
01724   fprintf (stderr, "calibrate_genetic: end\n");
01725 #endif
01726 }
01727
01732 void
01733 calibrate_save_old ()
01734 {
01735   unsigned int i, j;
01736 #if DEBUG
01737   fprintf (stderr, "calibrate_save_old: start\n");
01738 #endif
01739   memcpy (calibrate->error_old, calibrate->error_best,
01740           calibrate->nbest * sizeof (double));
01741   for (i = 0; i < calibrate->nbest; ++i)
01742     {
01743       j = calibrate->simulation_best[i];
01744       memcpy (calibrate->value_old + i * calibrate->nvariables,
01745               calibrate->value + j * calibrate->nvariables,
01746               calibrate->nvariables * sizeof (double));
01747     }
01748 #if DEBUG
01749   for (i = 0; i < calibrate->nvariables; ++i)
01750     fprintf (stderr, "calibrate_save_old: best variable %u=%lg\n",
01751              i, calibrate->value_old[i]);
01752   fprintf (stderr, "calibrate_save_old: end\n");
01753 #endif
01754 }
01755
01761 void
01762 calibrate_merge_old ()
01763 {
01764   unsigned int i, j, k;
01765   double v[calibrate->nbest * calibrate->nvariables], e[calibrate->
      nbest],
01766     *enew, *eold;
01767 #if DEBUG
01768   fprintf (stderr, "calibrate_merge_old: start\n");
01769 #endif
01770   enew = calibrate->error_best;
01771   eold = calibrate->error_old;
01772   i = j = k = 0;
01773   do
01774     {
01775       if (*enew < *eold)
01776         {
01777           memcpy (v + k * calibrate->nvariables,
01778                   calibrate->value
01779                   + calibrate->simulation_best[i] * calibrate->
```

```
      nvariables,
01780                   calibrate->nvariables * sizeof (double));
01781           e[k] = *enew;
01782           ++k;
01783           ++enew;
01784           ++i;
01785         }
01786       else
01787         {
01788           memcpy (v + k * calibrate->nvariables,
01789                   calibrate->value_old + j * calibrate->nvariables,
01790                   calibrate->nvariables * sizeof (double));
01791           e[k] = *eold;
01792           ++k;
01793           ++eold;
01794           ++j;
01795         }
01796     }
01797   while (k < calibrate->nbest);
01798   memcpy (calibrate->value_old, v, k * calibrate->nvariables * sizeof (double));
01799   memcpy (calibrate->error_old, e, k * sizeof (double));
01800 #if DEBUG
01801   fprintf (stderr, "calibrate_merge_old: end\n");
01802 #endif
01803 }
01804
01810 void
01811 calibrate_refine ()
01812 {
01813   unsigned int i, j;
01814   double d;
01815 #if HAVE_MPI
01816   MPI_Status mpi_stat;
01817 #endif
01818 #if DEBUG
01819   fprintf (stderr, "calibrate_refine: start\n");
01820 #endif
01821 #if HAVE_MPI
01822   if (!calibrate->mpi_rank)
01823     {
01824 #endif
01825       for (j = 0; j < calibrate->nvariables; ++j)
01826         {
01827           calibrate->rangemin[j] = calibrate->rangemax[j]
01828             = calibrate->value_old[j];
01829         }
01830       for (i = 0; ++i < calibrate->nbest;)
01831         {
01832           for (j = 0; j < calibrate->nvariables; ++j)
01833             {
01834               calibrate->rangemin[j]
01835                 = fmin (calibrate->rangemin[j],
01836                         calibrate->value_old[i * calibrate->nvariables + j]);
01837               calibrate->rangemax[j]
01838                 = fmax (calibrate->rangemax[j],
01839                         calibrate->value_old[i * calibrate->nvariables + j]);
01840             }
01841         }
01842       for (j = 0; j < calibrate->nvariables; ++j)
01843         {
01844           d = 0.5 * calibrate->tolerance
01845             * (calibrate->rangemax[j] - calibrate->rangemin[j]);
01846           calibrate->rangemin[j] -= d;
01847           calibrate->rangemin[j]
01848             = fmax (calibrate->rangemin[j], calibrate->rangeminabs[j]);
01849           calibrate->rangemax[j] += d;
01850           calibrate->rangemax[j]
01851             = fmin (calibrate->rangemax[j], calibrate->rangemaxabs[j]);
01852           printf ("%s min=%lg max=%lg\n", calibrate->label[j],
01853                   calibrate->rangemin[j], calibrate->rangemax[j]);
01854           fprintf (calibrate->file_result, "%s min=%lg max=%lg\n",
01855                    calibrate->label[j], calibrate->rangemin[j],
01856                    calibrate->rangemax[j]);
01857         }
01858 #if HAVE_MPI
01859       for (i = 1; i < ntasks; ++i)
01860         {
01861           MPI_Send (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, i,
01862                     1, MPI_COMM_WORLD);
01863           MPI_Send (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, i,
01864                     1, MPI_COMM_WORLD);
01865         }
01866     }
01867   else
01868     {
01869       MPI_Recv (calibrate->rangemin, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01870                 MPI_COMM_WORLD, &mpi_stat);
```

```
01871       MPI_Recv (calibrate->rangemax, calibrate->nvariables, MPI_DOUBLE, 0, 1,
01872                 MPI_COMM_WORLD, &mpi_stat);
01873     }
01874 #endif
01875 #if DEBUG
01876   fprintf (stderr, "calibrate_refine: end\n");
01877 #endif
01878 }
01879
01884 void
01885 calibrate_iterate ()
01886 {
01887   unsigned int i;
01888 #if DEBUG
01889   fprintf (stderr, "calibrate_iterate: start\n");
01890 #endif
01891   calibrate->error_old
01892     = (double *) g_malloc (calibrate->nbest * sizeof (double));
01893   calibrate->value_old = (double *)
01894     g_malloc (calibrate->nbest * calibrate->nvariables * sizeof (double));
01895   calibrate_step ();
01896   calibrate_save_old ();
01897   calibrate_refine ();
01898   calibrate_print ();
01899   for (i = 1; i < calibrate->niterations; ++i)
01900     {
01901       calibrate_step ();
01902       calibrate_merge_old ();
01903       calibrate_refine ();
01904       calibrate_print ();
01905     }
01906 #if DEBUG
01907   fprintf (stderr, "calibrate_iterate: end\n");
01908 #endif
01909 }
01910
01915 void
01916 calibrate_free ()
01917 {
01918   unsigned int i, j;
01919 #if DEBUG
01920   fprintf (stderr, "calibrate_free: start\n");
01921 #endif
01922   for (i = 0; i < calibrate->nexperiments; ++i)
01923     {
01924       for (j = 0; j < calibrate->ninputs; ++j)
01925         g_mapped_file_unref (calibrate->file[j][i]);
01926     }
01927   for (i = 0; i < calibrate->ninputs; ++i)
01928     g_free (calibrate->file[i]);
01929   g_free (calibrate->error_old);
01930   g_free (calibrate->value_old);
01931   g_free (calibrate->value);
01932   g_free (calibrate->genetic_variable);
01933   g_free (calibrate->rangemax);
01934   g_free (calibrate->rangemin);
01935 #if DEBUG
01936   fprintf (stderr, "calibrate_free: end\n");
01937 #endif
01938 }
01939
01944 void
01945 calibrate_new ()
01946 {
01947   GTimeZone *tz;
01948   GDateTime *t0, *t;
01949   unsigned int i, j, *nbits;
01950
01951 #if DEBUG
01952   fprintf (stderr, "calibrate_new: start\n");
01953 #endif
01954
01955   // Getting initial time
01956 #if DEBUG
01957   fprintf (stderr, "calibrate_new: getting initial time\n");
01958 #endif
01959   tz = g_time_zone_new_utc ();
01960   t0 = g_date_time_new_now (tz);
01961
01962   // Obtaining and initing the pseudo-random numbers generator seed
01963 #if DEBUG
01964   fprintf (stderr, "calibrate_new: getting initial seed\n");
01965 #endif
01966   calibrate->seed = input->seed;
01967   gsl_rng_set (calibrate->rng, calibrate->seed);
01968
01969   // Replacing the working directory
```

```
01970 #if DEBUG
01971   fprintf (stderr, "calibrate_new: replacing the working directory\n");
01972 #endif
01973   g_chdir (input->directory);
01974
01975   // Getting results file names
01976   calibrate->result = input->result;
01977   calibrate->variables = input->variables;
01978
01979   // Obtaining the simulator file
01980   calibrate->simulator = input->simulator;
01981
01982   // Obtaining the evaluator file
01983   calibrate->evaluator = input->evaluator;
01984
01985   // Reading the algorithm
01986   calibrate->algorithm = input->algorithm;
01987   switch (calibrate->algorithm)
01988     {
01989     case ALGORITHM_MONTE_CARLO:
01990       calibrate_step = calibrate_MonteCarlo;
01991       break;
01992     case ALGORITHM_SWEEP:
01993       calibrate_step = calibrate_sweep;
01994       break;
01995     default:
01996       calibrate_step = calibrate_genetic;
01997       calibrate->mutation_ratio = input->mutation_ratio;
01998       calibrate->reproduction_ratio = input->
    reproduction_ratio;
01999       calibrate->adaptation_ratio = input->adaptation_ratio;
02000     }
02001   calibrate->nsimulations = input->nsimulations;
02002   calibrate->niterations = input->niterations;
02003   calibrate->nbest = input->nbest;
02004   calibrate->tolerance = input->tolerance;
02005
02006   calibrate->simulation_best
02007     = (unsigned int *) alloca (calibrate->nbest * sizeof (unsigned int));
02008   calibrate->error_best
02009     = (double *) alloca (calibrate->nbest * sizeof (double));
02010
02011   // Reading the experimental data
02012 #if DEBUG
02013   fprintf (stderr, "calibrate_new: current directory=%s\n",
02014            g_get_current_dir ());
02015 #endif
02016   calibrate->nexperiments = input->nexperiments;
02017   calibrate->ninputs = input->ninputs;
02018   calibrate->experiment = input->experiment;
02019   calibrate->weight = input->weight;
02020   for (i = 0; i < input->ninputs; ++i)
02021     {
02022       calibrate->template[i] = input->template[i];
02023       calibrate->file[i]
02024         = g_malloc (input->nexperiments * sizeof (GMappedFile *));
02025     }
02026   for (i = 0; i < input->nexperiments; ++i)
02027     {
02028 #if DEBUG
02029       fprintf (stderr, "calibrate_new: i=%u\n", i);
02030       fprintf (stderr, "calibrate_new: experiment=%s\n",
02031                calibrate->experiment[i]);
02032       fprintf (stderr, "calibrate_new: weight=%lg\n", calibrate->weight[i]);
02033 #endif
02034       for (j = 0; j < input->ninputs; ++j)
02035         {
02036 #if DEBUG
02037           fprintf (stderr, "calibrate_new: template%u\n", j + 1);
02038           fprintf (stderr, "calibrate_new: experiment=%u template%u=%s\n",
02039                    i, j + 1, calibrate->template[j][i]);
02040 #endif
02041           calibrate->file[j][i]
02042             = g_mapped_file_new (input->template[j][i], 0, NULL);
02043         }
02044     }
02045
02046   // Reading the variables data
02047 #if DEBUG
02048   fprintf (stderr, "calibrate_new: reading variables\n");
02049 #endif
02050   calibrate->nvariables = input->nvariables;
02051   calibrate->label = input->label;
02052   j = input->nvariables * sizeof (double);
02053   calibrate->rangemin = (double *) g_malloc (j);
02054   calibrate->rangemax = (double *) g_malloc (j);
02055   memcpy (calibrate->rangemin, input->rangemin, j);
```

```
02056   memcpy (calibrate->rangemax, input->rangemax, j);
02057   calibrate->rangeminabs = input->rangeminabs;
02058   calibrate->rangemaxabs = input->rangemaxabs;
02059   calibrate->precision = input->precision;
02060   calibrate->nsweeps = input->nsweeps;
02061   nbits = input->nbits;
02062   if (input->algorithm == ALGORITHM_SWEEP)
02063     calibrate->nsimulations = 1;
02064   else if (input->algorithm == ALGORITHM_GENETIC)
02065     for (i = 0; i < input->nvariables; ++i)
02066       {
02067         if (calibrate->algorithm == ALGORITHM_SWEEP)
02068           {
02069             calibrate->nsimulations *= input->nsweeps[i];
02070 #if DEBUG
02071             fprintf (stderr, "calibrate_new: nsweeps=%u nsimulations=%u\n",
02072                      calibrate->nsweeps[i], calibrate->nsimulations);
02073 #endif
02074           }
02075       }
02076
02077   // Allocating values
02078 #if DEBUG
02079   fprintf (stderr, "calibrate_new: allocating variables\n");
02080   fprintf (stderr, "calibrate_new: nvariables=%u\n", calibrate->nvariables);
02081 #endif
02082   calibrate->genetic_variable = NULL;
02083   if (calibrate->algorithm == ALGORITHM_GENETIC)
02084     {
02085       calibrate->genetic_variable = (GeneticVariable *)
02086         g_malloc (calibrate->nvariables * sizeof (GeneticVariable));
02087       for (i = 0; i < calibrate->nvariables; ++i)
02088         {
02089 #if DEBUG
02090           fprintf (stderr, "calibrate_new: i=%u min=%lg max=%lg nbits=%u\n",
02091                    i, calibrate->rangemin[i], calibrate->rangemax[i], nbits[i]);
02092 #endif
02093           calibrate->genetic_variable[i].minimum = calibrate->
      rangemin[i];
02094           calibrate->genetic_variable[i].maximum = calibrate->
      rangemax[i];
02095           calibrate->genetic_variable[i].nbits = nbits[i];
02096         }
02097     }
02098 #if DEBUG
02099   fprintf (stderr, "calibrate_new: nvariables=%u nsimulations=%u\n",
02100            calibrate->nvariables, calibrate->nsimulations);
02101 #endif
02102   calibrate->value = (double *) g_malloc (calibrate->nsimulations *
02103                                           calibrate->nvariables *
02104                                           sizeof (double));
02105
02106   // Calculating simulations to perform on each task
02107 #if HAVE_MPI
02108 #if DEBUG
02109   fprintf (stderr, "calibrate_new: rank=%u ntasks=%u\n",
02110            calibrate->mpi_rank, ntasks);
02111 #endif
02112   calibrate->nstart = calibrate->mpi_rank * calibrate->
      nsimulations / ntasks;
02113   calibrate->nend = (1 + calibrate->mpi_rank) * calibrate->
      nsimulations
02114     / ntasks;
02115 #else
02116   calibrate->nstart = 0;
02117   calibrate->nend = calibrate->nsimulations;
02118 #endif
02119 #if DEBUG
02120   fprintf (stderr, "calibrate_new: nstart=%u nend=%u\n", calibrate->nstart,
02121            calibrate->nend);
02122 #endif
02123
02124   // Calculating simulations to perform on each thread
02125   calibrate->thread
02126     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
02127   for (i = 0; i <= nthreads; ++i)
02128     {
02129       calibrate->thread[i] = calibrate->nstart
02130         + i * (calibrate->nend - calibrate->nstart) / nthreads;
02131 #if DEBUG
02132       fprintf (stderr, "calibrate_new: i=%u thread=%u\n", i,
02133                calibrate->thread[i]);
02134 #endif
02135     }
02136
02137   // Opening result files
02138   calibrate->file_result = g_fopen (calibrate->result, "w");
```

```
02139   calibrate->file_variables = g_fopen (calibrate->variables, "w");
02140
02141   // Performing the algorithm
02142   switch (calibrate->algorithm)
02143     {
02144       // Genetic algorithm
02145     case ALGORITHM_GENETIC:
02146       calibrate_genetic ();
02147       break;
02148
02149       // Iterative algorithm
02150     default:
02151       calibrate_iterate ();
02152     }
02153
02154   // Getting calculation time
02155   t = g_date_time_new_now (tz);
02156   calibrate->calculation_time = 0.000001 * g_date_time_difference (t, t0);
02157   g_date_time_unref (t);
02158   g_date_time_unref (t0);
02159   g_time_zone_unref (tz);
02160   printf ("%s = %.6lg s\n",
02161           gettext ("Calculation time"), calibrate->calculation_time);
02162   fprintf (calibrate->file_result, "%s = %.6lg s\n",
02163            gettext ("Calculation time"), calibrate->calculation_time);
02164
02165   // Closing result files
02166   fclose (calibrate->file_variables);
02167   fclose (calibrate->file_result);
02168
02169 #if DEBUG
02170   fprintf (stderr, "calibrate_new: end\n");
02171 #endif
02172 }
02173
02174 #if HAVE_GTK
02175
02182 void
02183 input_save (char *filename)
02184 {
02185   unsigned int i, j;
02186   char *buffer;
02187   xmlDoc *doc;
02188   xmlNode *node, *child;
02189   GFile *file, *file2;
02190
02191   // Getting the input file directory
02192   input->name = g_path_get_basename (filename);
02193   input->directory = g_path_get_dirname (filename);
02194   file = g_file_new_for_path (input->directory);
02195
02196   // Opening the input file
02197   doc = xmlNewDoc ((const xmlChar *) "1.0");
02198
02199   // Setting root XML node
02200   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02201   xmlDocSetRootElement (doc, node);
02202
02203   // Adding properties to the root XML node
02204   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
02205     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02206   if (xmlStrcmp ((const xmlChar *) input->variables, variables_name))
02207     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->variables);
02208   file2 = g_file_new_for_path (input->simulator);
02209   buffer = g_file_get_relative_path (file, file2);
02210   g_object_unref (file2);
02211   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02212   g_free (buffer);
02213   if (input->evaluator)
02214     {
02215       file2 = g_file_new_for_path (input->evaluator);
02216       buffer = g_file_get_relative_path (file, file2);
02217       g_object_unref (file2);
02218       if (xmlStrlen ((xmlChar *) buffer))
02219         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02220       g_free (buffer);
02221     }
02222   if (input->seed != DEFAULT_RANDOM_SEED)
02223     xml_node_set_uint (node, XML_SEED, input->seed);
02224
02225   // Setting the algorithm
02226   buffer = (char *) g_malloc (64);
02227   switch (input->algorithm)
02228     {
02229     case ALGORITHM_MONTE_CARLO:
02230       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02231       snprintf (buffer, 64, "%u", input->nsimulations);
```

```
02232       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02233       snprintf (buffer, 64, "%u", input->niterations);
02234       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02235       snprintf (buffer, 64, "%.3lg", input->tolerance);
02236       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02237       snprintf (buffer, 64, "%u", input->nbest);
02238       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02239       break;
02240     case ALGORITHM_SWEEP:
02241       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02242       snprintf (buffer, 64, "%u", input->niterations);
02243       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02244       snprintf (buffer, 64, "%.3lg", input->tolerance);
02245       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02246       snprintf (buffer, 64, "%u", input->nbest);
02247       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02248       break;
02249     default:
02250       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02251       snprintf (buffer, 64, "%u", input->nsimulations);
02252       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02253       snprintf (buffer, 64, "%u", input->niterations);
02254       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02255       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02256       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02257       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02258       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02259       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02260       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02261       break;
02262     }
02263   g_free (buffer);
02264
02265   // Setting the experimental data
02266   for (i = 0; i < input->nexperiments; ++i)
02267     {
02268       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02269       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02270       if (input->weight[i] != 1.)
02271         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02272       for (j = 0; j < input->ninputs; ++j)
02273         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02274     }
02275
02276   // Setting the variables data
02277   for (i = 0; i < input->nvariables; ++i)
02278     {
02279       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02280       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02281       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02282       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02283         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM, input->
      rangeminabs[i]);
02284       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02285       if (input->rangemaxabs[i] != G_MAXDOUBLE)
02286         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM, input->
      rangemaxabs[i]);
02287       if (input->precision[i] != DEFAULT_PRECISION)
02288         xml_node_set_uint (child, XML_PRECISION, input->
      precision[i]);
02289       if (input->algorithm == ALGORITHM_SWEEP)
02290         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02291       else if (input->algorithm == ALGORITHM_GENETIC)
02292         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02293     }
02294
02295   // Saving the XML file
02296   xmlSaveFormatFile (filename, doc, 1);
02297
02298   // Freeing memory
02299   xmlFreeDoc (doc);
02300 }
02301
02306 void
02307 options_new ()
02308 {
02309   options->label_processors
02310     = (GtkLabel *) gtk_label_new (gettext ("Processors number"));
02311   options->spin_processors
02312     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02313   gtk_widget_set_tooltip_text
02314     (GTK_WIDGET (options->spin_processors),
```

```
02315       gettext ("Number of threads to perform the calibration/optimization"));
02316   gtk_spin_button_set_value (options->spin_processors, (gdouble)
    nthreads);
02317   options->label_seed = (GtkLabel *)
02318     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
02319   options->spin_seed = (GtkSpinButton *)
02320     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
02321   gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (options->spin_seed),
02323      gettext ("Seed to init the pseudo-random numbers generator"));
02324   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
02325   options->grid = (GtkGrid *) gtk_grid_new ();
02326   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_processors),
02327                    0, 0, 1, 1);
02328   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_processors),
02329                    1, 0, 1, 1);
02330   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 1, 1, 1);
02331   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 1, 1, 1);
02332   gtk_widget_show_all (GTK_WIDGET (options->grid));
02333   options->dialog = (GtkDialog *)
02334     gtk_dialog_new_with_buttons (gettext ("Options"),
02335                                  window->window,
02336                                  GTK_DIALOG_MODAL,
02337                                  gettext ("_OK"), GTK_RESPONSE_OK,
02338                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02339                                  NULL);
02340   gtk_container_add
02341     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
02342      GTK_WIDGET (options->grid));
02343   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
02344     {
02345       nthreads = gtk_spin_button_get_value_as_int (options->spin_processors);
02346       input->seed
02347         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
02348     }
02349   gtk_widget_destroy (GTK_WIDGET (options->dialog));
02350 }
02351
02356 void
02357 running_new ()
02358 {
02359 #if DEBUG
02360   fprintf (stderr, "running_new: start\n");
02361 #endif
02362   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
02363   running->dialog = (GtkDialog *)
02364     gtk_dialog_new_with_buttons (gettext ("Calculating"),
02365                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
02366   gtk_container_add
02367     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
02368      GTK_WIDGET (running->label));
02369   gtk_widget_show_all (GTK_WIDGET (running->dialog));
02370 #if DEBUG
02371   fprintf (stderr, "running_new: end\n");
02372 #endif
02373 }
02374
02380 int
02381 window_save ()
02382 {
02383   char *buffer;
02384   GtkFileChooserDialog *dlg;
02385
02386 #if DEBUG
02387   fprintf (stderr, "window_save: start\n");
02388 #endif
02389
02390   // Opening the saving dialog
02391   dlg = (GtkFileChooserDialog *)
02392     gtk_file_chooser_dialog_new (gettext ("Save file"),
02393                                  window->window,
02394                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02395                                  gettext ("_Cancel"),
02396                                  GTK_RESPONSE_CANCEL,
02397                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02398   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02399   buffer = g_build_filename (input->directory, input->name, NULL);
02400   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02401   g_free (buffer);
02402
02403   // If OK response then saving
02404   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02405     {
02406
02407       // Adding properties to the root XML node
02408       input->simulator = gtk_file_chooser_get_filename
02409         (GTK_FILE_CHOOSER (window->button_simulator));
```

```
02410       if (gtk_toggle_button_get_active
02411           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02412         input->evaluator = gtk_file_chooser_get_filename
02413           (GTK_FILE_CHOOSER (window->button_evaluator));
02414       else
02415         input->evaluator = NULL;
02416       input->result
02417         = (char *) xmlStrdup ((const xmlChar *)
02418                               gtk_entry_get_text (window->entry_result));
02419       input->variables
02420         = (char *) xmlStrdup ((const xmlChar *)
02421                               gtk_entry_get_text (window->entry_variables));
02422
02423       // Setting the algorithm
02424       switch (window_get_algorithm ())
02425         {
02426         case ALGORITHM_MONTE_CARLO:
02427           input->algorithm = ALGORITHM_MONTE_CARLO;
02428           input->nsimulations
02429             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02430           input->niterations
02431             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02432           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02433           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02434           break;
02435         case ALGORITHM_SWEEP:
02436           input->algorithm = ALGORITHM_SWEEP;
02437           input->niterations
02438             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02439           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02440           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02441           break;
02442         default:
02443           input->algorithm = ALGORITHM_GENETIC;
02444           input->nsimulations
02445             = gtk_spin_button_get_value_as_int (window->spin_population);
02446           input->niterations
02447             = gtk_spin_button_get_value_as_int (window->spin_generations);
02448           input->mutation_ratio
02449             = gtk_spin_button_get_value (window->spin_mutation);
02450           input->reproduction_ratio
02451             = gtk_spin_button_get_value (window->spin_reproduction);
02452           input->adaptation_ratio
02453             = gtk_spin_button_get_value (window->spin_adaptation);
02454           break;
02455         }
02456
02457       // Saving the XML file
02458       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02459       input_save (buffer);
02460
02461       // Closing and freeing memory
02462       g_free (buffer);
02463       gtk_widget_destroy (GTK_WIDGET (dlg));
02464 #if DEBUG
02465       fprintf (stderr, "window_save: end\n");
02466 #endif
02467       return 1;
02468     }
02469
02470   // Closing and freeing memory
02471   gtk_widget_destroy (GTK_WIDGET (dlg));
02472 #if DEBUG
02473   fprintf (stderr, "window_save: end\n");
02474 #endif
02475   return 0;
02476 }
02477
02482 void
02483 window_run ()
02484 {
02485   unsigned int i;
02486   char *msg, *msg2, buffer[64], buffer2[64];
02487 #if DEBUG
02488   fprintf (stderr, "window_run: start\n");
02489 #endif
02490   if (!window_save ())
02491     {
02492 #if DEBUG
02493       fprintf (stderr, "window_run: end\n");
02494 #endif
02495       return;
02496     }
```

```
02497   running_new ();
02498   while (gtk_events_pending ())
02499     gtk_main_iteration ();
02500   calibrate_new ();
02501   gtk_widget_destroy (GTK_WIDGET (running->dialog));
02502   snprintf (buffer, 64, "error = %.15le\n", calibrate->error_old[0]);
02503   msg2 = g_strdup (buffer);
02504   for (i = 0; i < calibrate->nvariables; ++i, msg2 = msg)
02505     {
02506       snprintf (buffer, 64, "%s = %s\n",
02507                 calibrate->label[i], format[calibrate->precision[i]]);
02508       snprintf (buffer2, 64, buffer, calibrate->value_old[i]);
02509       msg = g_strconcat (msg2, buffer2, NULL);
02510       g_free (msg2);
02511     }
02512   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
02513             calibrate->calculation_time);
02514   msg = g_strconcat (msg2, buffer, NULL);
02515   g_free (msg2);
02516   show_message (gettext ("Best result"), msg, INFO_TYPE);
02517   g_free (msg);
02518   calibrate_free ();
02519 #if DEBUG
02520   fprintf (stderr, "window_run: end\n");
02521 #endif
02522 }
02523
02528 void
02529 window_help ()
02530 {
02531   char *buffer, *buffer2;
02532   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
02533                               gettext ("user-manual.pdf"), NULL);
02534   buffer = g_filename_to_uri (buffer2, NULL, NULL);
02535   g_free (buffer2);
02536   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
02537   g_free (buffer);
02538 }
02539
02544 void
02545 window_about ()
02546 {
02547   static const gchar *authors[] = {
02548     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
02549     "Borja Latorre Garcés <borja.latorre@csic.es>",
02550     NULL
02551   };
02552   gtk_show_about_dialog
02553     (window->window,
02554      "program_name", "Calibrator",
02555      "comments",
02556      gettext ("A software to perform calibrations/optimizations of empirical "
02557               "parameters"),
02558      "authors", authors,
02559      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
02560      "version", "1.0.6",
02561      "copyright", "Copyright 2012-2015 Javier Burguete Tolosa",
02562      "logo", window->logo,
02563      "website", "https://github.com/jburguete/calibrator",
02564      "license-type", GTK_LICENSE_BSD, NULL);
02565 }
02566
02572 int
02573 window_get_algorithm ()
02574 {
02575   unsigned int i;
02576   for (i = 0; i < NALGORITHMS; ++i)
02577     if (gtk_toggle_button_get_active
02578         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02579       break;
02580   return i;
02581 }
02582
02587 void
02588 window_update ()
02589 {
02590   unsigned int i;
02591   gtk_widget_set_sensitive
02592     (GTK_WIDGET (window->button_evaluator),
02593      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02594                                    (window->check_evaluator)));
02595   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
02596   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
02597   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
02598   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
02599   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
02600   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
```

```
02601   gtk_widget_hide (GTK_WIDGET (window->label_bests));
02602   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
02603   gtk_widget_hide (GTK_WIDGET (window->label_population));
02604   gtk_widget_hide (GTK_WIDGET (window->spin_population));
02605   gtk_widget_hide (GTK_WIDGET (window->label_generations));
02606   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
02607   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
02608   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
02609   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
02610   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
02611   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
02612   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
02613   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
02614   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
02615   gtk_widget_hide (GTK_WIDGET (window->label_bits));
02616   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
02617   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
02618   switch (window_get_algorithm ())
02619     {
02620     case ALGORITHM_MONTE_CARLO:
02621       gtk_widget_show (GTK_WIDGET (window->label_simulations));
02622       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
02623       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02624       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02625       if (i > 1)
02626         {
02627           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02628           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02629           gtk_widget_show (GTK_WIDGET (window->label_bests));
02630           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02631         }
02632       break;
02633     case ALGORITHM_SWEEP:
02634       gtk_widget_show (GTK_WIDGET (window->label_iterations));
02635       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
02636       if (i > 1)
02637         {
02638           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
02639           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
02640           gtk_widget_show (GTK_WIDGET (window->label_bests));
02641           gtk_widget_show (GTK_WIDGET (window->spin_bests));
02642         }
02643       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
02644       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
02645       break;
02646     default:
02647       gtk_widget_show (GTK_WIDGET (window->label_population));
02648       gtk_widget_show (GTK_WIDGET (window->spin_population));
02649       gtk_widget_show (GTK_WIDGET (window->label_generations));
02650       gtk_widget_show (GTK_WIDGET (window->spin_generations));
02651       gtk_widget_show (GTK_WIDGET (window->label_mutation));
02652       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
02653       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
02654       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
02655       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
02656       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
02657       gtk_widget_show (GTK_WIDGET (window->label_bits));
02658       gtk_widget_show (GTK_WIDGET (window->spin_bits));
02659     }
02660   gtk_widget_set_sensitive
02661     (GTK_WIDGET (window->button_remove_experiment), input->
      nexperiments > 1);
02662   gtk_widget_set_sensitive
02663     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
02664   for (i = 0; i < input->ninputs; ++i)
02665     {
02666       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02667       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02668       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
02669       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
02670       g_signal_handler_block
02671         (window->check_template[i], window->id_template[i]);
02672       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02673       gtk_toggle_button_set_active
02674         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
02675       g_signal_handler_unblock
02676         (window->button_template[i], window->id_input[i]);
02677       g_signal_handler_unblock
02678         (window->check_template[i], window->id_template[i]);
02679     }
02680   if (i > 0)
02681     {
02682       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
02683       gtk_widget_set_sensitive
02684         (GTK_WIDGET (window->button_template[i - 1]),
```

```
02685           gtk_toggle_button_get_active
02686           GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
02687       }
02688     if (i < MAX_NINPUTS)
02689       {
02690         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
02691         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
02692         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
02693         gtk_widget_set_sensitive
02694           (GTK_WIDGET (window->button_template[i]),
02695            gtk_toggle_button_get_active
02696            GTK_TOGGLE_BUTTON (window->check_template[i]));
02697         g_signal_handler_block
02698           (window->check_template[i], window->id_template[i]);
02699         g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
02700         gtk_toggle_button_set_active
02701           (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
02702         g_signal_handler_unblock
02703           (window->button_template[i], window->id_input[i]);
02704         g_signal_handler_unblock
02705           (window->check_template[i], window->id_template[i]);
02706       }
02707     while (++i < MAX_NINPUTS)
02708       {
02709         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
02710         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
02711       }
02712     gtk_widget_set_sensitive
02713       (GTK_WIDGET (window->spin_minabs),
02714        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
02715     gtk_widget_set_sensitive
02716       (GTK_WIDGET (window->spin_maxabs),
02717        gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
02718 }
02719
02724 void
02725 window_set_algorithm ()
02726 {
02727   int i;
02728 #if DEBUG
02729   fprintf (stderr, "window_set_algorithm: start\n");
02730 #endif
02731   i = window_get_algorithm ();
02732   switch (i)
02733     {
02734     case ALGORITHM_SWEEP:
02735       input->nsweeps = (unsigned int *) g_realloc
02736         (input->nsweeps, input->nvariables * sizeof (unsigned int));
02737       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02738       if (i < 0)
02739         i = 0;
02740       gtk_spin_button_set_value (window->spin_sweeps,
02741                                  (gdouble) input->nsweeps[i]);
02742       break;
02743     case ALGORITHM_GENETIC:
02744       input->nbits = (unsigned int *) g_realloc
02745         (input->nbits, input->nvariables * sizeof (unsigned int));
02746       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02747       if (i < 0)
02748         i = 0;
02749       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
      nbits[i]);
02750     }
02751   window_update ();
02752 #if DEBUG
02753   fprintf (stderr, "window_set_algorithm: end\n");
02754 #endif
02755 }
02756
02761 void
02762 window_set_experiment ()
02763 {
02764   unsigned int i, j;
02765   char *buffer1, *buffer2;
02766 #if DEBUG
02767   fprintf (stderr, "window_set_experiment: start\n");
02768 #endif
02769   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02770   gtk_spin_button_set_value (window->spin_weight, input->weight[i]);
02771   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
02772   buffer2 = g_build_filename (input->directory, buffer1, NULL);
02773   g_free (buffer1);
02774   g_signal_handler_block
02775     (window->button_experiment, window->id_experiment_name);
02776   gtk_file_chooser_set_filename
02777     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
```

```
02778   g_signal_handler_unblock
02779     (window->button_experiment, window->id_experiment_name);
02780   g_free (buffer2);
02781   for (j = 0; j < input->ninputs; ++j)
02782     {
02783       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02784       buffer2
02785         = g_build_filename (input->directory, input->template[j][i], NULL);
02786       gtk_file_chooser_set_filename
02787         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
02788       g_free (buffer2);
02789       g_signal_handler_unblock
02790         (window->button_template[j], window->id_input[j]);
02791     }
02792 #if DEBUG
02793   fprintf (stderr, "window_set_experiment: end\n");
02794 #endif
02795 }
02796
02801 void
02802 window_remove_experiment ()
02803 {
02804   unsigned int i, j;
02805   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02806   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02807   gtk_combo_box_text_remove (window->combo_experiment, i);
02808   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02809   xmlFree (input->experiment[i]);
02810   --input->nexperiments;
02811   for (j = i; j < input->nexperiments; ++j)
02812     {
02813       input->experiment[j] = input->experiment[j + 1];
02814       input->weight[j] = input->weight[j + 1];
02815     }
02816   j = input->nexperiments - 1;
02817   if (i > j)
02818     i = j;
02819   for (j = 0; j < input->ninputs; ++j)
02820     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02821   g_signal_handler_block
02822     (window->button_experiment, window->id_experiment_name);
02823   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02824   g_signal_handler_unblock
02825     (window->button_experiment, window->id_experiment_name);
02826   for (j = 0; j < input->ninputs; ++j)
02827     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02828   window_update ();
02829 }
02830
02835 void
02836 window_add_experiment ()
02837 {
02838   unsigned int i, j;
02839   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02840   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02841   gtk_combo_box_text_insert_text
02842     (window->combo_experiment, i, input->experiment[i]);
02843   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02844   input->experiment = (char **) g_realloc
02845     (input->experiment, (input->nexperiments + 1) * sizeof (char *));
02846   input->weight = (double *) g_realloc
02847     (input->weight, (input->nexperiments + 1) * sizeof (double));
02848   for (j = input->nexperiments - 1; j > i; --j)
02849     {
02850       input->experiment[j + 1] = input->experiment[j];
02851       input->weight[j + 1] = input->weight[j];
02852     }
02853   input->experiment[j + 1]
02854     = (char *) xmlStrdup ((xmlChar *) input->experiment[j]);
02855   input->weight[j + 1] = input->weight[j];
02856   ++input->nexperiments;
02857   for (j = 0; j < input->ninputs; ++j)
02858     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
02859   g_signal_handler_block
02860     (window->button_experiment, window->id_experiment_name);
02861   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
02862   g_signal_handler_unblock
02863     (window->button_experiment, window->id_experiment_name);
02864   for (j = 0; j < input->ninputs; ++j)
```

```
02865     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
02866   window_update ();
02867 }
02868
02873 void
02874 window_name_experiment ()
02875 {
02876   unsigned int i;
02877   char *buffer;
02878   GFile *file1, *file2;
02879 #if DEBUG
02880   fprintf (stderr, "window_name_experiment: start\n");
02881 #endif
02882   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02883   file1
02884     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
02885   file2 = g_file_new_for_path (input->directory);
02886   buffer = g_file_get_relative_path (file2, file1);
02887   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
02888   gtk_combo_box_text_remove (window->combo_experiment, i);
02889   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
02890   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
02891   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
02892   g_free (buffer);
02893   g_object_unref (file2);
02894   g_object_unref (file1);
02895 #if DEBUG
02896   fprintf (stderr, "window_name_experiment: end\n");
02897 #endif
02898 }
02899
02904 void
02905 window_weight_experiment ()
02906 {
02907   unsigned int i;
02908 #if DEBUG
02909   fprintf (stderr, "window_weight_experiment: start\n");
02910 #endif
02911   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02912   input->weight[i] = gtk_spin_button_get_value (window->spin_weight);
02913 #if DEBUG
02914   fprintf (stderr, "window_weight_experiment: end\n");
02915 #endif
02916 }
02917
02923 void
02924 window_inputs_experiment ()
02925 {
02926   unsigned int j;
02927 #if DEBUG
02928   fprintf (stderr, "window_inputs_experiment: start\n");
02929 #endif
02930   j = input->ninputs - 1;
02931   if (j
02932       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02933                                         (window->check_template[j])))
02934     --input->ninputs;
02935   if (input->ninputs < MAX_NINPUTS
02936       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
02937                                        (window->check_template[j])))
02938     {
02939       ++input->ninputs;
02940       for (j = 0; j < input->ninputs; ++j)
02941         {
02942           input->template[j] = (char **)
02943             g_realloc (input->template[j], input->nvariables * sizeof (char *));
02944         }
02945     }
02946   window_update ();
02947 #if DEBUG
02948   fprintf (stderr, "window_inputs_experiment: end\n");
02949 #endif
02950 }
02951
02959 void
02960 window_template_experiment (void *data)
02961 {
02962   unsigned int i, j;
02963   char *buffer;
02964   GFile *file1, *file2;
02965 #if DEBUG
02966   fprintf (stderr, "window_template_experiment: start\n");
02967 #endif
02968   i = (size_t) data;
```

```
02969   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02970   file1
02971     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02972   file2 = g_file_new_for_path (input->directory);
02973   buffer = g_file_get_relative_path (file2, file1);
02974   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02975   g_free (buffer);
02976   g_object_unref (file2);
02977   g_object_unref (file1);
02978 #if DEBUG
02979   fprintf (stderr, "window_template_experiment: end\n");
02980 #endif
02981 }
02982
02987 void
02988 window_set_variable ()
02989 {
02990   unsigned int i;
02991 #if DEBUG
02992   fprintf (stderr, "window_set_variable: start\n");
02993 #endif
02994   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
02995   g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
02996   gtk_entry_set_text (window->entry_variable, input->label[i]);
02997   g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
02998   gtk_spin_button_set_value (window->spin_min, input->rangemin[i]);
02999   gtk_spin_button_set_value (window->spin_max, input->rangemax[i]);
03000   if (input->rangeminabs[i] != -G_MAXDOUBLE)
03001     {
03002       gtk_spin_button_set_value (window->spin_minabs, input->
    rangeminabs[i]);
03003       gtk_toggle_button_set_active
03004         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
03005     }
03006   else
03007     {
03008       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
03009       gtk_toggle_button_set_active
03010         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
03011     }
03012   if (input->rangemaxabs[i] != G_MAXDOUBLE)
03013     {
03014       gtk_spin_button_set_value (window->spin_maxabs, input->
    rangemaxabs[i]);
03015       gtk_toggle_button_set_active
03016         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
03017     }
03018   else
03019     {
03020       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
03021       gtk_toggle_button_set_active
03022         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
03023     }
03024   gtk_spin_button_set_value (window->spin_precision, input->
    precision[i]);
03025 #if DEBUG
03026   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
03027            input->precision[i]);
03028 #endif
03029   switch (window_get_algorithm ())
03030     {
03031     case ALGORITHM_SWEEP:
03032       gtk_spin_button_set_value (window->spin_sweeps,
03033                                  (gdouble) input->nsweeps[i]);
03034 #if DEBUG
03035       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
03036                input->nsweeps[i]);
03037 #endif
03038       break;
03039     case ALGORITHM_GENETIC:
03040       gtk_spin_button_set_value (window->spin_bits, (gdouble) input->
    nbits[i]);
03041 #if DEBUG
03042       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
03043                input->nbits[i]);
03044 #endif
03045       break;
03046     }
03047   window_update ();
03048 #if DEBUG
03049   fprintf (stderr, "window_set_variable: end\n");
03050 #endif
03051 }
03052
03057 void
```

```
03058 window_remove_variable ()
03059 {
03060   unsigned int i, j;
03061   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03062   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03063   gtk_combo_box_text_remove (window->combo_variable, i);
03064   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03065   xmlFree (input->label[i]);
03066   --input->nvariables;
03067   for (j = i; j < input->nvariables; ++j)
03068     {
03069       input->label[j] = input->label[j + 1];
03070       input->rangemin[j] = input->rangemin[j + 1];
03071       input->rangemax[j] = input->rangemax[j + 1];
03072       input->rangeminabs[j] = input->rangeminabs[j + 1];
03073       input->rangemaxabs[j] = input->rangemaxabs[j + 1];
03074       input->precision[j] = input->precision[j + 1];
03075       switch (window_get_algorithm ())
03076         {
03077         case ALGORITHM_SWEEP:
03078           input->nsweeps[j] = input->nsweeps[j + 1];
03079           break;
03080         case ALGORITHM_GENETIC:
03081           input->nbits[j] = input->nbits[j + 1];
03082         }
03083     }
03084   j = input->nvariables - 1;
03085   if (i > j)
03086     i = j;
03087   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03088   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03089   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03090   window_update ();
03091 }
03092
03097 void
03098 window_add_variable ()
03099 {
03100   unsigned int i, j;
03101 #if DEBUG
03102   fprintf (stderr, "window_add_variable: start\n");
03103 #endif
03104   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03105   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03106   gtk_combo_box_text_insert_text (window->combo_variable, i, input->
      label[i]);
03107   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03108   input->label = (char **) g_realloc
03109     (input->label, (input->nvariables + 1) * sizeof (char *));
03110   input->rangemin = (double *) g_realloc
03111     (input->rangemin, (input->nvariables + 1) * sizeof (double));
03112   input->rangemax = (double *) g_realloc
03113     (input->rangemax, (input->nvariables + 1) * sizeof (double));
03114   input->rangeminabs = (double *) g_realloc
03115     (input->rangeminabs, (input->nvariables + 1) * sizeof (double));
03116   input->rangemaxabs = (double *) g_realloc
03117     (input->rangemaxabs, (input->nvariables + 1) * sizeof (double));
03118   input->precision = (unsigned int *) g_realloc
03119     (input->precision, (input->nvariables + 1) * sizeof (unsigned int));
03120   for (j = input->nvariables - 1; j > i; --j)
03121     {
03122       input->label[j + 1] = input->label[j];
03123       input->rangemin[j + 1] = input->rangemin[j];
03124       input->rangemax[j + 1] = input->rangemax[j];
03125       input->rangeminabs[j + 1] = input->rangeminabs[j];
03126       input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03127       input->precision[j + 1] = input->precision[j];
03128     }
03129   input->label[j + 1] = (char *) xmlStrdup ((xmlChar *) input->label[j]);
03130   input->rangemin[j + 1] = input->rangemin[j];
03131   input->rangemax[j + 1] = input->rangemax[j];
03132   input->rangeminabs[j + 1] = input->rangeminabs[j];
03133   input->rangemaxabs[j + 1] = input->rangemaxabs[j];
03134   input->precision[j + 1] = input->precision[j];
03135   switch (window_get_algorithm ())
03136     {
03137     case ALGORITHM_SWEEP:
03138       input->nsweeps = (unsigned int *) g_realloc
03139         (input->nsweeps, (input->nvariables + 1) * sizeof (unsigned int));
03140       for (j = input->nvariables - 1; j > i; --j)
03141         input->nsweeps[j + 1] = input->nsweeps[j];
```

```
03142       input->nsweeps[j + 1] = input->nsweeps[j];
03143       break;
03144     case ALGORITHM_GENETIC:
03145       input->nbits = (unsigned int *) g_realloc
03146         (input->nbits, (input->nvariables + 1) * sizeof (unsigned int));
03147       for (j = input->nvariables - 1; j > i; --j)
03148         input->nbits[j + 1] = input->nbits[j];
03149       input->nbits[j + 1] = input->nbits[j];
03150     }
03151   ++input->nvariables;
03152   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03153   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
03154   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03155   window_update ();
03156 #if DEBUG
03157   fprintf (stderr, "window_add_variable: end\n");
03158 #endif
03159 }
03160
03165 void
03166 window_label_variable ()
03167 {
03168   unsigned int i;
03169   const char *buffer;
03170 #if DEBUG
03171   fprintf (stderr, "window_label_variable: start\n");
03172 #endif
03173   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03174   buffer = gtk_entry_get_text (window->entry_variable);
03175   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03176   gtk_combo_box_text_remove (window->combo_variable, i);
03177   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
03178   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
03179   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03180 #if DEBUG
03181   fprintf (stderr, "window_label_variable: end\n");
03182 #endif
03183 }
03184
03189 void
03190 window_precision_variable ()
03191 {
03192   unsigned int i;
03193 #if DEBUG
03194   fprintf (stderr, "window_precision_variable: start\n");
03195 #endif
03196   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03197   input->precision[i]
03198     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
03199   gtk_spin_button_set_digits (window->spin_min, input->precision[i]);
03200   gtk_spin_button_set_digits (window->spin_max, input->precision[i]);
03201   gtk_spin_button_set_digits (window->spin_minabs, input->precision[i]);
03202   gtk_spin_button_set_digits (window->spin_maxabs, input->precision[i]);
03203 #if DEBUG
03204   fprintf (stderr, "window_precision_variable: end\n");
03205 #endif
03206 }
03207
03212 void
03213 window_rangemin_variable ()
03214 {
03215   unsigned int i;
03216 #if DEBUG
03217   fprintf (stderr, "window_rangemin_variable: start\n");
03218 #endif
03219   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03220   input->rangemin[i] = gtk_spin_button_get_value (window->spin_min);
03221 #if DEBUG
03222   fprintf (stderr, "window_rangemin_variable: end\n");
03223 #endif
03224 }
03225
03230 void
03231 window_rangemax_variable ()
03232 {
03233   unsigned int i;
03234 #if DEBUG
03235   fprintf (stderr, "window_rangemax_variable: start\n");
03236 #endif
03237   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03238   input->rangemax[i] = gtk_spin_button_get_value (window->spin_max);
03239 #if DEBUG
03240   fprintf (stderr, "window_rangemax_variable: end\n");
```

```
03241 #endif
03242 }
03243
03248 void
03249 window_rangeminabs_variable ()
03250 {
03251   unsigned int i;
03252 #if DEBUG
03253   fprintf (stderr, "window_rangeminabs_variable: start\n");
03254 #endif
03255   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03256   input->rangeminabs[i] = gtk_spin_button_get_value (window->
      spin_minabs);
03257 #if DEBUG
03258   fprintf (stderr, "window_rangeminabs_variable: end\n");
03259 #endif
03260 }
03261
03266 void
03267 window_rangemaxabs_variable ()
03268 {
03269   unsigned int i;
03270 #if DEBUG
03271   fprintf (stderr, "window_rangemaxabs_variable: start\n");
03272 #endif
03273   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03274   input->rangemaxabs[i] = gtk_spin_button_get_value (window->
      spin_maxabs);
03275 #if DEBUG
03276   fprintf (stderr, "window_rangemaxabs_variable: end\n");
03277 #endif
03278 }
03279
03284 void
03285 window_update_variable ()
03286 {
03287   int i;
03288 #if DEBUG
03289   fprintf (stderr, "window_update_variable: start\n");
03290 #endif
03291   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
03292   if (i < 0)
03293     i = 0;
03294   switch (window_get_algorithm ())
03295     {
03296     case ALGORITHM_SWEEP:
03297       input->nsweeps[i]
03298         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
03299 #if DEBUG
03300       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
03301                input->nsweeps[i]);
03302 #endif
03303       break;
03304     case ALGORITHM_GENETIC:
03305       input->nbits[i] = gtk_spin_button_get_value_as_int (window->spin_bits);
03306 #if DEBUG
03307       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
03308                input->nbits[i]);
03309 #endif
03310     }
03311 #if DEBUG
03312   fprintf (stderr, "window_update_variable: end\n");
03313 #endif
03314 }
03315
03323 int
03324 window_read (char *filename)
03325 {
03326   unsigned int i;
03327   char *buffer;
03328 #if DEBUG
03329   fprintf (stderr, "window_read: start\n");
03330 #endif
03331
03332   // Reading new input file
03333   input_free ();
03334   if (!input_open (filename))
03335     return 0;
03336
03337   // Setting GTK+ widgets data
03338   gtk_entry_set_text (window->entry_result, input->result);
03339   gtk_entry_set_text (window->entry_variables, input->variables);
03340   buffer = g_build_filename (input->directory, input->simulator, NULL);
03341   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03342                                  (window->button_simulator), buffer);
03343   g_free (buffer);
03344   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
```

```
03345                                 (size_t) input->evaluator);
03346   if (input->evaluator)
03347     {
03348       buffer = g_build_filename (input->directory, input->evaluator, NULL);
03349       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03350                                      (window->button_evaluator), buffer);
03351       g_free (buffer);
03352     }
03353   gtk_toggle_button_set_active
03354     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03355   switch (input->algorithm)
03356     {
03357     case ALGORITHM_MONTE_CARLO:
03358       gtk_spin_button_set_value (window->spin_simulations,
03359                                  (gdouble) input->nsimulations);
03360     case ALGORITHM_SWEEP:
03361       gtk_spin_button_set_value (window->spin_iterations,
03362                                  (gdouble) input->niterations);
03363       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
03364       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
03365       break;
03366     default:
03367       gtk_spin_button_set_value (window->spin_population,
03368                                  (gdouble) input->nsimulations);
03369       gtk_spin_button_set_value (window->spin_generations,
03370                                  (gdouble) input->niterations);
03371       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03372       gtk_spin_button_set_value (window->spin_reproduction,
03373                                  input->reproduction_ratio);
03374       gtk_spin_button_set_value (window->spin_adaptation,
03375                                  input->adaptation_ratio);
03376     }
03377   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03378   g_signal_handler_block (window->button_experiment,
03379                           window->id_experiment_name);
03380   gtk_combo_box_text_remove_all (window->combo_experiment);
03381   for (i = 0; i < input->nexperiments; ++i)
03382     gtk_combo_box_text_append_text (window->combo_experiment,
03383                                     input->experiment[i]);
03384   g_signal_handler_unblock
03385     (window->button_experiment, window->id_experiment_name);
03386   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
03387   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03388   g_signal_handler_block (window->combo_variable, window->
      id_variable);
03389   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03390   gtk_combo_box_text_remove_all (window->combo_variable);
03391   for (i = 0; i < input->nvariables; ++i)
03392     gtk_combo_box_text_append_text (window->combo_variable, input->
      label[i]);
03393   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03394   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03395   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03396   window_set_variable ();
03397   window_update ();
03398
03399 #if DEBUG
03400   fprintf (stderr, "window_read: end\n");
03401 #endif
03402   return 1;
03403 }
03404
03409 void
03410 window_open ()
03411 {
03412   char *buffer, *directory, *name;
03413   GtkFileChooserDialog *dlg;
03414
03415 #if DEBUG
03416   fprintf (stderr, "window_open: start\n");
03417 #endif
03418
03419   // Saving a backup of the current input file
03420   directory = g_strdup (input->directory);
03421   name = g_strdup (input->name);
03422
03423   // Opening dialog
03424   dlg = (GtkFileChooserDialog *)
```

```
03425     gtk_file_chooser_dialog_new (gettext ("Open input file"),
03426                                  window->window,
03427                                  GTK_FILE_CHOOSER_ACTION_OPEN,
03428                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
03429                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
03430   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
03431     {
03432
03433       // Traying to open the input file
03434       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
03435       if (!window_read (buffer))
03436         {
03437 #if DEBUG
03438           fprintf (stderr, "window_open: error reading input file\n");
03439 #endif
03440
03441           // Reading backup file on error
03442           buffer = g_build_filename (directory, name, NULL);
03443           if (!input_open (buffer))
03444             {
03445
03446               // Closing on backup file reading error
03447 #if DEBUG
03448               fprintf (stderr, "window_read: error reading backup file\n");
03449 #endif
03450               g_free (buffer);
03451               g_free (name);
03452               g_free (directory);
03453 #if DEBUG
03454               fprintf (stderr, "window_open: end\n");
03455 #endif
03456               gtk_main_quit ();
03457             }
03458           g_free (buffer);
03459         }
03460       else
03461         break;
03462     }
03463
03464   // Freeing and closing
03465   g_free (name);
03466   g_free (directory);
03467   gtk_widget_destroy (GTK_WIDGET (dlg));
03468 #if DEBUG
03469   fprintf (stderr, "window_open: end\n");
03470 #endif
03471 }
03472
03477 void
03478 window_new ()
03479 {
03480   unsigned int i;
03481   char *buffer, *buffer2, buffer3[64];
03482   GtkViewport *viewport;
03483   char *label_algorithm[NALGORITHMS] = {
03484     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
03485   };
03486   char *tip_algorithm[NALGORITHMS] = {
03487     gettext ("Monte-Carlo brute force algorithm"),
03488     gettext ("Sweep brute force algorithm"),
03489     gettext ("Genetic algorithm")
03490   };
03491
03492   // Creating the window
03493   window->window = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
03494
03495   // Finish when closing the window
03496   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
03497
03498   // Setting the window title
03499   gtk_window_set_title (window->window, PROGRAM_INTERFACE);
03500
03501   // Creating the open button
03502   window->button_open = (GtkToolButton *) gtk_tool_button_new
03503     (gtk_image_new_from_icon_name ("document-open",
03504                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03505      gettext ("Open"));
03506   g_signal_connect (window->button_open, "clicked", window_open, NULL);
03507
03508   // Creating the save button
03509   window->button_save = (GtkToolButton *) gtk_tool_button_new
03510     (gtk_image_new_from_icon_name ("document-save",
03511                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03512      gettext ("Save"));
03513   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
03514                     NULL);
```

```
03515
03516   // Creating the run button
03517   window->button_run = (GtkToolButton *) gtk_tool_button_new
03518     (gtk_image_new_from_icon_name ("system-run",
03519                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03520      gettext ("Run"));
03521   g_signal_connect (window->button_run, "clicked", window_run, NULL);
03522
03523   // Creating the options button
03524   window->button_options = (GtkToolButton *) gtk_tool_button_new
03525     (gtk_image_new_from_icon_name ("preferences-system",
03526                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03527      gettext ("Options"));
03528   g_signal_connect (window->button_options, "clicked", options_new, NULL);
03529
03530   // Creating the help button
03531   window->button_help = (GtkToolButton *) gtk_tool_button_new
03532     (gtk_image_new_from_icon_name ("help-browser",
03533                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03534      gettext ("Help"));
03535   g_signal_connect (window->button_help, "clicked", window_help, NULL);
03536
03537   // Creating the about button
03538   window->button_about = (GtkToolButton *) gtk_tool_button_new
03539     (gtk_image_new_from_icon_name ("help-about",
03540                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03541      gettext ("About"));
03542   g_signal_connect (window->button_about, "clicked", window_about, NULL);
03543
03544   // Creating the exit button
03545   window->button_exit = (GtkToolButton *) gtk_tool_button_new
03546     (gtk_image_new_from_icon_name ("application-exit",
03547                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
03548      gettext ("Exit"));
03549   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
03550
03551   // Creating the buttons bar
03552   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
03553   gtk_toolbar_insert
03554     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
03555   gtk_toolbar_insert
03556     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
03557   gtk_toolbar_insert
03558     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
03559   gtk_toolbar_insert
03560     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
03561   gtk_toolbar_insert
03562     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
03563   gtk_toolbar_insert
03564     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
03565   gtk_toolbar_insert
03566     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
03567   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
03568
03569   // Creating the simulator program label and entry
03570   window->label_simulator
03571     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
03572   window->button_simulator = (GtkFileChooserButton *)
03573     gtk_file_chooser_button_new (gettext ("Simulator program"),
03574                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03575   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
03576                                gettext ("Simulator program executable file"));
03577
03578   // Creating the evaluator program label and entry
03579   window->check_evaluator = (GtkCheckButton *)
03580     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
03581   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
03582   window->button_evaluator = (GtkFileChooserButton *)
03583     gtk_file_chooser_button_new (gettext ("Evaluator program"),
03584                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03585   gtk_widget_set_tooltip_text
03586     (GTK_WIDGET (window->button_evaluator),
03587      gettext ("Optional evaluator program executable file"));
03588
03589   // Creating the results files labels and entries
03590   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
03591   window->entry_result = (GtkEntry *) gtk_entry_new ();
03592   gtk_widget_set_tooltip_text
03593     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
03594   window->label_variables
03595     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
03596   window->entry_variables = (GtkEntry *) gtk_entry_new ();
03597   gtk_widget_set_tooltip_text
03598     (GTK_WIDGET (window->entry_variables),
03599      gettext ("All simulated results file"));
03600
```

```
03601   // Creating the files grid and attaching widgets
03602   window->grid_files = (GtkGrid *) gtk_grid_new ();
03603   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
03604                    0, 0, 1, 1);
03605   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
03606                    1, 0, 1, 1);
03607   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
03608                    2, 0, 1, 1);
03609   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
03610                    3, 0, 1, 1);
03611   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
03612                    0, 1, 1, 1);
03613   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
03614                    1, 1, 1, 1);
03615   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
03616                    2, 1, 1, 1);
03617   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
03618                    3, 1, 1, 1);
03619
03620   // Creating the algorithm properties
03621   window->label_simulations = (GtkLabel *) gtk_label_new
03622     (gettext ("Simulations number"));
03623   window->spin_simulations
03624     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03625   gtk_widget_set_tooltip_text
03626     (GTK_WIDGET (window->spin_simulations),
03627      gettext ("Number of simulations to perform for each iteration"));
03628   window->label_iterations = (GtkLabel *)
03629     gtk_label_new (gettext ("Iterations number"));
03630   window->spin_iterations
03631     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03632   gtk_widget_set_tooltip_text
03633     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
03634   g_signal_connect
03635     (window->spin_iterations, "value-changed", window_update, NULL);
03636   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
03637   window->spin_tolerance
03638     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03639   gtk_widget_set_tooltip_text
03640     (GTK_WIDGET (window->spin_tolerance),
03641      gettext ("Tolerance to set the variable interval on the next iteration"));
03642   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
03643   window->spin_bests
03644     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03645   gtk_widget_set_tooltip_text
03646     (GTK_WIDGET (window->spin_bests),
03647      gettext ("Number of best simulations used to set the variable interval "
03648               "on the next iteration"));
03649   window->label_population
03650     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
03651   window->spin_population
03652     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03653   gtk_widget_set_tooltip_text
03654     (GTK_WIDGET (window->spin_population),
03655      gettext ("Number of population for the genetic algorithm"));
03656   window->label_generations
03657     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
03658   window->spin_generations
03659     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
03660   gtk_widget_set_tooltip_text
03661     (GTK_WIDGET (window->spin_generations),
03662      gettext ("Number of generations for the genetic algorithm"));
03663   window->label_mutation
03664     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
03665   window->spin_mutation
03666     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03667   gtk_widget_set_tooltip_text
03668     (GTK_WIDGET (window->spin_mutation),
03669      gettext ("Ratio of mutation for the genetic algorithm"));
03670   window->label_reproduction
03671     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
03672   window->spin_reproduction
03673     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03674   gtk_widget_set_tooltip_text
03675     (GTK_WIDGET (window->spin_reproduction),
03676      gettext ("Ratio of reproduction for the genetic algorithm"));
03677   window->label_adaptation
03678     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
03679   window->spin_adaptation
```

```
03680     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03681   gtk_widget_set_tooltip_text
03682     (GTK_WIDGET (window->spin_adaptation),
03683      gettext ("Ratio of adaptation for the genetic algorithm"));
03684
03685   // Creating the array of algorithms
03686   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
03687   window->button_algorithm[0] = (GtkRadioButton *)
03688     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
03689   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
03690                                tip_algorithm[0]);
03691   gtk_grid_attach (window->grid_algorithm,
03692                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
03693   g_signal_connect (window->button_algorithm[0], "clicked",
03694                     window_set_algorithm, NULL);
03695   for (i = 0; ++i < NALGORITHMS;)
03696     {
03697       window->button_algorithm[i] = (GtkRadioButton *)
03698         gtk_radio_button_new_with_mnemonic
03699         (gtk_radio_button_get_group (window->button_algorithm[0]),
03700          label_algorithm[i]);
03701       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
03702                                    tip_algorithm[i]);
03703       gtk_grid_attach (window->grid_algorithm,
03704                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
03705       g_signal_connect (window->button_algorithm[i], "clicked",
03706                         window_set_algorithm, NULL);
03707     }
03708   gtk_grid_attach (window->grid_algorithm,
03709                    GTK_WIDGET (window->label_simulations), 0,
03710                    NALGORITHMS, 1, 1);
03711   gtk_grid_attach (window->grid_algorithm,
03712                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
03713   gtk_grid_attach (window->grid_algorithm,
03714                    GTK_WIDGET (window->label_iterations), 0,
03715                    NALGORITHMS + 1, 1, 1);
03716   gtk_grid_attach (window->grid_algorithm,
03717                    GTK_WIDGET (window->spin_iterations), 1,
03718                    NALGORITHMS + 1, 1, 1);
03719   gtk_grid_attach (window->grid_algorithm,
03720                    GTK_WIDGET (window->label_tolerance), 0,
03721                    NALGORITHMS + 2, 1, 1);
03722   gtk_grid_attach (window->grid_algorithm,
03723                    GTK_WIDGET (window->spin_tolerance), 1,
03724                    NALGORITHMS + 2, 1, 1);
03725   gtk_grid_attach (window->grid_algorithm,
03726                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
03727   gtk_grid_attach (window->grid_algorithm,
03728                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
03729   gtk_grid_attach (window->grid_algorithm,
03730                    GTK_WIDGET (window->label_population), 0,
03731                    NALGORITHMS + 4, 1, 1);
03732   gtk_grid_attach (window->grid_algorithm,
03733                    GTK_WIDGET (window->spin_population), 1,
03734                    NALGORITHMS + 4, 1, 1);
03735   gtk_grid_attach (window->grid_algorithm,
03736                    GTK_WIDGET (window->label_generations), 0,
03737                    NALGORITHMS + 5, 1, 1);
03738   gtk_grid_attach (window->grid_algorithm,
03739                    GTK_WIDGET (window->spin_generations), 1,
03740                    NALGORITHMS + 5, 1, 1);
03741   gtk_grid_attach (window->grid_algorithm,
03742                    GTK_WIDGET (window->label_mutation), 0,
03743                    NALGORITHMS + 6, 1, 1);
03744   gtk_grid_attach (window->grid_algorithm,
03745                    GTK_WIDGET (window->spin_mutation), 1,
03746                    NALGORITHMS + 6, 1, 1);
03747   gtk_grid_attach (window->grid_algorithm,
03748                    GTK_WIDGET (window->label_reproduction), 0,
03749                    NALGORITHMS + 7, 1, 1);
03750   gtk_grid_attach (window->grid_algorithm,
03751                    GTK_WIDGET (window->spin_reproduction), 1,
03752                    NALGORITHMS + 7, 1, 1);
03753   gtk_grid_attach (window->grid_algorithm,
03754                    GTK_WIDGET (window->label_adaptation), 0,
03755                    NALGORITHMS + 8, 1, 1);
03756   gtk_grid_attach (window->grid_algorithm,
03757                    GTK_WIDGET (window->spin_adaptation), 1,
03758                    NALGORITHMS + 8, 1, 1);
03759   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
03760   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
03761                      GTK_WIDGET (window->grid_algorithm));
03762
03763   // Creating the variable widgets
03764   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
03765   gtk_widget_set_tooltip_text
03766     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
```

```
03767   window->id_variable = g_signal_connect
03768     (window->combo_variable, "changed", window_set_variable, NULL);
03769   window->button_add_variable
03770     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03771                                                    GTK_ICON_SIZE_BUTTON);
03772   g_signal_connect
03773     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
03774   gtk_widget_set_tooltip_text
03775     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
03776   window->button_remove_variable
03777     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03778                                                    GTK_ICON_SIZE_BUTTON);
03779   g_signal_connect
03780     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
03781   gtk_widget_set_tooltip_text
03782     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
03783   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
03784   window->entry_variable = (GtkEntry *) gtk_entry_new ();
03785   gtk_widget_set_tooltip_text
03786     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
03787   window->id_variable_label = g_signal_connect
03788     (window->entry_variable, "changed", window_label_variable, NULL);
03789   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
03790   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
03791     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03792   gtk_widget_set_tooltip_text
03793     (GTK_WIDGET (window->spin_min),
03794      gettext ("Minimum initial value of the variable"));
03795   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03796   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_min));
03797   window->scrolled_min
03798     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03799   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
03800                      GTK_WIDGET (viewport));
03801   g_signal_connect (window->spin_min, "value-changed",
03802                     window_rangemin_variable, NULL);
03803   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
03804   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
03805     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03806   gtk_widget_set_tooltip_text
03807     (GTK_WIDGET (window->spin_max),
03808      gettext ("Maximum initial value of the variable"));
03809   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03810   gtk_container_add (GTK_CONTAINER (viewport), GTK_WIDGET (window->spin_max));
03811   window->scrolled_max
03812     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03813   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
03814                      GTK_WIDGET (viewport));
03815   g_signal_connect (window->spin_max, "value-changed",
03816                     window_rangemax_variable, NULL);
03817   window->check_minabs = (GtkCheckButton *)
03818     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
03819   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
03820   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03821     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03822   gtk_widget_set_tooltip_text
03823     (GTK_WIDGET (window->spin_minabs),
03824      gettext ("Minimum allowed value of the variable"));
03825   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03826   gtk_container_add (GTK_CONTAINER (viewport),
03827                      GTK_WIDGET (window->spin_minabs));
03828   window->scrolled_minabs
03829     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03830   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
03831                      GTK_WIDGET (viewport));
03832   g_signal_connect (window->spin_minabs, "value-changed",
03833                     window_rangeminabs_variable, NULL);
03834   window->check_maxabs = (GtkCheckButton *)
03835     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
03836   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
03837   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
03838     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
03839   gtk_widget_set_tooltip_text
03840     (GTK_WIDGET (window->spin_maxabs),
03841      gettext ("Maximum allowed value of the variable"));
03842   viewport = (GtkViewport *) gtk_viewport_new (NULL, NULL);
03843   gtk_container_add (GTK_CONTAINER (viewport),
03844                      GTK_WIDGET (window->spin_maxabs));
03845   window->scrolled_maxabs
03846     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
03847   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
03848                      GTK_WIDGET (viewport));
03849   g_signal_connect (window->spin_maxabs, "value-changed",
03850                     window_rangemaxabs_variable, NULL);
03851   window->label_precision
```

```
03852     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
03853   window->spin_precision = (GtkSpinButton *)
03854     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
03855   gtk_widget_set_tooltip_text
03856     (GTK_WIDGET (window->spin_precision),
03857      gettext ("Number of precision floating point digits\n"
03858               "0 is for integer numbers"));
03859   g_signal_connect (window->spin_precision, "value-changed",
03860                     window_precision_variable, NULL);
03861   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
03862   window->spin_sweeps
03863     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
03864   gtk_widget_set_tooltip_text
03865     (GTK_WIDGET (window->spin_sweeps),
03866      gettext ("Number of steps sweeping the variable"));
03867   g_signal_connect
03868     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
03869   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
03870   window->spin_bits
03871     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
03872   gtk_widget_set_tooltip_text
03873     (GTK_WIDGET (window->spin_bits),
03874      gettext ("Number of bits to encode the variable"));
03875   g_signal_connect
03876     (window->spin_bits, "value-changed", window_update_variable, NULL);
03877   window->grid_variable = (GtkGrid *) gtk_grid_new ();
03878   gtk_grid_attach (window->grid_variable,
03879                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
03880   gtk_grid_attach (window->grid_variable,
03881                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
03882   gtk_grid_attach (window->grid_variable,
03883                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
03884   gtk_grid_attach (window->grid_variable,
03885                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
03886   gtk_grid_attach (window->grid_variable,
03887                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
03888   gtk_grid_attach (window->grid_variable,
03889                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
03890   gtk_grid_attach (window->grid_variable,
03891                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
03892   gtk_grid_attach (window->grid_variable,
03893                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
03894   gtk_grid_attach (window->grid_variable,
03895                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
03896   gtk_grid_attach (window->grid_variable,
03897                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
03898   gtk_grid_attach (window->grid_variable,
03899                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
03900   gtk_grid_attach (window->grid_variable,
03901                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
03902   gtk_grid_attach (window->grid_variable,
03903                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
03904   gtk_grid_attach (window->grid_variable,
03905                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
03906   gtk_grid_attach (window->grid_variable,
03907                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
03908   gtk_grid_attach (window->grid_variable,
03909                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
03910   gtk_grid_attach (window->grid_variable,
03911                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
03912   gtk_grid_attach (window->grid_variable,
03913                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
03914   gtk_grid_attach (window->grid_variable,
03915                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
03916   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
03917   gtk_container_add (GTK_CONTAINER (window->frame_variable),
03918                      GTK_WIDGET (window->grid_variable));
03919
03920   // Creating the experiment widgets
03921   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
03922   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
03923                                gettext ("Experiment selector"));
03924   window->id_experiment = g_signal_connect
03925     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
03926   window->button_add_experiment
03927     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
03928                                                    GTK_ICON_SIZE_BUTTON);
03929   g_signal_connect
03930     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
03931   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
03932                                gettext ("Add experiment"));
03933   window->button_remove_experiment
03934     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
03935                                                    GTK_ICON_SIZE_BUTTON);
03936   g_signal_connect (window->button_remove_experiment, "clicked",
```

```
03937                     window_remove_experiment, NULL);
03938   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
03939                                gettext ("Remove experiment"));
03940   window->label_experiment
03941     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
03942   window->button_experiment = (GtkFileChooserButton *)
03943     gtk_file_chooser_button_new (gettext ("Experimental data file"),
03944                                  GTK_FILE_CHOOSER_ACTION_OPEN);
03945   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
03946                                gettext ("Experimental data file"));
03947   window->id_experiment_name
03948     = g_signal_connect (window->button_experiment, "selection-changed",
03949                         window_name_experiment, NULL);
03950   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
03951   window->spin_weight
03952     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
03953   gtk_widget_set_tooltip_text
03954     (GTK_WIDGET (window->spin_weight),
03955      gettext ("Weight factor to build the objective function"));
03956   g_signal_connect
03957     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
03958   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
03959   gtk_grid_attach (window->grid_experiment,
03960                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
03961   gtk_grid_attach (window->grid_experiment,
03962                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
03963   gtk_grid_attach (window->grid_experiment,
03964                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
03965   gtk_grid_attach (window->grid_experiment,
03966                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
03967   gtk_grid_attach (window->grid_experiment,
03968                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
03969   gtk_grid_attach (window->grid_experiment,
03970                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
03971   gtk_grid_attach (window->grid_experiment,
03972                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
03973   for (i = 0; i < MAX_NINPUTS; ++i)
03974     {
03975       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
03976       window->check_template[i] = (GtkCheckButton *)
03977         gtk_check_button_new_with_label (buffer3);
03978       window->id_template[i]
03979         = g_signal_connect (window->check_template[i], "toggled",
03980                             window_inputs_experiment, NULL);
03981       gtk_grid_attach (window->grid_experiment,
03982                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
03983       window->button_template[i] = (GtkFileChooserButton *)
03984         gtk_file_chooser_button_new (gettext ("Input template"),
03985                                      GTK_FILE_CHOOSER_ACTION_OPEN);
03986       gtk_widget_set_tooltip_text
03987         (GTK_WIDGET (window->button_template[i]),
03988          gettext ("Experimental input template file"));
03989       window->id_input[i]
03990         = g_signal_connect_swapped (window->button_template[i],
03991                                     "selection-changed",
03992                                     (void (*)) window_template_experiment,
03993                                     (void *) (size_t) i);
03994       gtk_grid_attach (window->grid_experiment,
03995                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
03996     }
03997   window->frame_experiment
03998     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
03999   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
04000                      GTK_WIDGET (window->grid_experiment));
04001
04002   // Creating the grid and attaching the widgets to the grid
04003   window->grid = (GtkGrid *) gtk_grid_new ();
04004   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
04005   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 3, 1);
04006   gtk_grid_attach (window->grid,
04007                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
04008   gtk_grid_attach (window->grid,
04009                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
04010   gtk_grid_attach (window->grid,
04011                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
04012   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
04013
04014   // Setting the window logo
04015   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
04016   gtk_window_set_icon (window->window, window->logo);
04017
04018   // Showing the window
04019   gtk_widget_show_all (GTK_WIDGET (window->window));
04020
04021   // In GTK+ 3.18 the default scrolled size is wrong
```

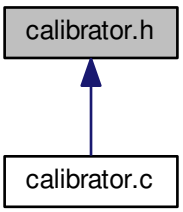
```
04022 #if GTK_MINOR_VERSION >= 18
04023   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
04024   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
04025   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
04026   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
04027 #endif
04028
04029   // Reading initial example
04030   input_new ();
04031   buffer2 = g_get_current_dir ();
04032   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
04033   g_free (buffer2);
04034   window_read (buffer);
04035   g_free (buffer);
04036 }
04037
04038 #endif
04039
04045 int
04046 cores_number ()
04047 {
04048 #ifdef G_OS_WIN32
04049   SYSTEM_INFO sysinfo;
04050   GetSystemInfo (&sysinfo);
04051   return sysinfo.dwNumberOfProcessors;
04052 #else
04053   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04054 #endif
04055 }
04056
04066 int
04067 main (int argn, char **argc)
04068 {
04069   // Starting pseudo-random numbers generator
04070   calibrate->rng = gsl_rng_alloc (gsl_rng_taus2);
04071   calibrate->seed = DEFAULT_RANDOM_SEED;
04072
04073   // Allowing spaces in the XML data file
04074   xmlKeepBlanksDefault (0);
04075
04076   // Starting MPI
04077 #if HAVE_MPI
04078   MPI_Init (&argn, &argc);
04079   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
04080   MPI_Comm_rank (MPI_COMM_WORLD, &calibrate->mpi_rank);
04081   printf ("rank=%d tasks=%d\n", calibrate->mpi_rank, ntasks);
04082 #else
04083   ntasks = 1;
04084 #endif
04085
04086 #if HAVE_GTK
04087
04088   // Getting threads number
04089   nthreads = cores_number ();
04090
04091   // Setting local language and international floating point numbers notation
04092   setlocale (LC_ALL, "");
04093   setlocale (LC_NUMERIC, "C");
04094   window->application_directory = g_get_current_dir ();
04095   bindtextdomain (PROGRAM_INTERFACE,
04096                   g_build_filename (window->application_directory,
04097                                     LOCALE_DIR, NULL));
04098   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
04099   textdomain (PROGRAM_INTERFACE);
04100
04101   // Initing GTK+
04102   gtk_disable_setlocale ();
04103   gtk_init (&argn, &argc);
04104
04105   // Opening the main window
04106   window_new ();
04107   gtk_main ();
04108
04109   // Freeing memory
04110   gtk_widget_destroy (GTK_WIDGET (window->window));
04111   g_free (window->application_directory);
04112
04113 #else
04114
04115   // Checking syntax
04116   if (!(argn == 2 || (argn == 4 && !strcmp (argc[1], "-nthreads"))))
04117     {
04118       printf ("The syntax is:\ncalibratorbin [-nthreads x] data_file\n");
04119       return 1;
04120     }
04121
04122   // Getting threads number
```

```
04123   if (argn == 2)
04124     nthreads = cores_number ();
04125   else
04126     nthreads = atoi (argc[2]);
04127   printf ("nthreads=%u\n", nthreads);
04128
04129   // Making calibration
04130   input_new ();
04131   if (input_open (argc[argn - 1]))
04132     calibrate_new ();
04133
04134   // Freeing memory
04135   calibrate_free ();
04136
04137 #endif
04138
04139   // Closing MPI
04140 #if HAVE_MPI
04141   MPI_Finalize ();
04142 #endif
04143
04144   // Freeing memory
04145   gsl_rng_free (calibrate->rng);
04146
04147   // Closing
04148   return 0;
04149 }
```

5.3 calibrator.h File Reference

Header file of the calibrator.

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



Data Structures

- struct Input

  Struct to define the calibration input file.
- struct Calibrate

  Struct to define the calibration data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Enumerations

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

  Enum to define the algorithms.

Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Function to calibrate with the genetic algorithm.

- void calibrate_save_old ()

  Function to save the best results on iterative methods.

- void calibrate_merge_old ()

  Function to merge the best results with the previous step best results on iterative methods.

- void calibrate_refine ()

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

- void calibrate_iterate ()

  Function to iterate the algorithm.

- void calibrate_new ()

  Function to open and perform a calibration.

5.3.1 Detailed Description

Header file of the calibrator.

Authors

Javier Burguete.

Copyright

Definition in file calibrator.h.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

Definition at line 43 of file calibrator.h.

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

5.3.3 Function Documentation

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

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 1341 of file calibrator.c.

```
01342 {
01343   unsigned int i, j;
01344   double e;
01345 #if DEBUG
01346   fprintf (stderr, "calibrate_best_sequential: start\n");
01347 #endif
01348   if (calibrate->nsaveds < calibrate->nbest
01349       || value < calibrate->error_best[calibrate->nsaveds - 1])
01350     {
01351       if (calibrate->nsaveds < calibrate->nbest)
01352         ++calibrate->nsaveds;
01353       calibrate->error_best[calibrate->nsaveds - 1] = value;
01354       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01355       for (i = calibrate->nsaveds; --i;)
01356         {
01357           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01358             {
01359               j = calibrate->simulation_best[i];
01360               e = calibrate->error_best[i];
01361               calibrate->simulation_best[i] = calibrate->
      simulation_best[i - 1];
01362               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01363               calibrate->simulation_best[i - 1] = j;
01364               calibrate->error_best[i - 1] = e;
01365             }
01366           else
01367             break;
01368         }
01369     }
01370 #if DEBUG
01371   fprintf (stderr, "calibrate_best_sequential: end\n");
01372 #endif
01373 }
```

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 1296 of file calibrator.c.

```
01297 {
01298   unsigned int i, j;
01299   double e;
01300 #if DEBUG
01301   fprintf (stderr, "calibrate_best_thread: start\n");
01302 #endif
01303   if (calibrate->nsaveds < calibrate->nbest
01304       || value < calibrate->error_best[calibrate->nsaveds - 1])
01305     {
01306       g_mutex_lock (mutex);
01307       if (calibrate->nsaveds < calibrate->nbest)
01308         ++calibrate->nsaveds;
01309       calibrate->error_best[calibrate->nsaveds - 1] = value;
01310       calibrate->simulation_best[calibrate->
      nsaveds - 1] = simulation;
01311       for (i = calibrate->nsaveds; --i;)
01312         {
01313           if (calibrate->error_best[i] < calibrate->
      error_best[i - 1])
01314             {
01315               j = calibrate->simulation_best[i];
01316               e = calibrate->error_best[i];
01317               calibrate->simulation_best[i] = calibrate->
```

```
      simulation_best[i - 1];
01318               calibrate->error_best[i] = calibrate->
      error_best[i - 1];
01319               calibrate->simulation_best[i - 1] = j;
01320               calibrate->error_best[i - 1] = e;
01321             }
01322           else
01323             break;
01324         }
01325       g_mutex_unlock (mutex);
01326     }
01327 #if DEBUG
01328   fprintf (stderr, "calibrate_best_thread: end\n");
01329 #endif
01330 }
```

5.3.3.3 double calibrate_genetic_objective (Entity * *entity*)

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

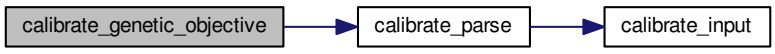
Returns

objective function value.

Definition at line 1650 of file calibrator.c.

```
01651 {
01652   unsigned int j;
01653   double objective;
01654   char buffer[64];
01655 #if DEBUG
01656   fprintf (stderr, "calibrate_genetic_objective: start\n");
01657 #endif
01658   for (j = 0; j < calibrate->nvariables; ++j)
01659     {
01660       calibrate->value[entity->id * calibrate->nvariables + j]
01661         = genetic_get_variable (entity, calibrate->genetic_variable + j);
01662     }
01663   for (j = 0, objective = 0.; j < calibrate->nexperiments; ++j)
01664     objective += calibrate_parse (entity->id, j);
01665   g_mutex_lock (mutex);
01666   for (j = 0; j < calibrate->nvariables; ++j)
01667     {
01668       snprintf (buffer, 64, "%s ", format[calibrate->precision[j]]);
01669       fprintf (calibrate->file_variables, buffer,
01670                genetic_get_variable (entity, calibrate->
      genetic_variable + j));
01671     }
01672   fprintf (calibrate->file_variables, "%.14le\n", objective);
01673   g_mutex_unlock (mutex);
01674 #if DEBUG
01675   fprintf (stderr, "calibrate_genetic_objective: end\n");
01676 #endif
01677   return objective;
01678 }
```

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 1049 of file calibrator.c.

```
01050 {
01051   unsigned int i;
01052   char buffer[32], value[32], *buffer2, *buffer3, *content;
01053   FILE *file;
01054   gsize length;
01055   GRegex *regex;
01056
01057 #if DEBUG
01058   fprintf (stderr, "calibrate_input: start\n");
01059 #endif
01060
01061   // Checking the file
01062   if (!template)
01063     goto calibrate_input_end;
01064
01065   // Opening template
01066   content = g_mapped_file_get_contents (template);
01067   length = g_mapped_file_get_length (template);
01068 #if DEBUG
01069   fprintf (stderr, "calibrate_input: length=%lu\ncontent:\n%s", length,
01070            content);
01071 #endif
01072   file = g_fopen (input, "w");
01073
01074   // Parsing template
01075   for (i = 0; i < calibrate->nvariables; ++i)
01076     {
01077 #if DEBUG
01078       fprintf (stderr, "calibrate_input: variable=%u\n", i);
01079 #endif
01080       snprintf (buffer, 32, "@variable%u@", i + 1);
01081       regex = g_regex_new (buffer, 0, 0, NULL);
01082       if (i == 0)
01083         {
01084           buffer2 = g_regex_replace_literal (regex, content, length, 0,
01085                                              calibrate->label[i], 0, NULL);
01086 #if DEBUG
01087           fprintf (stderr, "calibrate_input: buffer2\n%s", buffer2);
01088 #endif
01089         }
01090       else
01091         {
01092           length = strlen (buffer3);
01093           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
01094                                              calibrate->label[i], 0, NULL);
01095           g_free (buffer3);
01096         }
01097       g_regex_unref (regex);
01098       length = strlen (buffer2);
01099       snprintf (buffer, 32, "@value%u@", i + 1);
01100       regex = g_regex_new (buffer, 0, 0, NULL);
01101       snprintf (value, 32, format[calibrate->precision[i]],
01102                 calibrate->value[simulation * calibrate->
      nvariables + i]);
01103
01104 #if DEBUG
01105       fprintf (stderr, "calibrate_input: value=%s\n", value);
01106 #endif
01107       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
01108                                          0, NULL);
01109       g_free (buffer2);
01110       g_regex_unref (regex);
01111     }
01112
01113   // Saving input file
01114   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
01115   g_free (buffer3);
01116   fclose (file);
01117
01118 calibrate_input_end:
01119 #if DEBUG
01120   fprintf (stderr, "calibrate_input: end\n");
01121 #endif
01122   return;
01123 }
```

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 1457 of file calibrator.c.

```
01459 {
01460   unsigned int i, j, k, s[calibrate->nbest];
01461   double e[calibrate->nbest];
01462 #if DEBUG
01463   fprintf (stderr, "calibrate_merge: start\n");
01464 #endif
01465   i = j = k = 0;
01466   do
01467     {
01468       if (i == calibrate->nsaveds)
01469         {
01470           s[k] = simulation_best[j];
01471           e[k] = error_best[j];
01472           ++j;
01473           ++k;
01474           if (j == nsaveds)
01475             break;
01476         }
01477       else if (j == nsaveds)
01478         {
01479           s[k] = calibrate->simulation_best[i];
01480           e[k] = calibrate->error_best[i];
01481           ++i;
01482           ++k;
01483           if (i == calibrate->nsaveds)
01484             break;
01485         }
01486       else if (calibrate->error_best[i] > error_best[j])
01487         {
01488           s[k] = simulation_best[j];
01489           e[k] = error_best[j];
01490           ++j;
01491           ++k;
01492         }
01493       else
01494         {
01495           s[k] = calibrate->simulation_best[i];
01496           e[k] = calibrate->error_best[i];
01497           ++i;
01498           ++k;
01499         }
01500     }
01501   while (k < calibrate->nbest);
01502   calibrate->nsaveds = k;
01503   memcpy (calibrate->simulation_best, s, k * sizeof (unsigned int));
01504   memcpy (calibrate->error_best, e, k * sizeof (double));
01505 #if DEBUG
01506   fprintf (stderr, "calibrate_merge: end\n");
01507 #endif
01508 }
```

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 1136 of file calibrator.c.

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

```
01216           snprintf (buffer, 512, RM " %s", &input[i][0]);
01217           system (buffer);
01218         }
01219     }
01220   snprintf (buffer, 512, RM " %s %s", output, result);
01221   system (buffer);
01222 #endif
01223
01224 #if DEBUG
01225   fprintf (stderr, "calibrate_parse: end\n");
01226 #endif
01227
01228   // Returning the objective function
01229   return e * calibrate->weight[experiment];
01230 }
```

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 1268 of file calibrator.c.

```
01269 {
01270   unsigned int i;
01271   char buffer[64];
01272 #if DEBUG
01273   fprintf (stderr, "calibrate_save_variables: start\n");
01274 #endif
01275   for (i = 0; i < calibrate->nvariables; ++i)
01276     {
01277       snprintf (buffer, 64, "%s ", format[calibrate->precision[i]]);
01278       fprintf (calibrate->file_variables, buffer,
01279                calibrate->value[simulation * calibrate->
      nvariables + i]);
01280     }
01281   fprintf (calibrate->file_variables, "%.14le\n", error);
01282 #if DEBUG
01283   fprintf (stderr, "calibrate_save_variables: end\n");
01284 #endif
01285 }
```

5.3.3.8 void* calibrate_thread (ParallelData * *data*)

Function to calibrate on a thread.

Parameters

data	Function data.

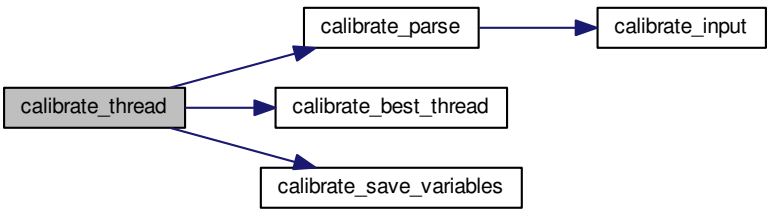
Returns

NULL

Definition at line 1383 of file calibrator.c.

```
01384 {
01385   unsigned int i, j, thread;
01386   double e;
01387 #if DEBUG
01388   fprintf (stderr, "calibrate_thread: start\n");
01389 #endif
01390   thread = data->thread;
01391 #if DEBUG
01392   fprintf (stderr, "calibrate_thread: thread=%u start=%u end=%u\n", thread,
01393            calibrate->thread[thread], calibrate->thread[thread + 1]);
01394 #endif
01395   for (i = calibrate->thread[thread]; i < calibrate->thread[thread + 1]; ++i)
01396     {
01397       e = 0.;
01398       for (j = 0; j < calibrate->nexperiments; ++j)
01399         e += calibrate_parse (i, j);
01400       calibrate_best_thread (i, e);
01401       g_mutex_lock (mutex);
01402       calibrate_save_variables (i, e);
01403       g_mutex_unlock (mutex);
01404 #if DEBUG
01405       fprintf (stderr, "calibrate_thread: i=%u e=%lg\n", i, e);
01406 #endif
01407     }
01408 #if DEBUG
01409   fprintf (stderr, "calibrate_thread: end\n");
01410 #endif
01411   g_thread_exit (NULL);
01412   return NULL;
01413 }
```

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 479 of file calibrator.c.

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

```
00555           msg = gettext ("Bad simulations number");
00556           goto exit_on_error;
00557         }
00558     }
00559   else if (!xmlStrcmp (buffer, XML_SWEEP))
00560     input->algorithm = ALGORITHM_SWEEP;
00561   else if (!xmlStrcmp (buffer, XML_GENETIC))
00562     {
00563       input->algorithm = ALGORITHM_GENETIC;
00564
00565       // Obtaining population
00566       if (xmlHasProp (node, XML_NPOPULATION))
00567         {
00568           input->nsimulations
00569             = xml_node_get_uint (node, XML_NPOPULATION, &error_code);
00570           if (error_code || input->nsimulations < 3)
00571             {
00572               msg = gettext ("Invalid population number");
00573               goto exit_on_error;
00574             }
00575         }
00576       else
00577         {
00578           msg = gettext ("No population number");
00579           goto exit_on_error;
00580         }
00581
00582       // Obtaining generations
00583       if (xmlHasProp (node, XML_NGENERATIONS))
00584         {
00585           input->niterations
00586             = xml_node_get_uint (node, XML_NGENERATIONS, &error_code);
00587           if (error_code || !input->niterations)
00588             {
00589               msg = gettext ("Invalid generations number");
00590               goto exit_on_error;
00591             }
00592         }
00593       else
00594         {
00595           msg = gettext ("No generations number");
00596           goto exit_on_error;
00597         }
00598
00599       // Obtaining mutation probability
00600       if (xmlHasProp (node, XML_MUTATION))
00601         {
00602           input->mutation_ratio
00603             = xml_node_get_float (node, XML_MUTATION, &error_code);
00604           if (error_code || input->mutation_ratio < 0.
00605               || input->mutation_ratio >= 1.)
00606             {
00607               msg = gettext ("Invalid mutation probability");
00608               goto exit_on_error;
00609             }
00610         }
00611       else
00612         {
00613           msg = gettext ("No mutation probability");
00614           goto exit_on_error;
00615         }
00616
00617       // Obtaining reproduction probability
00618       if (xmlHasProp (node, XML_REPRODUCTION))
00619         {
00620           input->reproduction_ratio
00621             = xml_node_get_float (node, XML_REPRODUCTION, &error_code);
00622           if (error_code || input->reproduction_ratio < 0.
00623               || input->reproduction_ratio >= 1.0)
00624             {
00625               msg = gettext ("Invalid reproduction probability");
00626               goto exit_on_error;
00627             }
00628         }
00629       else
00630         {
00631           msg = gettext ("No reproduction probability");
00632           goto exit_on_error;
00633         }
00634
00635       // Obtaining adaptation probability
00636       if (xmlHasProp (node, XML_ADAPTATION))
00637         {
00638           input->adaptation_ratio
00639             = xml_node_get_float (node, XML_ADAPTATION, &error_code);
00640           if (error_code || input->adaptation_ratio < 0.
00641               || input->adaptation_ratio >= 1.)
```

```
00642             {
00643               msg = gettext ("Invalid adaptation probability");
00644               goto exit_on_error;
00645             }
00646         }
00647       else
00648         {
00649           msg = gettext ("No adaptation probability");
00650           goto exit_on_error;
00651         }
00652
00653       // Checking survivals
00654       i = input->mutation_ratio * input->nsimulations;
00655       i += input->reproduction_ratio * input->
      nsimulations;
00656       i += input->adaptation_ratio * input->
      nsimulations;
00657       if (i > input->nsimulations - 2)
00658         {
00659           msg = gettext
00660             ("No enough survival entities to reproduce the population");
00661           goto exit_on_error;
00662         }
00663     }
00664   else
00665     {
00666       msg = gettext ("Unknown algorithm");
00667       goto exit_on_error;
00668     }
00669
00670   if (input->algorithm == ALGORITHM_MONTE_CARLO
00671       || input->algorithm == ALGORITHM_SWEEP)
00672     {
00673
00674       // Obtaining iterations number
00675       input->niterations
00676         = xml_node_get_int (node, XML_NITERATIONS, &error_code);
00677       if (error_code == 1)
00678         input->niterations = 1;
00679       else if (error_code)
00680         {
00681           msg = gettext ("Bad iterations number");
00682           goto exit_on_error;
00683         }
00684
00685       // Obtaining best number
00686       if (xmlHasProp (node, XML_NBEST))
00687         {
00688           input->nbest = xml_node_get_uint (node,
      XML_NBEST, &error_code);
00689           if (error_code || !input->nbest)
00690             {
00691               msg = gettext ("Invalid best number");
00692               goto exit_on_error;
00693             }
00694         }
00695       else
00696         input->nbest = 1;
00697
00698       // Obtaining tolerance
00699       if (xmlHasProp (node, XML_TOLERANCE))
00700         {
00701           input->tolerance
00702             = xml_node_get_float (node, XML_TOLERANCE, &error_code);
00703           if (error_code || input->tolerance < 0.)
00704             {
00705               msg = gettext ("Invalid tolerance");
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         input->tolerance = 0.;
00711     }
00712
00713   // Reading the experimental data
00714   for (child = node->children; child; child = child->next)
00715     {
00716       if (xmlStrcmp (child->name, XML_EXPERIMENT))
00717         break;
00718 #if DEBUG
00719       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00720 #endif
00721       if (xmlHasProp (child, XML_NAME))
00722         {
00723           input->experiment
00724             = g_realloc (input->experiment,
00725                          (1 + input->nexperiments) * sizeof (char *));
```

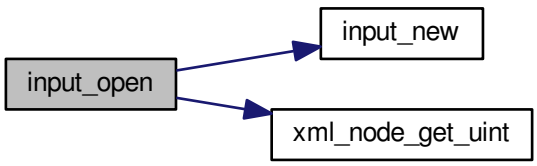
```
00726           input->experiment[input->nexperiments]
00727             = (char *) xmlGetProp (child, XML_NAME);
00728         }
00729       else
00730         {
00731           snprintf (buffer2, 64, "%s %u: %s",
00732                     gettext ("Experiment"),
00733                     input->nexperiments + 1, gettext ("no data file name"));
00734           msg = buffer2;
00735           goto exit_on_error;
00736         }
00737 #if DEBUG
00738       fprintf (stderr, "input_open: experiment=%s\n",
00739                input->experiment[input->nexperiments]);
00740 #endif
00741       input->weight = g_realloc (input->weight,
00742                                  (1 + input->nexperiments) * sizeof (double));
00743       if (xmlHasProp (child, XML_WEIGHT))
00744         {
00745           input->weight[input->nexperiments]
00746             = xml_node_get_float (child, XML_WEIGHT, &error_code);
00747           if (error_code)
00748             {
00749               snprintf (buffer2, 64, "%s %u: %s",
00750                         gettext ("Experiment"),
00751                         input->nexperiments + 1, gettext ("bad weight"));
00752               msg = buffer2;
00753               goto exit_on_error;
00754             }
00755         }
00756       else
00757         input->weight[input->nexperiments] = 1.;
00758 #if DEBUG
00759       fprintf (stderr, "input_open: weight=%lg\n",
00760                input->weight[input->nexperiments]);
00761 #endif
00762       if (!input->nexperiments)
00763         input->ninputs = 0;
00764 #if DEBUG
00765       fprintf (stderr, "input_open: template[0]\n");
00766 #endif
00767       if (xmlHasProp (child, XML_TEMPLATE1))
00768         {
00769           input->template[0]
00770             = (char **) g_realloc (input->template[0],
00771                                    (1 + input->nexperiments) * sizeof (char *));
00772           input->template[0][input->nexperiments]
00773             = (char *) xmlGetProp (child, template[0]);
00774 #if DEBUG
00775           fprintf (stderr, "input_open: experiment=%u template1=%s\n",
00776                    input->nexperiments,
00777                    input->template[0][input->nexperiments]);
00778 #endif
00779           if (!input->nexperiments)
00780             ++input->ninputs;
00781 #if DEBUG
00782           fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00783 #endif
00784         }
00785       else
00786         {
00787           snprintf (buffer2, 64, "%s %u: %s",
00788                     gettext ("Experiment"),
00789                     input->nexperiments + 1, gettext ("no template"));
00790           msg = buffer2;
00791           goto exit_on_error;
00792         }
00793       for (i = 1; i < MAX_NINPUTS; ++i)
00794         {
00795 #if DEBUG
00796           fprintf (stderr, "input_open: template%u\n", i + 1);
00797 #endif
00798           if (xmlHasProp (child, template[i]))
00799             {
00800               if (input->nexperiments && input->ninputs <= i)
00801                 {
00802                   snprintf (buffer2, 64, "%s %u: %s",
00803                             gettext ("Experiment"),
00804                             input->nexperiments + 1,
00805                             gettext ("bad templates number"));
00806                   msg = buffer2;
00807                   goto exit_on_error;
00808                 }
00809               input->template[i] = (char **)
00810                 g_realloc (input->template[i],
00811                            (1 + input->nexperiments) * sizeof (char *));
00812               input->template[i][input->nexperiments]
```

```
00813                 = (char *) xmlGetProp (child, template[i]);
00814 #if DEBUG
00815               fprintf (stderr, "input_open: experiment=%u template%u=%s\n",
00816                        input->nexperiments, i + 1,
00817                        input->template[i][input->nexperiments]);
00818 #endif
00819               if (!input->nexperiments)
00820                 ++input->ninputs;
00821 #if DEBUG
00822               fprintf (stderr, "input_open: ninputs=%u\n", input->ninputs);
00823 #endif
00824             }
00825           else if (input->nexperiments && input->ninputs >= i)
00826             {
00827               snprintf (buffer2, 64, "%s %u: %s%u",
00828                         gettext ("Experiment"),
00829                         input->nexperiments + 1,
00830                         gettext ("no template"), i + 1);
00831               msg = buffer2;
00832               goto exit_on_error;
00833             }
00834           else
00835             break;
00836         }
00837       ++input->nexperiments;
00838 #if DEBUG
00839       fprintf (stderr, "input_open: nexperiments=%u\n", input->nexperiments);
00840 #endif
00841     }
00842   if (!input->nexperiments)
00843     {
00844       msg = gettext ("No calibration experiments");
00845       goto exit_on_error;
00846     }
00847
00848   // Reading the variables data
00849   for (; child; child = child->next)
00850     {
00851       if (xmlStrcmp (child->name, XML_VARIABLE))
00852         {
00853           snprintf (buffer2, 64, "%s %u: %s",
00854                     gettext ("Variable"),
00855                     input->nvariables + 1, gettext ("bad XML node"));
00856           msg = buffer2;
00857           goto exit_on_error;
00858         }
00859       if (xmlHasProp (child, XML_NAME))
00860         {
00861           input->label = g_realloc
00862             (input->label, (1 + input->nvariables) * sizeof (char *));
00863           input->label[input->nvariables]
00864             = (char *) xmlGetProp (child, XML_NAME);
00865         }
00866       else
00867         {
00868           snprintf (buffer2, 64, "%s %u: %s",
00869                     gettext ("Variable"),
00870                     input->nvariables + 1, gettext ("no name"));
00871           msg = buffer2;
00872           goto exit_on_error;
00873         }
00874       if (xmlHasProp (child, XML_MINIMUM))
00875         {
00876           input->rangemin = g_realloc
00877             (input->rangemin, (1 + input->nvariables) * sizeof (double));
00878           input->rangeminabs = g_realloc
00879             (input->rangeminabs, (1 + input->nvariables) * sizeof (double));
00880           input->rangemin[input->nvariables]
00881             = xml_node_get_float (child, XML_MINIMUM, &error_code);
00882           if (xmlHasProp (child, XML_ABSOLUTE_MINIMUM))
00883             {
00884               input->rangeminabs[input->nvariables]
00885                 = xml_node_get_float (child,
      XML_ABSOLUTE_MINIMUM, &error_code);
00886             }
00887           else
00888             input->rangeminabs[input->nvariables] = -G_MAXDOUBLE;
00889           if (input->rangemin[input->nvariables]
00890               < input->rangeminabs[input->nvariables])
00891             {
00892               snprintf (buffer2, 64, "%s %u: %s",
00893                         gettext ("Variable"),
00894                         input->nvariables + 1,
00895                         gettext ("minimum range not allowed"));
00896               msg = buffer2;
00897               goto exit_on_error;
00898             }
```

```
00899         }
00900       else
00901         {
00902           snprintf (buffer2, 64, "%s %u: %s",
00903                     gettext ("Variable"),
00904                     input->nvariables + 1, gettext ("no minimum range"));
00905           msg = buffer2;
00906           goto exit_on_error;
00907         }
00908       if (xmlHasProp (child, XML_MAXIMUM))
00909         {
00910           input->rangemax = g_realloc
00911             (input->rangemax, (1 + input->nvariables) * sizeof (double));
00912           input->rangemaxabs = g_realloc
00913             (input->rangemaxabs, (1 + input->nvariables) * sizeof (double));
00914           input->rangemax[input->nvariables]
00915             = xml_node_get_float (child, XML_MAXIMUM, &error_code);
00916           if (xmlHasProp (child, XML_ABSOLUTE_MAXIMUM))
00917             input->rangemaxabs[input->nvariables]
00918               = xml_node_get_float (child,
      XML_ABSOLUTE_MAXIMUM, &error_code);
00919           else
00920             input->rangemaxabs[input->nvariables] = G_MAXDOUBLE;
00921           if (input->rangemax[input->nvariables]
00922               > input->rangemaxabs[input->nvariables])
00923             {
00924               snprintf (buffer2, 64, "%s %u: %s",
00925                         gettext ("Variable"),
00926                         input->nvariables + 1,
00927                         gettext ("maximum range not allowed"));
00928               msg = buffer2;
00929               goto exit_on_error;
00930             }
00931         }
00932       else
00933         {
00934           snprintf (buffer2, 64, "%s %u: %s",
00935                     gettext ("Variable"),
00936                     input->nvariables + 1, gettext ("no maximum range"));
00937           msg = buffer2;
00938           goto exit_on_error;
00939         }
00940       if (input->rangemax[input->nvariables]
00941           < input->rangemin[input->nvariables])
00942         {
00943           snprintf (buffer2, 64, "%s %u: %s",
00944                     gettext ("Variable"),
00945                     input->nvariables + 1, gettext ("bad range"));
00946           msg = buffer2;
00947           goto exit_on_error;
00948         }
00949       input->precision = g_realloc
00950         (input->precision, (1 + input->nvariables) * sizeof (unsigned int));
00951       if (xmlHasProp (child, XML_PRECISION))
00952         input->precision[input->nvariables]
00953           = xml_node_get_uint (child, XML_PRECISION, &error_code);
00954       else
00955         input->precision[input->nvariables] =
      DEFAULT_PRECISION;
00956       if (input->algorithm == ALGORITHM_SWEEP)
00957         {
00958           if (xmlHasProp (child, XML_NSWEEPS))
00959             {
00960               input->nsweeps = (unsigned int *)
00961                 g_realloc (input->nsweeps,
00962                            (1 + input->nvariables) * sizeof (unsigned int));
00963               input->nsweeps[input->nvariables]
00964                 = xml_node_get_uint (child, XML_NSWEEPS, &error_code);
00965             }
00966           else
00967             {
00968               snprintf (buffer2, 64, "%s %u: %s",
00969                         gettext ("Variable"),
00970                         input->nvariables + 1, gettext ("no sweeps number"));
00971               msg = buffer2;
00972               goto exit_on_error;
00973             }
00974 #if DEBUG
00975           fprintf (stderr, "input_open: nsweeps=%u nsimulations=%u\n",
00976                    input->nsweeps[input->nvariables],
      input->nsimulations);
00977 #endif
00978         }
00979       if (input->algorithm == ALGORITHM_GENETIC)
00980         {
00981           // Obtaining bits representing each variable
00982           if (xmlHasProp (child, XML_NBITS))
```

```
00983             {
00984               input->nbits = (unsigned int *)
00985                 g_realloc (input->nbits,
00986                            (1 + input->nvariables) * sizeof (unsigned int));
00987               i = xml_node_get_uint (child, XML_NBITS, &error_code);
00988               if (error_code || !i)
00989                 {
00990                   snprintf (buffer2, 64, "%s %u: %s",
00991                             gettext ("Variable"),
00992                             input->nvariables + 1,
00993                             gettext ("invalid bits number"));
00994                   msg = buffer2;
00995                   goto exit_on_error;
00996                 }
00997               input->nbits[input->nvariables] = i;
00998             }
00999           else
01000             {
01001               snprintf (buffer2, 64, "%s %u: %s",
01002                         gettext ("Variable"),
01003                         input->nvariables + 1, gettext ("no bits number"));
01004               msg = buffer2;
01005               goto exit_on_error;
01006             }
01007         }
01008       ++input->nvariables;
01009     }
01010   if (!input->nvariables)
01011     {
01012       msg = gettext ("No calibration variables");
01013       goto exit_on_error;
01014     }
01015
01016   // Getting the working directory
01017   input->directory = g_path_get_dirname (filename);
01018   input->name = g_path_get_basename (filename);
01019
01020   // Closing the XML document
01021   xmlFreeDoc (doc);
01022
01023 #if DEBUG
01024   fprintf (stderr, "input_open: end\n");
01025 #endif
01026   return 1;
01027
01028 exit_on_error:
01029   show_error (msg);
01030   input_free ();
01031 #if DEBUG
01032   fprintf (stderr, "input_open: end\n");
01033 #endif
01034   return 0;
01035 }
```

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.

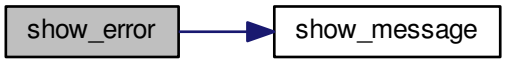
Parameters

msg	Error message.

Definition at line 251 of file calibrator.c.

```
00252 {
00253   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00254 }
```

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 221 of file calibrator.c.

```
00222 {
00223 #if HAVE_GTK
00224   GtkMessageDialog *dlg;
00225
00226   // Creating the dialog
00227   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00228     (window->window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00229
00230   // Setting the dialog title
00231   gtk_window_set_title (GTK_WINDOW (dlg), title);
00232
00233   // Showing the dialog and waiting response
00234   gtk_dialog_run (GTK_DIALOG (dlg));
00235
00236   // Closing and freeing memory
00237   gtk_widget_destroy (GTK_WIDGET (dlg));
00238
00239 #else
00240   printf ("%s: %s\n", title, msg);
00241 #endif
00242 }
```

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 330 of file calibrator.c.

```
00331 {
00332   double x = 0.;
00333   xmlChar *buffer;
00334   buffer = xmlGetProp (node, prop);
00335   if (!buffer)
00336     *error_code = 1;
00337   else
00338     {
00339       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343       xmlFree (buffer);
00344     }
00345   return x;
00346 }
```

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 268 of file calibrator.c.

```
00269 {
00270   int i = 0;
00271   xmlChar *buffer;
00272   buffer = xmlGetProp (node, prop);
00273   if (!buffer)
00274     *error_code = 1;
00275   else
00276     {
00277       if (sscanf ((char *) buffer, "%d", &i) != 1)
00278         *error_code = 2;
00279       else
00280         *error_code = 0;
00281       xmlFree (buffer);
00282     }
00283   return i;
00284 }
```

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 299 of file calibrator.c.

```
00300 {
00301   unsigned int i = 0;
00302   xmlChar *buffer;
00303   buffer = xmlGetProp (node, prop);
00304   if (!buffer)
00305     *error_code = 1;
00306   else
00307     {
00308       if (sscanf ((char *) buffer, "%u", &i) != 1)
00309         *error_code = 2;
00310       else
00311         *error_code = 0;
00312       xmlFree (buffer);
00313     }
00314   return i;
00315 }
```

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 397 of file calibrator.c.

```
00398 {
00399   xmlChar buffer[64];
00400   snprintf ((char *) buffer, 64, "%.14lg", value);
00401   xmlSetProp (node, prop, buffer);
00402 }
```

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 359 of file calibrator.c.

```
00360 {
00361   xmlChar buffer[64];
00362   snprintf ((char *) buffer, 64, "%d", value);
00363   xmlSetProp (node, prop, buffer);
00364 }
```

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 378 of file calibrator.c.

```
00379 {
00380   xmlChar buffer[64];
00381   snprintf ((char *) buffer, 64, "%u", value);
00382   xmlSetProp (node, prop, buffer);
00383 }
```

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 *result;
00057   char *variables;
00058   char *simulator;
00059   char *evaluator;
00061   char **experiment;
00062   char **template[MAX_NINPUTS];
00063   char **label;
00064   char *directory;
00065   char *name;
00066   double *rangemin;
00067   double *rangemax;
00068   double *rangeminabs;
00069   double *rangemaxabs;
00070   double *weight;
00071   double tolerance;
00072   double mutation_ratio;
00073   double reproduction_ratio;
00074   double adaptation_ratio;
00075   unsigned long int seed;
00077   unsigned int nvariables;
00078   unsigned int nexperiments;
00079   unsigned int ninputs;
00080   unsigned int nsimulations;
```

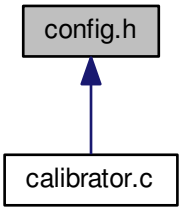
```
00081   unsigned int algorithm;
00082   unsigned int *precision;
00083   unsigned int *nsweeps;
00084   unsigned int *nbits;
00086   unsigned int niterations;
00087   unsigned int nbest;
00088 } Input;
00089
00094 typedef struct
00095 {
00096   char *result;
00097   char *variables;
00098   char *simulator;
00099   char *evaluator;
00101   char **experiment;
00102   char **template[MAX_NINPUTS];
00103   char **label;
00104   unsigned int nvariables;
00105   unsigned int nexperiments;
00106   unsigned int ninputs;
00107   unsigned int nsimulations;
00108   unsigned int algorithm;
00109   unsigned int *precision;
00110   unsigned int *nsweeps;
00111   unsigned int nstart;
00112   unsigned int nend;
00113   unsigned int *thread;
00115   unsigned int niterations;
00116   unsigned int nbest;
00117   unsigned int nsaveds;
00118   unsigned int *simulation_best;
00119   unsigned long int seed;
00121   double *value;
00122   double *rangemin;
00123   double *rangemax;
00124   double *rangeminabs;
00125   double *rangemaxabs;
00126   double *error_best;
00127   double *weight;
00128   double *value_old;
00130   double *error_old;
00132   double tolerance;
00133   double mutation_ratio;
00134   double reproduction_ratio;
00135   double adaptation_ratio;
00136   double calculation_time;
00137   FILE *file_result;
00138   FILE *file_variables;
00139   gsl_rng *rng;
00140   GMappedFile **file[MAX_NINPUTS];
00141   GeneticVariable *genetic_variable;
00143 #if HAVE_MPI
00144   int mpi_rank;
00145 #endif
00146 } Calibrate;
00147
00152 typedef struct
00153 {
00154   unsigned int thread;
00155 } ParallelData;
00156
00157 // Public functions
00158 void show_message (char *title, char *msg, int type);
00159 void show_error (char *msg);
00160 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00161 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00162                                 int *error_code);
00163 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00164                            int *error_code);
00165 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00166 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00167                         unsigned int value);
00168 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00169 void input_new ();
00170 void input_free ();
00171 int input_open (char *filename);
00172 void calibrate_input (unsigned int simulation, char *input,
00173                       GMappedFile * template);
00174 double calibrate_parse (unsigned int simulation, unsigned int experiment);
00175 void calibrate_print ();
00176 void calibrate_save_variables (unsigned int simulation, double error);
00177 void calibrate_best_thread (unsigned int simulation, double value);
00178 void calibrate_best_sequential (unsigned int simulation, double value);
00179 void *calibrate_thread (ParallelData * data);
00180 void calibrate_sequential ();
00181 void calibrate_merge (unsigned int nsaveds, unsigned int *simulation_best,
00182                       double *error_best);
```

```
00183 #if HAVE_MPI
00184 void calibrate_synchronise ();
00185 #endif
00186 void calibrate_sweep ();
00187 void calibrate_MonteCarlo ();
00188 double calibrate_genetic_objective (Entity * entity);
00189 void calibrate_genetic ();
00190 void calibrate_save_old ();
00191 void calibrate_merge_old ();
00192 void calibrate_refine ();
00193 void calibrate_iterate ();
00194 void calibrate_new ();
00195
00196 #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_RESULT (const xmlChar*)"result"
  result 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_VARIABLES (const xmlChar*)"variables"

  variables 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

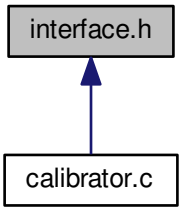
```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 Calibrator: a software to make calibrations of empirical parameters.
00004
00005 AUTHORS: Javier Burguete and Borja Latorre.
00006
00007 Copyright 2012-2014, AUTHORS.
00008
00009 Redistribution and use in source and binary 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_RESULT (const xmlChar*)"result"
00096 #define XML_SIMULATOR (const xmlChar*)"simulator"
00097 #define XML_SEED (const xmlChar*)"seed"
00099 #define XML_SWEEP (const xmlChar*)"sweep"
00100 #define XML_TEMPLATE1 (const xmlChar*)"template1"
00101 #define XML_TEMPLATE2 (const xmlChar*)"template2"
00103 #define XML_TEMPLATE3 (const xmlChar*)"template3"
00105 #define XML_TEMPLATE4 (const xmlChar*)"template4"
00107 #define XML_TEMPLATE5 (const xmlChar*)"template5"
00109 #define XML_TEMPLATE6 (const xmlChar*)"template6"
00111 #define XML_TEMPLATE7 (const xmlChar*)"template7"
00113 #define XML_TEMPLATE8 (const xmlChar*)"template8"
00115 #define XML_TOLERANCE (const xmlChar*)"tolerance"
00117 #define XML_VARIABLE (const xmlChar*)"variable"
00119 #define XML_VARIABLES (const xmlChar*)"variables"
00120 #define XML_WEIGHT (const xmlChar*)"weight"
00122
00123 #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 4046 of file calibrator.c.

```
04047 {
04048 #ifdef G_OS_WIN32
04049   SYSTEM_INFO sysinfo;
04050   GetSystemInfo (&sysinfo);
04051   return sysinfo.dwNumberOfProcessors;
04052 #else
04053   return (int) sysconf (_SC_NPROCESSORS_ONLN);
04054 #endif
04055 }
```

5.7.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

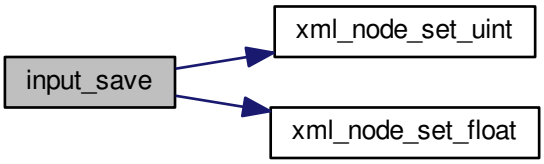
Definition at line 2183 of file calibrator.c.

```
02184 {
02185   unsigned int i, j;
02186   char *buffer;
02187   xmlDoc *doc;
02188   xmlNode *node, *child;
02189   GFile *file, *file2;
02190
02191   // Getting the input file directory
02192   input->name = g_path_get_basename (filename);
02193   input->directory = g_path_get_dirname (filename);
02194   file = g_file_new_for_path (input->directory);
02195
02196   // Opening the input file
02197   doc = xmlNewDoc ((const xmlChar *) "1.0");
02198
02199   // Setting root XML node
02200   node = xmlNewDocNode (doc, 0, XML_CALIBRATE, 0);
02201   xmlDocSetRootElement (doc, node);
02202
02203   // Adding properties to the root XML node
02204   if (xmlStrcmp ((const xmlChar *) input->result, result_name))
```

```
02205     xmlSetProp (node, XML_RESULT, (xmlChar *) input->result);
02206   if (xmlStrcmp ((const xmlChar *) input->variables,
      variables_name))
02207     xmlSetProp (node, XML_VARIABLES, (xmlChar *) input->
      variables);
02208   file2 = g_file_new_for_path (input->simulator);
02209   buffer = g_file_get_relative_path (file, file2);
02210   g_object_unref (file2);
02211   xmlSetProp (node, XML_SIMULATOR, (xmlChar *) buffer);
02212   g_free (buffer);
02213   if (input->evaluator)
02214     {
02215       file2 = g_file_new_for_path (input->evaluator);
02216       buffer = g_file_get_relative_path (file, file2);
02217       g_object_unref (file2);
02218       if (xmlStrlen ((xmlChar *) buffer))
02219         xmlSetProp (node, XML_EVALUATOR, (xmlChar *) buffer);
02220       g_free (buffer);
02221     }
02222   if (input->seed != DEFAULT_RANDOM_SEED)
02223     xml_node_set_uint (node, XML_SEED, input->seed);
02224
02225   // Setting the algorithm
02226   buffer = (char *) g_malloc (64);
02227   switch (input->algorithm)
02228     {
02229     case ALGORITHM_MONTE_CARLO:
02230       xmlSetProp (node, XML_ALGORITHM, XML_MONTE_CARLO);
02231       snprintf (buffer, 64, "%u", input->nsimulations);
02232       xmlSetProp (node, XML_NSIMULATIONS, (xmlChar *) buffer);
02233       snprintf (buffer, 64, "%u", input->niterations);
02234       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02235       snprintf (buffer, 64, "%.3lg", input->tolerance);
02236       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02237       snprintf (buffer, 64, "%u", input->nbest);
02238       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02239       break;
02240     case ALGORITHM_SWEEP:
02241       xmlSetProp (node, XML_ALGORITHM, XML_SWEEP);
02242       snprintf (buffer, 64, "%u", input->niterations);
02243       xmlSetProp (node, XML_NITERATIONS, (xmlChar *) buffer);
02244       snprintf (buffer, 64, "%.3lg", input->tolerance);
02245       xmlSetProp (node, XML_TOLERANCE, (xmlChar *) buffer);
02246       snprintf (buffer, 64, "%u", input->nbest);
02247       xmlSetProp (node, XML_NBEST, (xmlChar *) buffer);
02248       break;
02249     default:
02250       xmlSetProp (node, XML_ALGORITHM, XML_GENETIC);
02251       snprintf (buffer, 64, "%u", input->nsimulations);
02252       xmlSetProp (node, XML_NPOPULATION, (xmlChar *) buffer);
02253       snprintf (buffer, 64, "%u", input->niterations);
02254       xmlSetProp (node, XML_NGENERATIONS, (xmlChar *) buffer);
02255       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
02256       xmlSetProp (node, XML_MUTATION, (xmlChar *) buffer);
02257       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
02258       xmlSetProp (node, XML_REPRODUCTION, (xmlChar *) buffer);
02259       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
02260       xmlSetProp (node, XML_ADAPTATION, (xmlChar *) buffer);
02261       break;
02262     }
02263   g_free (buffer);
02264
02265   // Setting the experimental data
02266   for (i = 0; i < input->nexperiments; ++i)
02267     {
02268       child = xmlNewChild (node, 0, XML_EXPERIMENT, 0);
02269       xmlSetProp (child, XML_NAME, (xmlChar *) input->experiment[i]);
02270       if (input->weight[i] != 1.)
02271         xml_node_set_float (child, XML_WEIGHT, input->
      weight[i]);
02272       for (j = 0; j < input->ninputs; ++j)
02273         xmlSetProp (child, template[j], (xmlChar *) input->template[j][i]);
02274     }
02275
02276   // Setting the variables data
02277   for (i = 0; i < input->nvariables; ++i)
02278     {
02279       child = xmlNewChild (node, 0, XML_VARIABLE, 0);
02280       xmlSetProp (child, XML_NAME, (xmlChar *) input->label[i]);
02281       xml_node_set_float (child, XML_MINIMUM, input->
      rangemin[i]);
02282       if (input->rangeminabs[i] != -G_MAXDOUBLE)
02283         xml_node_set_float (child, XML_ABSOLUTE_MINIMUM,
      input->rangeminabs[i]);
02284       xml_node_set_float (child, XML_MAXIMUM, input->
      rangemax[i]);
02285       if (input->rangemaxabs[i] != G_MAXDOUBLE)
```

```
02286         xml_node_set_float (child, XML_ABSOLUTE_MAXIMUM,
      input->rangemaxabs[i]);
02287       if (input->precision[i] != DEFAULT_PRECISION)
02288         xml_node_set_uint (child, XML_PRECISION,
      input->precision[i]);
02289       if (input->algorithm == ALGORITHM_SWEEP)
02290         xml_node_set_uint (child, XML_NSWEEPS, input->
      nsweeps[i]);
02291       else if (input->algorithm == ALGORITHM_GENETIC)
02292         xml_node_set_uint (child, XML_NBITS, input->
      nbits[i]);
02293     }
02294
02295   // Saving the XML file
02296   xmlSaveFormatFile (filename, doc, 1);
02297
02298   // Freeing memory
02299   xmlFreeDoc (doc);
02300 }
```

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 2573 of file calibrator.c.

```
02574 {
02575   unsigned int i;
02576   for (i = 0; i < NALGORITHMS; ++i)
02577     if (gtk_toggle_button_get_active
02578         (GTK_TOGGLE_BUTTON (window->button_algorithm[i])))
02579       break;
02580   return i;
02581 }
```

5.7.2.4 int window_read (char * *filename*)

Function to read the input data of a file.

Parameters

filename	File name.

Returns

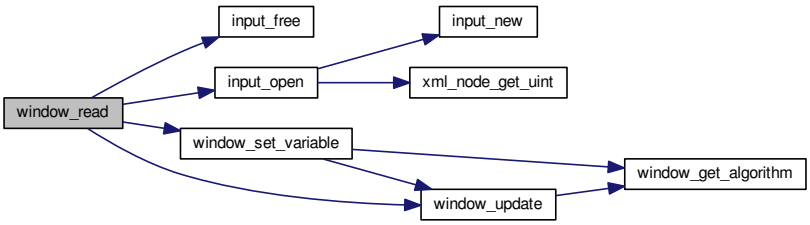
1 on succes, 0 on error.

Definition at line 3324 of file calibrator.c.

```
03325 {
03326   unsigned int i;
03327   char *buffer;
03328 #if DEBUG
03329   fprintf (stderr, "window_read: start\n");
03330 #endif
03331
03332   // Reading new input file
03333   input_free ();
03334   if (!input_open (filename))
03335     return 0;
03336
03337   // Setting GTK+ widgets data
03338   gtk_entry_set_text (window->entry_result, input->result);
03339   gtk_entry_set_text (window->entry_variables, input->
      variables);
03340   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
03341   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03342                                  (window->button_simulator), buffer);
03343   g_free (buffer);
03344   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
03345                                 (size_t) input->evaluator);
03346   if (input->evaluator)
03347     {
03348       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
03349       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
03350                                      (window->button_evaluator), buffer);
03351       g_free (buffer);
03352     }
03353   gtk_toggle_button_set_active
03354     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
03355   switch (input->algorithm)
03356     {
03357     case ALGORITHM_MONTE_CARLO:
03358       gtk_spin_button_set_value (window->spin_simulations,
03359                                  (gdouble) input->nsimulations);
03360     case ALGORITHM_SWEEP:
03361       gtk_spin_button_set_value (window->spin_iterations,
03362                                  (gdouble) input->niterations);
03363       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
03364       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
03365       break;
03366     default:
03367       gtk_spin_button_set_value (window->spin_population,
03368                                  (gdouble) input->nsimulations);
03369       gtk_spin_button_set_value (window->spin_generations,
03370                                  (gdouble) input->niterations);
03371       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
03372       gtk_spin_button_set_value (window->spin_reproduction,
03373                                  input->reproduction_ratio);
03374       gtk_spin_button_set_value (window->spin_adaptation,
03375                                  input->adaptation_ratio);
03376     }
03377   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
03378   g_signal_handler_block (window->button_experiment,
03379                           window->id_experiment_name);
03380   gtk_combo_box_text_remove_all (window->combo_experiment);
03381   for (i = 0; i < input->nexperiments; ++i)
03382     gtk_combo_box_text_append_text (window->combo_experiment,
03383                                     input->experiment[i]);
03384   g_signal_handler_unblock
03385     (window->button_experiment, window->
      id_experiment_name);
03386   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
03387   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
03388   g_signal_handler_block (window->combo_variable, window->
      id_variable);
```

```
03389   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
03390   gtk_combo_box_text_remove_all (window->combo_variable);
03391   for (i = 0; i < input->nvariables; ++i)
03392     gtk_combo_box_text_append_text (window->combo_variable,
      input->label[i]);
03393   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
03394   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
03395   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
03396   window_set_variable ();
03397   window_update ();
03398
03399 #if DEBUG
03400   fprintf (stderr, "window_read: end\n");
03401 #endif
03402   return 1;
03403 }
```

Here is the call graph for this function:



5.7.2.5 int window_save ( )

Function to save the input file.

Returns

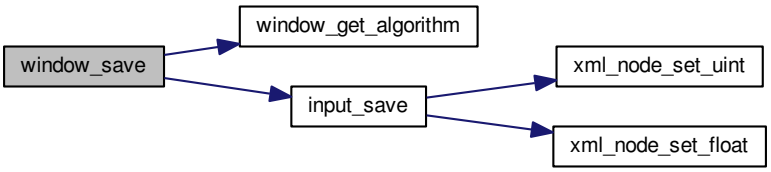
1 on OK, 0 on Cancel.

Definition at line 2381 of file calibrator.c.

```
02382 {
02383   char *buffer;
02384   GtkFileChooserDialog *dlg;
02385
02386 #if DEBUG
02387   fprintf (stderr, "window_save: start\n");
02388 #endif
02389
02390   // Opening the saving dialog
02391   dlg = (GtkFileChooserDialog *)
02392     gtk_file_chooser_dialog_new (gettext ("Save file"),
02393                                  window->window,
02394                                  GTK_FILE_CHOOSER_ACTION_SAVE,
02395                                  gettext ("_Cancel"),
02396                                  GTK_RESPONSE_CANCEL,
02397                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02398   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
02399   buffer = g_build_filename (input->directory, input->name, NULL);
02400   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
02401   g_free (buffer);
02402
02403   // If OK response then saving
02404   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02405     {
02406
02407       // Adding properties to the root XML node
```

```
02408       input->simulator = gtk_file_chooser_get_filename
02409         (GTK_FILE_CHOOSER (window->button_simulator));
02410       if (gtk_toggle_button_get_active
02411           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
02412         input->evaluator = gtk_file_chooser_get_filename
02413           (GTK_FILE_CHOOSER (window->button_evaluator));
02414       else
02415         input->evaluator = NULL;
02416       input->result
02417         = (char *) xmlStrdup ((const xmlChar *)
02418                               gtk_entry_get_text (window->entry_result));
02419       input->variables
02420         = (char *) xmlStrdup ((const xmlChar *)
02421                               gtk_entry_get_text (window->entry_variables));
02422
02423       // Setting the algorithm
02424       switch (window_get_algorithm ())
02425         {
02426         case ALGORITHM_MONTE_CARLO:
02427           input->algorithm = ALGORITHM_MONTE_CARLO;
02428           input->nsimulations
02429             = gtk_spin_button_get_value_as_int (window->spin_simulations);
02430           input->niterations
02431             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02432           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02433           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02434           break;
02435         case ALGORITHM_SWEEP:
02436           input->algorithm = ALGORITHM_SWEEP;
02437           input->niterations
02438             = gtk_spin_button_get_value_as_int (window->spin_iterations);
02439           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
02440           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
02441           break;
02442         default:
02443           input->algorithm = ALGORITHM_GENETIC;
02444           input->nsimulations
02445             = gtk_spin_button_get_value_as_int (window->spin_population);
02446           input->niterations
02447             = gtk_spin_button_get_value_as_int (window->spin_generations);
02448           input->mutation_ratio
02449             = gtk_spin_button_get_value (window->spin_mutation);
02450           input->reproduction_ratio
02451             = gtk_spin_button_get_value (window->spin_reproduction);
02452           input->adaptation_ratio
02453             = gtk_spin_button_get_value (window->spin_adaptation);
02454           break;
02455         }
02456
02457       // Saving the XML file
02458       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02459       input_save (buffer);
02460
02461       // Closing and freeing memory
02462       g_free (buffer);
02463       gtk_widget_destroy (GTK_WIDGET (dlg));
02464 #if DEBUG
02465       fprintf (stderr, "window_save: end\n");
02466 #endif
02467       return 1;
02468     }
02469
02470   // Closing and freeing memory
02471   gtk_widget_destroy (GTK_WIDGET (dlg));
02472 #if DEBUG
02473   fprintf (stderr, "window_save: end\n");
02474 #endif
02475   return 0;
02476 }
```

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 2960 of file calibrator.c.

```
02961 {
02962   unsigned int i, j;
02963   char *buffer;
02964   GFile *file1, *file2;
02965 #if DEBUG
02966   fprintf (stderr, "window_template_experiment: start\n");
02967 #endif
02968   i = (size_t) data;
02969   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
02970   file1
02971     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
02972   file2 = g_file_new_for_path (input->directory);
02973   buffer = g_file_get_relative_path (file2, file1);
02974   input->template[i][j] = (char *) xmlStrdup ((xmlChar *) buffer);
02975   g_free (buffer);
02976   g_object_unref (file2);
02977   g_object_unref (file1);
02978 #if DEBUG
02979   fprintf (stderr, "window_template_experiment: end\n");
02980 #endif
02981 }
```

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   GtkGrid *grid_files;
00113   GtkLabel *label_simulator;
00114   GtkFileChooserButton *button_simulator;
00116   GtkCheckButton *check_evaluator;
00117   GtkFileChooserButton *button_evaluator;
00119   GtkLabel *label_result;
00120   GtkEntry *entry_result;
00121   GtkLabel *label_variables;
00122   GtkEntry *entry_variables;
00123   GtkFrame *frame_algorithm;
00124   GtkGrid *grid_algorithm;
00125   GtkRadioButton *button_algorithm[NALGORITHMS];
00127   GtkLabel *label_simulations;
00128   GtkSpinButton *spin_simulations;
00130   GtkLabel *label_iterations;
00131   GtkSpinButton *spin_iterations;
00133   GtkLabel *label_tolerance;
00134   GtkSpinButton *spin_tolerance;
00135   GtkLabel *label_bests;
00136   GtkSpinButton *spin_bests;
00137   GtkLabel *label_population;
00138   GtkSpinButton *spin_population;
00140   GtkLabel *label_generations;
00141   GtkSpinButton *spin_generations;
00143   GtkLabel *label_mutation;
00144   GtkSpinButton *spin_mutation;
00145   GtkLabel *label_reproduction;
00146   GtkSpinButton *spin_reproduction;
```

```
00148   GtkLabel *label_adaptation;
00149   GtkSpinButton *spin_adaptation;
00151   GtkFrame *frame_variable;
00152   GtkGrid *grid_variable;
00153   GtkComboBoxText *combo_variable;
00155   GtkButton *button_add_variable;
00156   GtkButton *button_remove_variable;
00157   GtkLabel *label_variable;
00158   GtkEntry *entry_variable;
00159   GtkLabel *label_min;
00160   GtkSpinButton *spin_min;
00161   GtkScrolledWindow *scrolled_min;
00162   GtkLabel *label_max;
00163   GtkSpinButton *spin_max;
00164   GtkScrolledWindow *scrolled_max;
00165   GtkCheckButton *check_minabs;
00166   GtkSpinButton *spin_minabs;
00167   GtkScrolledWindow *scrolled_minabs;
00168   GtkCheckButton *check_maxabs;
00169   GtkSpinButton *spin_maxabs;
00170   GtkScrolledWindow *scrolled_maxabs;
00171   GtkLabel *label_precision;
00172   GtkSpinButton *spin_precision;
00173   GtkLabel *label_sweeps;
00174   GtkSpinButton *spin_sweeps;
00175   GtkLabel *label_bits;
00176   GtkSpinButton *spin_bits;
00177   GtkFrame *frame_experiment;
00178   GtkGrid *grid_experiment;
00179   GtkComboBoxText *combo_experiment;
00180   GtkButton *button_add_experiment;
00181   GtkButton *button_remove_experiment;
00182   GtkLabel *label_experiment;
00183   GtkFileChooserButton *button_experiment;
00185   GtkLabel *label_weight;
00186   GtkSpinButton *spin_weight;
00187   GtkCheckButton *check_template[MAX_NINPUTS];
00189   GtkFileChooserButton *button_template[MAX_NINPUTS];
00191   GdkPixbuf *logo;
00192   Experiment *experiment;
00193   Variable *variable;
00194   char *application_directory;
00195   gulong id_experiment;
00196   gulong id_experiment_name;
00197   gulong id_variable;
00198   gulong id_variable_label;
00199   gulong id_template[MAX_NINPUTS];
00201   gulong id_input[MAX_NINPUTS];
00203   unsigned int nexperiments;
00204   unsigned int nvariables;
00205 } Window;
00206
00207 // Public functions
00208 void input_save (char *filename);
00209 void options_new ();
00210 void running_new ();
00211 int window_save ();
00212 void window_run ();
00213 void window_help ();
00214 int window_get_algorithm ();
00215 void window_update ();
00216 void window_set_algorithm ();
00217 void window_set_experiment ();
00218 void window_remove_experiment ();
00219 void window_add_experiment ();
00220 void window_name_experiment ();
00221 void window_weight_experiment ();
00222 void window_inputs_experiment ();
00223 void window_template_experiment (void *data);
00224 void window_set_variable ();
00225 void window_remove_variable ();
00226 void window_add_variable ();
00227 void window_label_variable ();
00228 void window_precision_variable ();
00229 void window_rangemin_variable ();
00230 void window_rangemax_variable ();
00231 void window_rangeminabs_variable ();
00232 void window_rangemaxabs_variable ();
00233 void window_update_variable ();
00234 int window_read (char *filename);
00235 void window_open ();
00236 void window_new ();
00237 int cores_number ();
00238
00239 #endif
```

Index