MPCOTool

4.0.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 Experiment Struct Reference

Struct to define the experiment data.

```
#include <experiment.h>
```

Data Fields

- char ∗ name

  File name.
- char ∗ stencil [MAX_NINPUTS]

  Array of template names of input files.
- double weight

  Objective function weight.
- unsigned int ninputs

  Number of input files to the simulator.

3.1.1 Detailed Description

Struct to define the experiment data.

Definition at line 45 of file experiment.h.

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

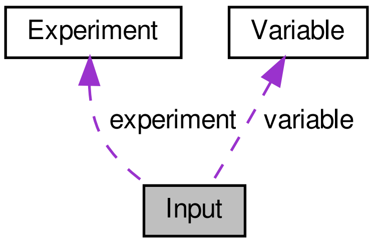
- experiment.h

3.2 Input Struct Reference

Struct to define the optimization input file.

```
#include <input.h>
```

Collaboration diagram for Input:



Data Fields

- Experiment ∗ experiment

    Array or experiments.
- Variable ∗ variable

    Array of variables.
- 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 ∗ directory

    Working directory.
- char ∗ name

    Input data file name.
- double tolerance

    Algorithm tolerance.
- double mutation_ratio

    Mutation probability.
- double reproduction_ratio

    Reproduction probability.
- double adaptation_ratio

    Adaptation probability.
- double relaxation

    Relaxation parameter.

- double p

  Exponent of the P error norm.
- double threshold

  Threshold to finish the optimization.
- unsigned long int seed

  Seed of the pseudo-random numbers generator.
- unsigned int nvariables

  Variables number.
- unsigned int nexperiments

  Experiments number.
- unsigned int nsimulations

  Simulations number per experiment.
- unsigned int algorithm

  Algorithm type.
- unsigned int nsteps

  Number of steps to do the hill climbing method.
- unsigned int climbing

  Method to estimate the hill climbing.
- unsigned int nestimates

  Number of simulations to estimate the hill climbing.
- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int norm

  Error norm type.
- unsigned int type

  Type of input file.

3.2.1 Detailed Description

Struct to define the optimization input file.

Definition at line 65 of file input.h.

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

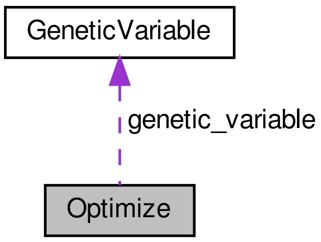
- input.h

3.3 Optimize Struct Reference

Struct to define the optimization ation data.

```
#include <optimize.h>
```

Collaboration diagram for Optimize:



Data Fields

- GMappedFile ∗∗ file [MAX_NINPUTS]

  Matrix of input template files.
- char ∗∗ experiment

  Array of experimental data file names.
- char ∗∗ label

  Array of variable names.
- gsl_rng ∗ rng

  GSL random number generator.
- **GeneticVariable** ∗ genetic_variable

  Array of variables for the genetic algorithm.
- FILE ∗ file_result

  Result file.
- FILE ∗ file_variables

  Variables file.
- 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.
- 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 ∗ step

    Array of hill climbing method step sizes.
- double ∗ climbing

    Vector of hill climbing estimation.
- 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.
- unsigned int ∗ precision

    Array of variable precisions.
- unsigned int ∗ nsweeps

    Array of sweeps of the sweep algorithm.
- unsigned int ∗ nbits

    Array of bits number of the genetic algorithm.
- unsigned int ∗ thread

    Array of simulation numbers to calculate on the thread.
- unsigned int ∗ thread_climbing
- unsigned int ∗ simulation_best

    Array of best simulation numbers.
- double tolerance

    Algorithm tolerance.
- double mutation_ratio

    Mutation probability.
- double reproduction_ratio

    Reproduction probability.
- double adaptation_ratio

    Adaptation probability.
- double relaxation

    Relaxation parameter.
- double calculation_time

    Calculation time.
- double p

    Exponent of the P error norm.
- double threshold

    Threshold to finish the optimization.
- 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 nsteps

    Number of steps for the hill climbing method.

- unsigned int nestimates

  Number of simulations to estimate the climbing.
- unsigned int algorithm

  Algorithm type.
- unsigned int nstart

  Beginning simulation number of the task.
- unsigned int nend

  Ending simulation number of the task.
- unsigned int nstart_climbing

  Beginning simulation number of the task for the hill climbing method.
- unsigned int nend_climbing

  Ending simulation number of the task for the hill climbing method.
- unsigned int niterations

  Number of algorithm iterations.
- unsigned int nbest

  Number of best simulations.
- unsigned int nsaveds

  Number of saved simulations.
- unsigned int stop

  To stop the simulations.
- int mpi_rank

  Number of MPI task.

3.3.1 Detailed Description

Struct to define the optimization ation data.

Definition at line 45 of file optimize.h.

3.3.2 Field Documentation

3.3.2.1 thread_climbing

```
unsigned int* Optimize::thread_climbing
```

Array of simulation numbers to calculate on the thread for the hill climbing method.

Definition at line 79 of file optimize.h.

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

- optimize.h

3.4 Options Struct Reference

Struct to define the options dialog.

```
#include <interface.h>
```

Data Fields

- GtkDialog ∗ dialog

  Main GtkDialog.
- GtkGrid ∗ grid

  Main GtkGrid.
- GtkLabel ∗ label_seed

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

  Pseudo-random numbers generator seed GtkSpinButton.
- GtkLabel ∗ label_threads

  Threads number GtkLabel.
- GtkSpinButton ∗ spin_threads

  Threads number GtkSpinButton.
- GtkLabel ∗ label_climbing

  Climbing threads number GtkLabel.
- GtkSpinButton ∗ spin_climbing

  Climbing threads number GtkSpinButton.

3.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 48 of file interface.h.

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

- interface.h

3.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

```
#include <optimize.h>
```

Data Fields

- unsigned int thread

  Thread number.

3.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 121 of file optimize.h.

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

- optimize.h

3.6 Running Struct Reference

Struct to define the running dialog.

```
#include <interface.h>
```

Data Fields

- GtkDialog ∗ dialog

    Main GtkDialog.
- GtkLabel ∗ label

    Label GtkLabel.
- GtkSpinner ∗ spinner

    Animation GtkSpinner.
- GtkGrid ∗ grid

    Grid GtkGrid.

3.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 66 of file interface.h.

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

- interface.h

3.7 Variable Struct Reference

Struct to define the variable data.

```
#include <variable.h>
```

Data Fields

- char ∗ name

  Variable name.
- double rangemin

  Minimum variable value.
- double rangemax

  Maximum variable value.
- double rangeminabs

  Absolute minimum variable value.
- double rangemaxabs

  Absolute maximum variable value.
- double step

  Hill climbing method step size.
- unsigned int precision

  Variable precision.
- unsigned int nsweeps

  Sweeps of the sweep algorithm.
- unsigned int nbits

  Bits number of the genetic algorithm.

3.7.1 Detailed Description

Struct to define the variable data.

Definition at line 54 of file variable.h.

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

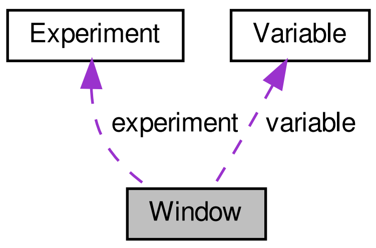
- variable.h

3.8 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_norm

    GtkFrame to set the error norm.
- GtkGrid ∗ grid_norm

    GtkGrid to set the error norm.
- GtkRadioButton ∗ button_norm [NNORMS]

    Array of GtkButtons to set the error norm.
- GtkLabel ∗ label_p

    GtkLabel to set the p parameter.
- GtkSpinButton ∗ spin_p

    GtkSpinButton to set the p parameter.
- GtkScrolledWindow ∗ scrolled_p

GtkScrolledWindow to set the p parameter.

- 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.
- GtkCheckButton ∗ check_climbing

  GtkCheckButton to check running the hill climbing method.
- GtkGrid ∗ grid_climbing

  GtkGrid to pack the hill climbing method widgets.
- GtkRadioButton ∗ button_climbing [NCLIMBINGS]

  GtkRadioButtons array to set the hill climbing method.
- GtkLabel ∗ label_steps

  GtkLabel to set the steps number.

- GtkSpinButton ∗ spin_steps

  GtkSpinButton to set the steps number.
- GtkLabel ∗ label_estimates

  GtkLabel to set the estimates number.
- GtkSpinButton ∗ spin_estimates

  GtkSpinButton to set the estimates number.
- GtkLabel ∗ label_relaxation

  GtkLabel to set the relaxation parameter.
- GtkSpinButton ∗ spin_relaxation

  GtkSpinButton to set the relaxation parameter.
- GtkLabel ∗ label_threshold

  GtkLabel to set the threshold.
- GtkSpinButton ∗ spin_threshold

  GtkSpinButton to set the threshold.
- GtkScrolledWindow ∗ scrolled_threshold

  GtkScrolledWindow to set the threshold.
- 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.
- GtkLabel ∗ label_step

  GtkLabel to set the step.
- GtkSpinButton ∗ spin_step

  GtkSpinButton to set the step.
- GtkScrolledWindow ∗ scrolled_step

  step GtkScrolledWindow.
- 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.

3.8.1 Detailed Description

Struct to define the main window.

Definition at line 78 of file interface.h.

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

- interface.h

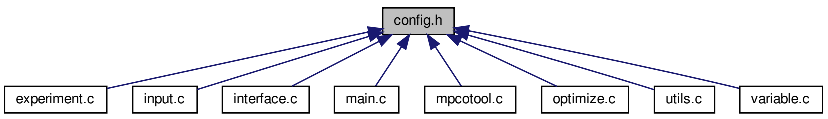
Chapter 4

File Documentation

4.1 config.h File Reference

Configuration header file.

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



Macros

- #define _(string) (gettext(string))
- #define MAX_NINPUTS 8

  Maximum number of input files in the simulator program.
- #define NALGORITHMS 4

  Number of stochastic algorithms.
- #define NCLIMBINGS 2

  Number of hill climbing estimate methods.
- #define NNORMS 4

  Number of error norms.
- #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 DEFAULT_RELAXATION 1.

  Default relaxation parameter.
- #define LOCALE_DIR "locales"

Locales directory.
- #define PROGRAM_INTERFACE "mpcotool"

  Name of the interface program.
- #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"

  absolute minimum label.
- #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"

  absolute maximum label.
- #define LABEL_ADAPTATION "adaptation"

  adaption label.
- #define LABEL_ALGORITHM "algorithm"

  algoritm label.
- #define LABEL_CLIMBING "climbing"

  climbing label.
- #define LABEL_COORDINATES "coordinates"

  coordinates label.
- #define LABEL_EUCLIDIAN "euclidian"

  euclidian label.
- #define LABEL_EVALUATOR "evaluator"

  evaluator label.
- #define LABEL_EXPERIMENT "experiment"

  experiment label.
- #define LABEL_EXPERIMENTS "experiments"

  experiment label.
- #define LABEL_GENETIC "genetic"

  genetic label.
- #define LABEL_MINIMUM "minimum"

  minimum label.
- #define LABEL_MAXIMUM "maximum"

  maximum label.
- #define LABEL_MONTE_CARLO "Monte-Carlo"

  Monte-Carlo label.
- #define LABEL_MUTATION "mutation"

  mutation label.
- #define LABEL_NAME "name"

  name label.
- #define LABEL_NBEST "nbest"

  nbest label.
- #define LABEL_NBITS "nbits"

  nbits label.
- #define LABEL_NESTIMATES "nestimates"

  nestimates label.
- #define LABEL_NGENERATIONS "ngenerations"

  ngenerations label.
- #define LABEL_NITERATIONS "niterations"

  niterations label.
- #define LABEL_NORM "norm"

  norm label.
- #define LABEL_NPOPULATION "npopulation"

  npopulation label.
- #define LABEL_NSIMULATIONS "nsimulations"

  nsimulations label.

- #define LABEL_NSTEPS "nsteps"

  nsteps label.
- #define LABEL_NSWEEPS "nsweeps"

  nsweeps label.
- #define LABEL_OPTIMIZE "optimize"

  optimize label.
- #define LABEL_ORTHOGONAL "orthogonal"

  orthogonal label.
- #define LABEL_P "p"

  p label.
- #define LABEL_PRECISION "precision"

  precision label.
- #define LABEL_RANDOM "random"

  random label.
- #define LABEL_RELAXATION "relaxation"

  relaxation label.
- #define LABEL_REPRODUCTION "reproduction"

  reproduction label.
- #define LABEL_RESULT_FILE "result_file"

  result_file label.
- #define LABEL_SIMULATOR "simulator"

  simulator label.
- #define LABEL_SEED "seed"

  seed label.
- #define LABEL_STEP "step"

  step label.
- #define LABEL_SWEEP "sweep"

  sweep label.
- #define LABEL_TAXICAB "taxicab"

  taxicab label.
- #define LABEL_TEMPLATE1 "template1"

  template1 label.
- #define LABEL_TEMPLATE2 "template2"

  template2 label.
- #define LABEL_TEMPLATE3 "template3"

  template3 label.
- #define LABEL_TEMPLATE4 "template4"

  template4 label.
- #define LABEL_TEMPLATE5 "template5"

  template5 label.
- #define LABEL_TEMPLATE6 "template6"

  template6 label.
- #define LABEL_TEMPLATE7 "template7"

  template7 label.
- #define LABEL_TEMPLATE8 "template8"

  template8 label.
- #define LABEL_THRESHOLD "threshold"

  threshold label.
- #define LABEL_TOLERANCE "tolerance"

  tolerance label.
- #define LABEL_VARIABLE "variable"

variable label.
- #define LABEL_VARIABLES "variables"

  variables label.
- #define LABEL_VARIABLES_FILE "variables_file"

  variables label.
- #define LABEL_WEIGHT "weight"

  weight label.

Enumerations

- enum INPUT_TYPE { INPUT_TYPE_XML = 0, INPUT_TYPE_JSON = 1 }

  Enum to define the input file types.

4.1.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file config.h.

4.1.2 Enumeration Type Documentation

4.1.2.1 INPUT_TYPE

```
enum INPUT_TYPE
```

Enum to define the input file types.

Enumerator

INPUT_TYPE_XML	XML input file.
INPUT_TYPE_JSON	JSON input file.

Definition at line 126 of file config.h.

```
{
  INPUT_TYPE_XML = 0,
  INPUT_TYPE_JSON = 1
};
```

4.2 config.h

```
/* config.h.  Generated from config.h.in by configure.  */
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef CONFIG__H
#define CONFIG__H 1

// Gettext simplification
#define _(string) (gettext(string))

// Array sizes

#define MAX_NINPUTS 8
#define NALGORITHMS 4
#define NCLIMBINGS 2
#define NNORMS 4
#define NPRECISIONS 15

// Default choices

#define DEFAULT_PRECISION (NPRECISIONS - 1)
#define DEFAULT_RANDOM_SEED 7007
#define DEFAULT_RELAXATION 1.

// Interface labels

#define LOCALE_DIR "locales"
#define PROGRAM_INTERFACE "mpcotool"

// Labels

#define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
#define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
#define LABEL_ADAPTATION "adaptation"
#define LABEL_ALGORITHM "algorithm"
#define LABEL_CLIMBING "climbing"
#define LABEL_COORDINATES "coordinates"
#define LABEL_EUCLIDIAN "euclidian"
#define LABEL_EVALUATOR "evaluator"
#define LABEL_EXPERIMENT "experiment"
#define LABEL_EXPERIMENTS "experiments"
#define LABEL_GENETIC "genetic"
#define LABEL_MINIMUM "minimum"
#define LABEL_MAXIMUM "maximum"
#define LABEL_MONTE_CARLO "Monte-Carlo"
#define LABEL_MUTATION "mutation"
#define LABEL_NAME "name"
```

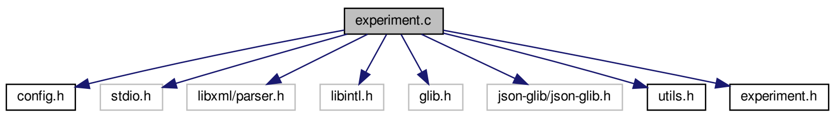
```
#define LABEL_NBEST "nbest"
#define LABEL_NBITS "nbits"
#define LABEL_NESTIMATES "nestimates"
#define LABEL_NGENERATIONS "ngenerations"
#define LABEL_NITERATIONS "niterations"
#define LABEL_NORM "norm"
#define LABEL_NPOPULATION "npopulation"
#define LABEL_NSIMULATIONS "nsimulations"
#define LABEL_NSTEPS "nsteps"
#define LABEL_NSWEEPS "nsweeps"
#define LABEL_OPTIMIZE "optimize"
#define LABEL_ORTHOGONAL "orthogonal"
#define LABEL_P "p"
#define LABEL_PRECISION "precision"
#define LABEL_RANDOM "random"
#define LABEL_RELAXATION "relaxation"
#define LABEL_REPRODUCTION "reproduction"
#define LABEL_RESULT_FILE "result_file"
#define LABEL_SIMULATOR "simulator"
#define LABEL_SEED "seed"
#define LABEL_STEP "step"
#define LABEL_SWEEP "sweep"
#define LABEL_TAXICAB "taxicab"
#define LABEL_TEMPLATE1 "template1"
#define LABEL_TEMPLATE2 "template2"
#define LABEL_TEMPLATE3 "template3"
#define LABEL_TEMPLATE4 "template4"
#define LABEL_TEMPLATE5 "template5"
#define LABEL_TEMPLATE6 "template6"
#define LABEL_TEMPLATE7 "template7"
#define LABEL_TEMPLATE8 "template8"
#define LABEL_THRESHOLD "threshold"
#define LABEL_TOLERANCE "tolerance"
#define LABEL_VARIABLE "variable"
#define LABEL_VARIABLES "variables"
#define LABEL_VARIABLES_FILE "variables_file"
#define LABEL_WEIGHT "weight"

// Enumerations

enum INPUT_TYPE
{
  INPUT_TYPE_XML = 0,
  INPUT_TYPE_JSON = 1
};

#endif
```

4.3 experiment.c File Reference

Source file to define the experiment data.

```
#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"
```

Include dependency graph for experiment.c:



Macros

- #define DEBUG_EXPERIMENT 0

  Macro to debug experiment functions.

Functions

- void experiment_new (Experiment ∗experiment)
- void experiment_free (Experiment ∗experiment, unsigned int type)
- void experiment_error (Experiment ∗experiment, char ∗message)
- int experiment_open_xml (Experiment ∗experiment, xmlNode ∗node, unsigned int ninputs)
- int experiment_open_json (Experiment ∗experiment, JsonNode ∗node, unsigned int ninputs)

Variables

- const char ∗ stencil [MAX_NINPUTS]

  Array of xmlChar strings with stencil labels.

4.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file experiment.c.

4.3.2 Function Documentation

4.3.2.1 experiment_error()

```
void experiment_error (
            Experiment ∗ experiment,
            char ∗ message )
```

Function to print a message error opening an Experiment struct.

Parameters

experiment	Experiment struct.
message	Error message.

Definition at line 109 of file experiment.c.

```
{
  char buffer[64];
  if (!experiment->name)
    snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
              experiment->name, message);
  error_message = g_strdup (buffer);
}
```

4.3.2.2 experiment_free()

```
void experiment_free (
            Experiment * experiment,
            unsigned int type )
```

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.
type	Type of input file.

Definition at line 80 of file experiment.c.

```
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    {
      for (i = 0; i < experiment->ninputs; ++i)
        xmlFree (experiment->stencil[i]);
      xmlFree (experiment->name);
    }
  else
    {
      for (i = 0; i < experiment->ninputs; ++i)
        g_free (experiment->stencil[i]);
      g_free (experiment->name);
    }
  experiment->ninputs = 0;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: end\n");
#endif
}
```

4.3.2.3 experiment_new()

```
void experiment_new (
            Experiment * experiment )
```

Function to create a new Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 61 of file experiment.c.

```
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_new: start\n");
#endif
  experiment->name = NULL;
  experiment->ninputs = 0;
  for (i = 0; i < MAX_NINPUTS; ++i)
    experiment->stencil[i] = NULL;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "input_new: end\n");
#endif
}
```

4.3.2.4 experiment_open_json()

```
int experiment_open_json (
            Experiment * experiment,
            JsonNode * node,
            unsigned int ninputs )
```

Function to open the Experiment struct on a XML node.

Returns

1 on success, 0 on error.

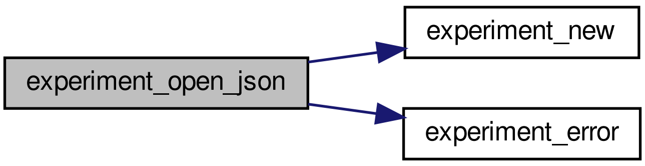
Parameters

experiment	Experiment struct.
node	JSON node.
ninputs	Number of the simulator input files.

Definition at line 231 of file experiment.c.

```
{
  char buffer[64];
  JsonObject *object;
  const char *name;
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Getting JSON object
  object = json_node_get_object (node);

  // Reading the experimental data
  name = json_object_get_string_member (object, LABEL_NAME);
  if (!name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
  experiment->name = g_strdup (name);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
#endif
  experiment->weight
    = json_object_get_float_with_default (object,
LABEL_WEIGHT, 1.,
                                          &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
#endif
  name = json_object_get_string_member (object, stencil[0]);
  if (name)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
               name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  experiment->stencil[0] = g_strdup (name);
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
#endif
      if (json_object_get_member (object, stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          name = json_object_get_string_member (object, stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_json: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, name, stencil[i]);
#endif
          experiment->stencil[i] = g_strdup (name);
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_JSON);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.3.2.5 experiment_open_xml()

```
int experiment_open_xml (
            Experiment * experiment,
            xmlNode * node,
            unsigned int ninputs )
```

Function to open the Experiment struct on a XML node.

Returns

1 on success, 0 on error.

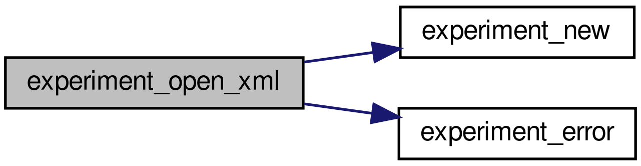
Parameters

experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 127 of file experiment.c.

```
{
  char buffer[64];
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Reading the experimental data
  experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!experiment->name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
#endif
  experiment->weight
    =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
                                     &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
#endif
  experiment->stencil[0]
    = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
  if (experiment->stencil[0])
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
               experiment->name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
#endif
      if (xmlHasProp (node, (const xmlChar *) stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          experiment->stencil[i]
            = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_xml: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, experiment->name,
                   experiment->stencil[i]);
#endif
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_XML);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.3.3 Variable Documentation

4.3.3.1 stencil

```
const char* stencil[MAX_NINPUTS]
```

Initial value:

```
= {
  LABEL_TEMPLATE1, LABEL_TEMPLATE2,
      LABEL_TEMPLATE3, LABEL_TEMPLATE4,
  LABEL_TEMPLATE5, LABEL_TEMPLATE6,
      LABEL_TEMPLATE7, LABEL_TEMPLATE8
}
```

Array of xmlChar strings with stencil labels.

Definition at line 50 of file experiment.c.

4.4 experiment.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"

#define DEBUG_EXPERIMENT 0

const char *stencil[MAX_NINPUTS] = {
  LABEL_TEMPLATE1, LABEL_TEMPLATE2,
LABEL_TEMPLATE3, LABEL_TEMPLATE4,
  LABEL_TEMPLATE5, LABEL_TEMPLATE6,
LABEL_TEMPLATE7, LABEL_TEMPLATE8
};



void
experiment_new (Experiment * experiment)
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_new: start\n");
#endif
  experiment->name = NULL;
  experiment->ninputs = 0;
  for (i = 0; i < MAX_NINPUTS; ++i)
    experiment->stencil[i] = NULL;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "input_new: end\n");
#endif
}

void
experiment_free (Experiment * experiment,
                 unsigned int type)
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    {
      for (i = 0; i < experiment->ninputs; ++i)
        xmlFree (experiment->stencil[i]);
      xmlFree (experiment->name);
    }
  else
    {
      for (i = 0; i < experiment->ninputs; ++i)
        g_free (experiment->stencil[i]);
      g_free (experiment->name);
    }
  experiment->ninputs = 0;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: end\n");
#endif
}

void
experiment_error (Experiment * experiment,
                  char *message)
{
  char buffer[64];
  if (!experiment->name)
    snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
              experiment->name, message);
  error_message = g_strdup (buffer);
}

int
experiment_open_xml (Experiment * experiment,
                         xmlNode * node,
                         unsigned int ninputs)
{
  char buffer[64];
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Reading the experimental data
  experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!experiment->name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
#endif
  experiment->weight
    =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
                                     &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
#endif
  experiment->stencil[0]
    = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
  if (experiment->stencil[0])
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
               experiment->name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
#endif
      if (xmlHasProp (node, (const xmlChar *) stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          experiment->stencil[i]
            = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_xml: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, experiment->name,
                   experiment->stencil[i]);
#endif
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
```

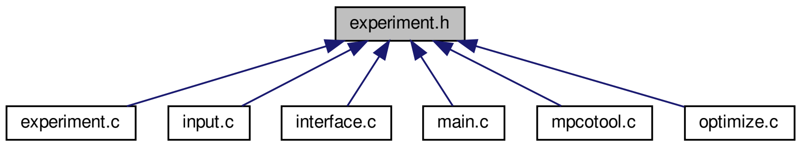
```c
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_XML);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
  return 0;
}

int
experiment_open_json (Experiment * experiment,
                      JsonNode * node,
                      unsigned int ninputs)
{
  char buffer[64];
  JsonObject *object;
  const char *name;
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Getting JSON object
  object = json_node_get_object (node);

  // Reading the experimental data
  name = json_object_get_string_member (object, LABEL_NAME);
  if (!name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
  experiment->name = g_strdup (name);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
#endif
  experiment->weight
    = json_object_get_float_with_default (object,
LABEL_WEIGHT, 1.,
                                          &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
#endif
  name = json_object_get_string_member (object, stencil[0]);
  if (name)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
               name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  experiment->stencil[0] = g_strdup (name);
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
#endif
      if (json_object_get_member (object, stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          name = json_object_get_string_member (object, stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_json: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, name, stencil[i]);
#endif
```

```
          experiment->stencil[i] = g_strdup (name);
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_JSON);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 0;
}
```

4.5 experiment.h File Reference

Header file to define the experiment data.

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



Data Structures

- struct Experiment

    Struct to define the experiment data.

Functions

- void experiment_new (Experiment ∗experiment)
- void experiment_free (Experiment ∗experiment, unsigned int type)
- void experiment_error (Experiment ∗experiment, char ∗message)
- int experiment_open_xml (Experiment ∗experiment, xmlNode ∗node, unsigned int ninputs)
- int experiment_open_json (Experiment ∗experiment, JsonNode ∗node, unsigned int ninputs)

Variables

- const char ∗ stencil [MAX_NINPUTS]

    Array of xmlChar strings with stencil labels.

4.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file experiment.h.

4.5.2 Function Documentation

4.5.2.1 experiment_error()

```
void experiment_error (
            Experiment * experiment,
            char * message )
```

Function to print a message error opening an Experiment struct.

Parameters

experiment	Experiment struct.
message	Error message.

Definition at line 109 of file experiment.c.

```
{
  char buffer[64];
  if (!experiment->name)
    snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
              experiment->name, message);
  error_message = g_strdup (buffer);
}
```

4.5.2.2 experiment_free()

```
void experiment_free (
            Experiment * experiment,
            unsigned int type )
```

Function to free the memory of an Experiment struct.

Parameters

experiment	Experiment struct.
type	Type of input file.

Definition at line 80 of file experiment.c.

```
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    {
      for (i = 0; i < experiment->ninputs; ++i)
        xmlFree (experiment->stencil[i]);
      xmlFree (experiment->name);
    }
  else
    {
      for (i = 0; i < experiment->ninputs; ++i)
        g_free (experiment->stencil[i]);
      g_free (experiment->name);
    }
  experiment->ninputs = 0;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_free: end\n");
#endif
}
```

4.5.2.3 experiment_new()

```
void experiment_new (
              Experiment * experiment )
```

Function to create a new Experiment struct.

Parameters

experiment	Experiment struct.

Definition at line 61 of file experiment.c.

```
{
  unsigned int i;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_new: start\n");
#endif
  experiment->name = NULL;
  experiment->ninputs = 0;
  for (i = 0; i < MAX_NINPUTS; ++i)
    experiment->stencil[i] = NULL;
#if DEBUG_EXPERIMENT
  fprintf (stderr, "input_new: end\n");
#endif
}
```

4.5.2.4 experiment_open_json()

```
int experiment_open_json (
            Experiment * experiment,
            JsonNode * node,
            unsigned int ninputs )
```

Function to open the Experiment struct on a XML node.

Returns

1 on success, 0 on error.

Parameters

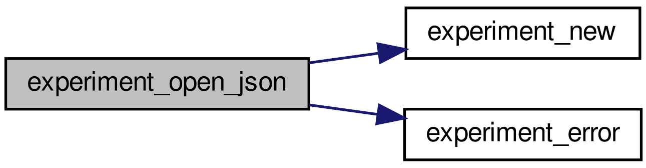
experiment	Experiment struct.
node	JSON node.
ninputs	Number of the simulator input files.

Definition at line 231 of file experiment.c.

```
{
  char buffer[64];
  JsonObject *object;
  const char *name;
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Getting JSON object
  object = json_node_get_object (node);

  // Reading the experimental data
  name = json_object_get_string_member (object, LABEL_NAME);
  if (!name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
  experiment->name = g_strdup (name);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
#endif
  experiment->weight
    = json_object_get_float_with_default (object,
LABEL_WEIGHT, 1.,
                                                   &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
#endif
  name = json_object_get_string_member (object, stencil[0]);
  if (name)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
               name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
```

```
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  experiment->stencil[0] = g_strdup (name);
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
#endif
      if (json_object_get_member (object, stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          name = json_object_get_string_member (object, stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_json: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, name, stencil[i]);
#endif
          experiment->stencil[i] = g_strdup (name);
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_JSON);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.5.2.5 experiment_open_xml()

```
int experiment_open_xml (
            Experiment * experiment,
            xmlNode * node,
            unsigned int ninputs )
```

Function to open the Experiment struct on a XML node.

Returns

1 on success, 0 on error.

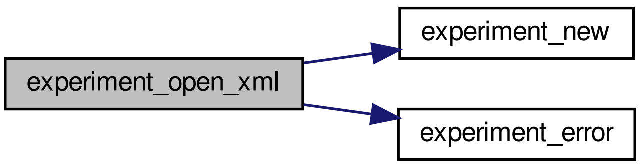
Parameters

experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 127 of file experiment.c.

```
{
  char buffer[64];
  int error_code;
  unsigned int i;

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: start\n");
#endif

  // Resetting experiment data
  experiment_new (experiment);

  // Reading the experimental data
  experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!experiment->name)
    {
      experiment_error (experiment, _("no data file name"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
#endif
  experiment->weight
    =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
                                     &error_code);
  if (error_code)
    {
      experiment_error (experiment, _("bad weight"));
      goto exit_on_error;
    }
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
#endif
  experiment->stencil[0]
    = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
  if (experiment->stencil[0])
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
               experiment->name, stencil[0]);
#endif
      ++experiment->ninputs;
    }
  else
    {
      experiment_error (experiment, _("no template"));
      goto exit_on_error;
    }
  for (i = 1; i < MAX_NINPUTS; ++i)
    {
#if DEBUG_EXPERIMENT
      fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
#endif
      if (xmlHasProp (node, (const xmlChar *) stencil[i]))
        {
          if (ninputs && ninputs <= i)
            {
              experiment_error (experiment, _("bad templates number"));
              goto exit_on_error;
            }
          experiment->stencil[i]
            = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
#if DEBUG_EXPERIMENT
          fprintf (stderr,
                   "experiment_open_xml: experiment=%s stencil%u=%s\n",
                   experiment->nexperiments, experiment->name,
                   experiment->stencil[i]);
#endif
          ++experiment->ninputs;
        }
      else if (ninputs && ninputs > i)
        {
          snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
          experiment_error (experiment, buffer);
          goto exit_on_error;
        }
      else
        break;
    }

#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
  return 1;

exit_on_error:
  experiment_free (experiment, INPUT_TYPE_XML);
#if DEBUG_EXPERIMENT
  fprintf (stderr, "experiment_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.6 experiment.h

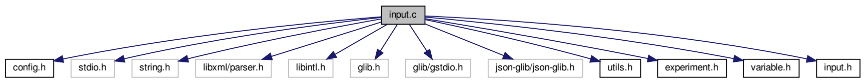
```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef EXPERIMENT__H
#define EXPERIMENT__H 1

typedef struct
{
  char *name;
  char *stencil[MAX_NINPUTS];
  double weight;
  unsigned int ninputs;
} Experiment;

extern const char *stencil[MAX_NINPUTS];

// Public functions
void experiment_new (Experiment * experiment);
void experiment_free (Experiment * experiment, unsigned int type);
void experiment_error (Experiment * experiment, char *message);
int experiment_open_xml (Experiment * experiment, xmlNode * node,
                         unsigned int ninputs);
int experiment_open_json (Experiment * experiment, JsonNode * node,
                          unsigned int ninputs);

#endif
```

4.7 input.c File Reference

Source file to define the input functions.

```
#include "config.h"
#include <stdio.h>
#include <string.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
```

Include dependency graph for input.c:



Macros

- #define DEBUG_INPUT 0

    Macro to debug input functions.

Functions

- void input_new ()
- void input_free ()
- void input_error (char ∗message)
- int input_open_xml (xmlDoc ∗doc)
- int input_open_json (JsonParser ∗parser)
- int input_open (char ∗filename)

Variables

- Input input [1]

  Global Input struct to set the input data.
- const char ∗ result_name = "result"

  Name of the result file.
- const char ∗ variables_name = "variables"

  Name of the variables file.

4.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file input.c.

4.7.2 Function Documentation

4.7.2.1 input_error()

```
void input_error (
             char ∗ message )
```

Function to print an error message opening an Input struct.

Parameters

message	Error message.

Definition at line 119 of file input.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
  error_message = g_strdup (buffer);
}
```

4.7.2.2 input_free()

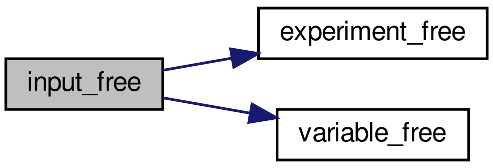
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 81 of file input.c.

```
{
  unsigned int i;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: start\n");
#endif
  g_free (input->name);
  g_free (input->directory);
  for (i = 0; i < input->nexperiments; ++i)
    experiment_free (input->experiment + i, input->
type);
  for (i = 0; i < input->nvariables; ++i)
    variable_free (input->variable + i, input->
type);
  g_free (input->experiment);
  g_free (input->variable);
  if (input->type == INPUT_TYPE_XML)
    {
      xmlFree (input->evaluator);
      xmlFree (input->simulator);
      xmlFree (input->result);
      xmlFree (input->variables);
    }
  else
    {
      g_free (input->evaluator);
      g_free (input->simulator);
      g_free (input->result);
      g_free (input->variables);
    }
  input->nexperiments = input->nvariables =
input->nsteps = 0;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: end\n");
#endif
}
```

Here is the call graph for this function:



4.7.2.3 input_new()

```
void input_new ( )
```

Function to create a new Input struct.

Definition at line 63 of file input.c.

```
{
#if DEBUG_INPUT
  fprintf (stderr, "input_new: start\n");
#endif
  input->nvariables = input->nexperiments =
input->nsteps = 0;
  input->simulator = input->evaluator = input->
directory = input->name = NULL;
  input->experiment = NULL;
  input->variable = NULL;
#if DEBUG_INPUT
  fprintf (stderr, "input_new: end\n");
#endif
}
```

4.7.2.4 input_open()

```
int input_open (
            char * filename )
```

Function to open the input file.

Returns

1_on_success, 0_on_error.

Parameters

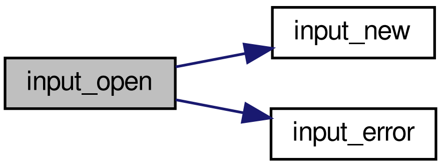
filename	Input data file name.

Definition at line 957 of file input.c.

```
{
  xmlDoc *doc;
  JsonParser *parser;

#if DEBUG_INPUT
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  input_new ();

  // Opening input file
#if DEBUG_INPUT
  fprintf (stderr, "input_open: opening the input file %s\n", filename);
  fprintf (stderr, "input_open: trying XML format\n");
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
#if DEBUG_INPUT
```

```
      fprintf (stderr, "input_open: trying JSON format\n");
#endif
      parser = json_parser_new ();
      if (!json_parser_load_from_file (parser, filename, NULL))
        {
          input_error (_("Unable to parse the input file"));
          goto exit_on_error;
        }
      if (!input_open_json (parser))
        goto exit_on_error;
    }
  else if (!input_open_xml (doc))
    goto exit_on_error;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  show_error (error_message);
  g_free (error_message);
  input_free ();
#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.7.2.5 input_open_json()

```
int input_open_json (
            JsonParser * parser )
```

Function to open the input file in JSON format.

Returns

1_on_success, 0_on_error.

Parameters

parser	JsonParser struct.

Definition at line 568 of file input.c.

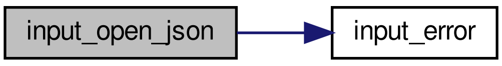
```
{
  JsonNode *node, *child;
  JsonObject *object;
  JsonArray *array;
  const char *buffer;
  int error_code;
  unsigned int i, n;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: start\n");
#endif

  // Resetting input data
  input->type = INPUT_TYPE_JSON;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: getting the root node\n");
#endif
  node = json_parser_get_root (parser);
  object = json_node_get_object (node);

  // Getting result and variables file names
  if (!input->result)
    {
      buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
      if (!buffer)
        buffer = result_name;
      input->result = g_strdup (buffer);
    }
  else
    input->result = g_strdup (result_name);
  if (!input->variables)
    {
      buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
      if (!buffer)
        buffer = variables_name;
      input->variables = g_strdup (buffer);
    }
  else
    input->variables = g_strdup (variables_name);

  // Opening simulator program name
  buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
  if (!buffer)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }
  input->simulator = g_strdup (buffer);

  // Opening evaluator program name
  buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
  if (buffer)
    input->evaluator = g_strdup (buffer);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = json_object_get_uint_with_default (object,
LABEL_SEED,
                                          DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
  if (!strcmp (buffer, LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!strcmp (buffer, LABEL_SWEEP))
```

```
    input->algorithm = ALGORITHM_SWEEP;
  else if (!strcmp (buffer, LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!strcmp (buffer, LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (json_object_get_member (object, LABEL_NPOPULATION))
        {
          input->nsimulations
            = json_object_get_uint (object,
LABEL_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (json_object_get_member (object, LABEL_NGENERATIONS))
        {
          input->niterations
            = json_object_get_uint (object,
LABEL_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (json_object_get_member (object, LABEL_MUTATION))
        {
          input->mutation_ratio
            = json_object_get_float (object, LABEL_MUTATION, &error_code
);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (json_object_get_member (object, LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = json_object_get_float (object,
LABEL_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (json_object_get_member (object, LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = json_object_get_float (object,
LABEL_ADAPTATION, &error_code);
```

```
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = json_object_get_uint (object, LABEL_NITERATIONS, &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = json_object_get_uint_with_default (object, LABEL_NBEST, 1,
                                             &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = json_object_get_float_with_default (object, LABEL_TOLERANCE, 0.,
                                              &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (json_object_get_member (object, LABEL_NSTEPS))
        {
          input->nsteps
            = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
          buffer = json_object_get_string_member (object, LABEL_CLIMBING);
          if (!strcmp (buffer, LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!strcmp (buffer, LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          input->relaxation
            = json_object_get_float_with_default (object,
LABEL_RELAXATION,
                                                  DEFAULT_RELAXATION,
                                                  &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold
    = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
                                          &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
  n = json_array_get_length (array);
  input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
  for (i = 0; i < n; ++i)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
      child = json_array_get_element (array, i);
      if (!input->nexperiments)
        {
          if (!experiment_open_json (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_json (input->experiment +
input->nexperiments,
                                     child, input->experiment->
ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }

  // Reading the variables data
  array = json_object_get_array_member (object, LABEL_VARIABLES);
  n = json_array_get_length (array);
  input->variable = (Variable *) g_malloc (n * sizeof (
Variable));
  for (i = 0; i < n; ++i)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nvariables=%u\n", input->
nvariables);
```

```
#endif
      child = json_array_get_element (array, i);
      if (!variable_open_json (input->variable + input->nvariables, child,
                               input->algorithm, input->nsteps))
        goto exit_on_error;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }

  // Obtaining the error norm
  if (json_object_get_member (object, LABEL_NORM))
    {
      buffer = json_object_get_string_member (object, LABEL_NORM);
      if (!strcmp (buffer, LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!strcmp (buffer, LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!strcmp (buffer, LABEL_P))
        {
          input->norm = ERROR_NORM_P;
          input->p = json_object_get_float (object, LABEL_P, &error_code);
          if (!error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!strcmp (buffer, LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the JSON document
  g_object_unref (parser);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 1;

exit_on_error:
  g_object_unref (parser);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.7.2.6 input_open_xml()

```
int input_open_xml (
              xmlDoc * doc )
```

Function to open the input file in XML format.

Returns

1_on_success, 0_on_error.

Parameters

doc	xmlDoc struct.

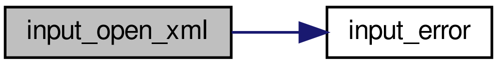
Definition at line 132 of file input.c.

```
{
  char buffer2[64];
  xmlNode *node, *child;
  xmlChar *buffer;
  int error_code;
  unsigned int i;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input->type = INPUT_TYPE_XML;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
    {
      input_error (_("Bad root XML node"));
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
      if (!input->result)
        input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
#endif
  if (!input->variables)
    {
      input->variables =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
      if (!input->variables)
        input->variables =
          (char *) xmlStrdup ((const xmlChar *) variables_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
#endif

  // Opening simulator program name
  input->simulator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
  if (!input->simulator)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                      DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
  if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                            &error_code);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
                                 &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
                                 &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
                                  &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
                                  &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
                                  &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->
nsimulations;
      i += input->adaptation_ratio * input->
nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
                             &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
                                          1, &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node,
                                           (const xmlChar *) LABEL_TOLERANCE,
                                           0., &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
        {
          input->nsteps =
            xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
                               &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
#if DEBUG_INPUT
          fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
#endif
          buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
          if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
                                     &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node,
                                               (const xmlChar *)
                                               LABEL_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
                                     0., &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
```

```
    {
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
        break;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
      input->experiment = (Experiment *)
        g_realloc (input->experiment,
                   (1 + input->nexperiments) * sizeof (
Experiment));
      if (!input->nexperiments)
        {
          if (!experiment_open_xml (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_xml (input->experiment +
input->nexperiments,
                                    child, input->experiment->
ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    input->nsimulations = 1;
  for (; child; child = child->next)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
#endif
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    _("Variable"), input->nvariables + 1, _("bad XML node"));
          input_error (buffer2);
          goto exit_on_error;
        }
      input->variable = (Variable *)
        g_realloc (input->variable,
                   (1 + input->nvariables) * sizeof (Variable));
      if (!variable_open_xml (input->variable +
input->nvariables, child,
                              input->algorithm, input->nsteps))
        goto exit_on_error;
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        input->nsimulations *= input->variable[
input->nvariables].nsweeps;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }
  if (input->nbest > input->nsimulations)
    {
      input_error (_("Best number higher than simulations number"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Obtaining the error norm
  if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
    {
      buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
      if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
```

```
        {
          input->norm = ERROR_NORM_P;
          input->p
            = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
          if (error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
      xmlFree (buffer);
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.8 input.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <string.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"

#define DEBUG_INPUT 0

Input input[1];

const char *result_name = "result";
const char *variables_name = "variables";

void
input_new ()
{
#if DEBUG_INPUT
  fprintf (stderr, "input_new: start\n");
#endif
  input->nvariables = input->nexperiments = input->nsteps = 0;
  input->simulator = input->evaluator = input->directory = input->
name = NULL;
  input->experiment = NULL;
  input->variable = NULL;
#if DEBUG_INPUT
  fprintf (stderr, "input_new: end\n");
#endif
}

void
input_free ()
{
  unsigned int i;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: start\n");
#endif
  g_free (input->name);
  g_free (input->directory);
  for (i = 0; i < input->nexperiments; ++i)
    experiment_free (input->experiment + i, input->type);
  for (i = 0; i < input->nvariables; ++i)
    variable_free (input->variable + i, input->type);
  g_free (input->experiment);
  g_free (input->variable);
  if (input->type == INPUT_TYPE_XML)
    {
      xmlFree (input->evaluator);
      xmlFree (input->simulator);
      xmlFree (input->result);
      xmlFree (input->variables);
    }
  else
    {
      g_free (input->evaluator);
      g_free (input->simulator);
      g_free (input->result);
      g_free (input->variables);
    }
  input->nexperiments = input->nvariables = input->nsteps = 0;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: end\n");
#endif
}

void
input_error (char *message)
{
  char buffer[64];
  snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
  error_message = g_strdup (buffer);
}

int
input_open_xml (xmlDoc * doc)
{
  char buffer2[64];
  xmlNode *node, *child;
  xmlChar *buffer;
  int error_code;
  unsigned int i;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input->type = INPUT_TYPE_XML;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
    {
      input_error (_("Bad root XML node"));
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
      if (!input->result)
        input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
#endif
  if (!input->variables)
    {
      input->variables =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
      if (!input->variables)
        input->variables =
          (char *) xmlStrdup ((const xmlChar *) variables_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
#endif

  // Opening simulator program name
  input->simulator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
  if (!input->simulator)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                      DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
  if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                            &error_code);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
                                 &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
                                 &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
                                  &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
                                  &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
                                  &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
                             &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
                                          1, &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node,
                                           (const xmlChar *) LABEL_TOLERANCE,
                                           0., &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
        {
          input->nsteps =
            xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
                               &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
#if DEBUG_INPUT
          fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
#endif
          buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
          if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
                                     &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node,
                                               (const xmlChar *)
                                               LABEL_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
                                     0., &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
    {
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
        break;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
      input->experiment = (Experiment *)
        g_realloc (input->experiment,
                   (1 + input->nexperiments) * sizeof (Experiment));
      if (!input->nexperiments)
        {
          if (!experiment_open_xml (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_xml (input->experiment + input->
nexperiments,
                                    child, input->experiment->ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    input->nsimulations = 1;
  for (; child; child = child->next)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
#endif
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    _("Variable"), input->nvariables + 1, _("bad XML node"));
          input_error (buffer2);
          goto exit_on_error;
        }
      input->variable = (Variable *)
        g_realloc (input->variable,
                   (1 + input->nvariables) * sizeof (Variable));
      if (!variable_open_xml (input->variable + input->
nvariables, child,
                              input->algorithm, input->nsteps))
        goto exit_on_error;
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        input->nsimulations *= input->variable[input->
nvariables].nsweeps;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }
  if (input->nbest > input->nsimulations)
    {
      input_error (_("Best number higher than simulations number"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Obtaining the error norm
  if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
    {
      buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
      if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
        {
          input->norm = ERROR_NORM_P;
          input->p
            = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
          if (error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
      xmlFree (buffer);
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 1;

```

```
exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 0;
}

int
input_open_json (JsonParser * parser)
{
  JsonNode *node, *child;
  JsonObject *object;
  JsonArray *array;
  const char *buffer;
  int error_code;
  unsigned int i, n;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: start\n");
#endif

  // Resetting input data
  input->type = INPUT_TYPE_JSON;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: getting the root node\n");
#endif
  node = json_parser_get_root (parser);
  object = json_node_get_object (node);

  // Getting result and variables file names
  if (!input->result)
    {
      buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
      if (!buffer)
        buffer = result_name;
      input->result = g_strdup (buffer);
    }
  else
    input->result = g_strdup (result_name);
  if (!input->variables)
    {
      buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
      if (!buffer)
        buffer = variables_name;
      input->variables = g_strdup (buffer);
    }
  else
    input->variables = g_strdup (variables_name);

  // Opening simulator program name
  buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
  if (!buffer)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }
  input->simulator = g_strdup (buffer);

  // Opening evaluator program name
  buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
  if (buffer)
    input->evaluator = g_strdup (buffer);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = json_object_get_uint_with_default (object,
LABEL_SEED,
                                         DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
  if (!strcmp (buffer, LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!strcmp (buffer, LABEL_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!strcmp (buffer, LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!strcmp (buffer, LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (json_object_get_member (object, LABEL_NPOPULATION))
        {
          input->nsimulations
            = json_object_get_uint (object,
LABEL_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (json_object_get_member (object, LABEL_NGENERATIONS))
        {
          input->niterations
            = json_object_get_uint (object,
LABEL_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (json_object_get_member (object, LABEL_MUTATION))
        {
          input->mutation_ratio
            = json_object_get_float (object, LABEL_MUTATION, &error_code
);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (json_object_get_member (object, LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = json_object_get_float (object,
LABEL_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (json_object_get_member (object, LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = json_object_get_float (object,
LABEL_ADAPTATION, &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->nsimulations;
      i += input->adaptation_ratio * input->nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
                                             &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
                                              &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (json_object_get_member (object, LABEL_NSTEPS))
        {
          input->nsteps
            = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
          buffer = json_object_get_string_member (object, LABEL_CLIMBING);
          if (!strcmp (buffer, LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!strcmp (buffer, LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          input->relaxation
            = json_object_get_float_with_default (object,
LABEL_RELAXATION,
                                                  DEFAULT_RELAXATION,
                                                  &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold
    = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
                                          &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
  n = json_array_get_length (array);
  input->experiment = (Experiment *) g_malloc (n * sizeof (
Experiment));
  for (i = 0; i < n; ++i)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
      child = json_array_get_element (array, i);
      if (!input->nexperiments)
        {
          if (!experiment_open_json (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_json (input->experiment + input->
nexperiments,
                                     child, input->experiment->ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }

  // Reading the variables data
  array = json_object_get_array_member (object, LABEL_VARIABLES);
  n = json_array_get_length (array);
  input->variable = (Variable *) g_malloc (n * sizeof (Variable));
  for (i = 0; i < n; ++i)
```

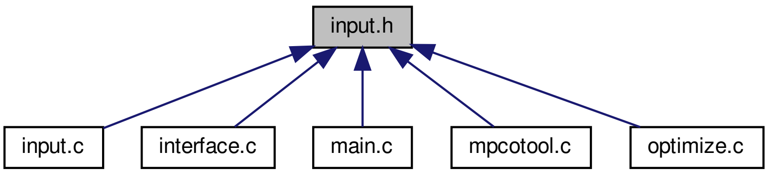
```
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
#endif
      child = json_array_get_element (array, i);
      if (!variable_open_json (input->variable + input->nvariables, child,
                               input->algorithm, input->nsteps))
        goto exit_on_error;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }

  // Obtaining the error norm
  if (json_object_get_member (object, LABEL_NORM))
    {
      buffer = json_object_get_string_member (object, LABEL_NORM);
      if (!strcmp (buffer, LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!strcmp (buffer, LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!strcmp (buffer, LABEL_P))
        {
          input->norm = ERROR_NORM_P;
          input->p = json_object_get_float (object, LABEL_P, &error_code);
          if (!error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!strcmp (buffer, LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the JSON document
  g_object_unref (parser);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 1;

exit_on_error:
  g_object_unref (parser);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 0;
}

int
input_open (char *filename)
{
  xmlDoc *doc;
  JsonParser *parser;

#if DEBUG_INPUT
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  input_new ();

  // Opening input file
#if DEBUG_INPUT
  fprintf (stderr, "input_open: opening the input file %s\n", filename);
  fprintf (stderr, "input_open: trying XML format\n");
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open: trying JSON format\n");
#endif
```

```
      parser = json_parser_new ();
      if (!json_parser_load_from_file (parser, filename, NULL))
        {
          input_error (_("Unable to parse the input file"));
          goto exit_on_error;
        }
      if (!input_open_json (parser))
        goto exit_on_error;
    }
  else if (!input_open_xml (doc))
    goto exit_on_error;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  show_error (error_message);
  g_free (error_message);
  input_free ();
#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}
```

4.9 input.h File Reference

Header file to define the input functions.

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



Data Structures

- struct Input

  Struct to define the optimization input file.

Enumerations

- enum ClimbingMethod { CLIMBING_METHOD_COORDINATES = 0, CLIMBING_METHOD_RANDOM = 1 }

  Enum to define the methods to estimate the hill climbing.
- enum ErrorNorm { ERROR_NORM_EUCLIDIAN = 0, ERROR_NORM_MAXIMUM = 1, ERROR_NORM_P = 2, ERROR_NORM_TAXICAB = 3 }

  Enum to define the error norm.

Functions

- void input_new ()
- void input_free ()
- void input_error (char ∗message)
- int input_open_xml (xmlDoc ∗doc)
- int input_open_json (JsonParser ∗parser)
- int input_open (char ∗filename)

Variables

- Input input [1]

    Global Input struct to set the input data.
- const char ∗ result_name

    Name of the result file.
- const char ∗ variables_name

    Name of the variables file.

4.9.1 Detailed Description

Header file to define the input functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file input.h.

4.9.2 Enumeration Type Documentation

4.9.2.1 ClimbingMethod

```
enum ClimbingMethod
```

Enum to define the methods to estimate the hill climbing.

Enumerator

CLIMBING_METHOD_COORDINATES	Coordinates hill climbing method.
CLIMBING_METHOD_RANDOM	Random hill climbing method.

Definition at line 42 of file input.h.

```
{
  CLIMBING_METHOD_COORDINATES = 0,
  CLIMBING_METHOD_RANDOM = 1,
};
```

4.9.2.2 ErrorNorm

```
enum ErrorNorm
```

Enum to define the error norm.

Enumerator

ERROR_NORM_EUCLIDIAN	Euclidian norm: $\sqrt{\sum_i \left(w_i\, x_i\right)^2}$.
ERROR_NORM_MAXIMUM	Maximum norm: $\max_i \left\|w_i\, x_i\right\|$.
ERROR_NORM_P	P-norm $\sqrt[p]{\sum_i \left\|w_i\, x_i\right\|^p}$.
ERROR_NORM_TAXICAB	Taxicab norm $\sum_i \left\|w_i\, x_i\right\|$.

Definition at line 49 of file input.h.

```
{
  ERROR_NORM_EUCLIDIAN = 0,
  ERROR_NORM_MAXIMUM = 1,
  ERROR_NORM_P = 2,
  ERROR_NORM_TAXICAB = 3
};
```

4.9.3 Function Documentation

4.9.3.1 input_error()

```
void input_error (
            char * message )
```

Function to print an error message opening an Input struct.

Parameters

message	Error message.

Definition at line 119 of file input.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
  error_message = g_strdup (buffer);
}
```

4.9.3.2 input_free()

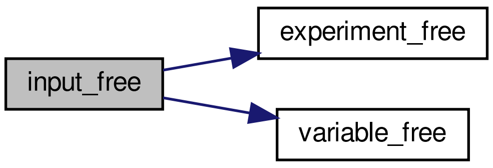
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 81 of file input.c.

```
{
  unsigned int i;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: start\n");
#endif
  g_free (input->name);
  g_free (input->directory);
  for (i = 0; i < input->nexperiments; ++i)
    experiment_free (input->experiment + i, input->
type);
  for (i = 0; i < input->nvariables; ++i)
    variable_free (input->variable + i, input->
type);
  g_free (input->experiment);
  g_free (input->variable);
  if (input->type == INPUT_TYPE_XML)
    {
      xmlFree (input->evaluator);
      xmlFree (input->simulator);
      xmlFree (input->result);
      xmlFree (input->variables);
    }
  else
    {
      g_free (input->evaluator);
      g_free (input->simulator);
      g_free (input->result);
      g_free (input->variables);
    }
  input->nexperiments = input->nvariables =
input->nsteps = 0;
#if DEBUG_INPUT
  fprintf (stderr, "input_free: end\n");
#endif
}
```

Here is the call graph for this function:



4.9.3.3 input_new()

```
void input_new ( )
```

Function to create a new Input struct.

Definition at line 63 of file input.c.

```
{
#if DEBUG_INPUT
  fprintf (stderr, "input_new: start\n");
#endif
  input->nvariables = input->nexperiments =
input->nsteps = 0;
  input->simulator = input->evaluator = input->
directory = input->name = NULL;
  input->experiment = NULL;
  input->variable = NULL;
#if DEBUG_INPUT
  fprintf (stderr, "input_new: end\n");
#endif
}
```

4.9.3.4 input_open()

```
int input_open (
            char * filename )
```

Function to open the input file.

Returns

1_on_success, 0_on_error.

Parameters

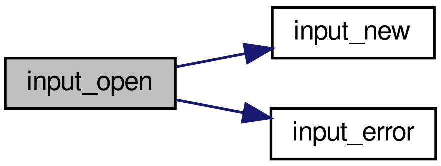
filename	Input data file name.

Definition at line 957 of file input.c.

```
{
  xmlDoc *doc;
  JsonParser *parser;

#if DEBUG_INPUT
  fprintf (stderr, "input_open: start\n");
#endif

  // Resetting input data
  input_new ();

  // Opening input file
#if DEBUG_INPUT
  fprintf (stderr, "input_open: opening the input file %s\n", filename);
  fprintf (stderr, "input_open: trying XML format\n");
#endif
  doc = xmlParseFile (filename);
  if (!doc)
    {
#if DEBUG_INPUT
```

```
      fprintf (stderr, "input_open: trying JSON format\n");
#endif
      parser = json_parser_new ();
      if (!json_parser_load_from_file (parser, filename, NULL))
        {
          input_error (_("Unable to parse the input file"));
          goto exit_on_error;
        }
      if (!input_open_json (parser))
        goto exit_on_error;
    }
  else if (!input_open_xml (doc))
    goto exit_on_error;

  // Getting the working directory
  input->directory = g_path_get_dirname (filename);
  input->name = g_path_get_basename (filename);

#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 1;

exit_on_error:
  show_error (error_message);
  g_free (error_message);
  input_free ();
#if DEBUG_INPUT
  fprintf (stderr, "input_open: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.9.3.5 input_open_json()

```
int input_open_json (
            JsonParser * parser )
```

Function to open the input file in JSON format.

Returns

1_on_success, 0_on_error.

Parameters

parser	JsonParser struct.

Definition at line 568 of file input.c.

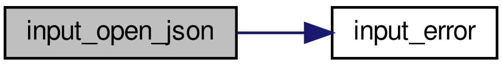
```
{
  JsonNode *node, *child;
  JsonObject *object;
  JsonArray *array;
  const char *buffer;
  int error_code;
  unsigned int i, n;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: start\n");
#endif

  // Resetting input data
  input->type = INPUT_TYPE_JSON;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: getting the root node\n");
#endif
  node = json_parser_get_root (parser);
  object = json_node_get_object (node);

  // Getting result and variables file names
  if (!input->result)
    {
      buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
      if (!buffer)
        buffer = result_name;
      input->result = g_strdup (buffer);
    }
  else
    input->result = g_strdup (result_name);
  if (!input->variables)
    {
      buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
      if (!buffer)
        buffer = variables_name;
      input->variables = g_strdup (buffer);
    }
  else
    input->variables = g_strdup (variables_name);

  // Opening simulator program name
  buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
  if (!buffer)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }
  input->simulator = g_strdup (buffer);

  // Opening evaluator program name
  buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
  if (buffer)
    input->evaluator = g_strdup (buffer);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = json_object_get_uint_with_default (object,
LABEL_SEED,
                                         DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
  if (!strcmp (buffer, LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!strcmp (buffer, LABEL_SWEEP))
```

```
    input->algorithm = ALGORITHM_SWEEP;
  else if (!strcmp (buffer, LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!strcmp (buffer, LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (json_object_get_member (object, LABEL_NPOPULATION))
        {
          input->nsimulations
            = json_object_get_uint (object,
LABEL_NPOPULATION, &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (json_object_get_member (object, LABEL_NGENERATIONS))
        {
          input->niterations
            = json_object_get_uint (object,
LABEL_NGENERATIONS, &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (json_object_get_member (object, LABEL_MUTATION))
        {
          input->mutation_ratio
            = json_object_get_float (object, LABEL_MUTATION, &error_code
);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (json_object_get_member (object, LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = json_object_get_float (object,
LABEL_REPRODUCTION, &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (json_object_get_member (object, LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = json_object_get_float (object,
LABEL_ADAPTATION, &error_code);
```

```
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->
nsimulations;
      i += input->adaptation_ratio * input->
nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
                                             &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
                                              &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (json_object_get_member (object, LABEL_NSTEPS))
        {
          input->nsteps
            = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
          buffer = json_object_get_string_member (object, LABEL_CLIMBING);
          if (!strcmp (buffer, LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!strcmp (buffer, LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = json_object_get_uint (object, LABEL_NESTIMATES, &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          input->relaxation
            = json_object_get_float_with_default (object, LABEL_RELAXATION,
                                                  DEFAULT_RELAXATION,
                                                  &error_code);
          if (error_code || input->relaxation < 0. || input->relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold
    = json_object_get_float_with_default (object, LABEL_THRESHOLD, 0.,
                                          &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
  n = json_array_get_length (array);
  input->experiment = (Experiment *) g_malloc (n * sizeof (Experiment));
  for (i = 0; i < n; ++i)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
      child = json_array_get_element (array, i);
      if (!input->nexperiments)
        {
          if (!experiment_open_json (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_json (input->experiment + input->nexperiments,
                                     child, input->experiment->ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }

  // Reading the variables data
  array = json_object_get_array_member (object, LABEL_VARIABLES);
  n = json_array_get_length (array);
  input->variable = (Variable *) g_malloc (n * sizeof (Variable));
  for (i = 0; i < n; ++i)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
```

```
#endif
      child = json_array_get_element (array, i);
      if (!variable_open_json (input->variable + input->nvariables, child,
                               input->algorithm, input->nsteps))
        goto exit_on_error;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }

  // Obtaining the error norm
  if (json_object_get_member (object, LABEL_NORM))
    {
      buffer = json_object_get_string_member (object, LABEL_NORM);
      if (!strcmp (buffer, LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!strcmp (buffer, LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!strcmp (buffer, LABEL_P))
        {
          input->norm = ERROR_NORM_P;
          input->p = json_object_get_float (object, LABEL_P, &error_code);
          if (!error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!strcmp (buffer, LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the JSON document
  g_object_unref (parser);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 1;

exit_on_error:
  g_object_unref (parser);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.9.3.6 input_open_xml()

```
int input_open_xml (
            xmlDoc * doc )
```

Function to open the input file in XML format.

Returns

1_on_success, 0_on_error.

Parameters

doc	xmlDoc struct.

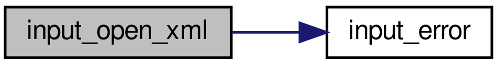
Definition at line 132 of file input.c.

```
{
  char buffer2[64];
  xmlNode *node, *child;
  xmlChar *buffer;
  int error_code;
  unsigned int i;

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: start\n");
#endif

  // Resetting input data
  buffer = NULL;
  input->type = INPUT_TYPE_XML;

  // Getting the root node
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: getting the root node\n");
#endif
  node = xmlDocGetRootElement (doc);
  if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
    {
      input_error (_("Bad root XML node"));
      goto exit_on_error;
    }

  // Getting result and variables file names
  if (!input->result)
    {
      input->result =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
      if (!input->result)
        input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
#endif
  if (!input->variables)
    {
      input->variables =
        (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
      if (!input->variables)
        input->variables =
          (char *) xmlStrdup ((const xmlChar *) variables_name);
    }
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
#endif

  // Opening simulator program name
  input->simulator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
  if (!input->simulator)
    {
      input_error (_("Bad simulator program"));
      goto exit_on_error;
    }

  // Opening evaluator program name
  input->evaluator =
    (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);

  // Obtaining pseudo-random numbers generator seed
  input->seed
    = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                      DEFAULT_RANDOM_SEED, &error_code);
  if (error_code)
    {
      input_error (_("Bad pseudo-random numbers generator seed"));
      goto exit_on_error;
    }

  // Opening algorithm
  buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
  if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
    {
      input->algorithm = ALGORITHM_MONTE_CARLO;

      // Obtaining simulations number
      input->nsimulations
        = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                            &error_code);
      if (error_code)
        {
          input_error (_("Bad simulations number"));
          goto exit_on_error;
        }
    }
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
    input->algorithm = ALGORITHM_SWEEP;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
    input->algorithm = ALGORITHM_ORTHOGONAL;
  else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
    {
      input->algorithm = ALGORITHM_GENETIC;

      // Obtaining population
      if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
        {
          input->nsimulations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
                                 &error_code);
          if (error_code || input->nsimulations < 3)
            {
              input_error (_("Invalid population number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No population number"));
          goto exit_on_error;
        }

      // Obtaining generations
      if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
        {
          input->niterations
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
                                 &error_code);
          if (error_code || !input->niterations)
            {
              input_error (_("Invalid generations number"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No generations number"));
          goto exit_on_error;
        }

      // Obtaining mutation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
        {
          input->mutation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
                                  &error_code);
          if (error_code || input->mutation_ratio < 0.
              || input->mutation_ratio >= 1.)
            {
              input_error (_("Invalid mutation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No mutation probability"));
          goto exit_on_error;
        }

      // Obtaining reproduction probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
        {
          input->reproduction_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
                                  &error_code);
          if (error_code || input->reproduction_ratio < 0.
              || input->reproduction_ratio >= 1.0)
            {
              input_error (_("Invalid reproduction probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No reproduction probability"));
          goto exit_on_error;
        }

      // Obtaining adaptation probability
      if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
        {
          input->adaptation_ratio
            = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
                                  &error_code);
          if (error_code || input->adaptation_ratio < 0.
              || input->adaptation_ratio >= 1.)
            {
              input_error (_("Invalid adaptation probability"));
              goto exit_on_error;
            }
        }
      else
        {
          input_error (_("No adaptation probability"));
          goto exit_on_error;
        }

      // Checking survivals
      i = input->mutation_ratio * input->nsimulations;
      i += input->reproduction_ratio * input->
nsimulations;
      i += input->adaptation_ratio * input->
nsimulations;
      if (i > input->nsimulations - 2)
        {
          input_error
            (_("No enough survival entities to reproduce the population"));
          goto exit_on_error;
        }
    }
  else
    {
      input_error (_("Unknown algorithm"));
      goto exit_on_error;
    }
  xmlFree (buffer);
  buffer = NULL;

  if (input->algorithm == ALGORITHM_MONTE_CARLO
      || input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {

      // Obtaining iterations number
      input->niterations
        = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
                             &error_code);
      if (error_code == 1)
        input->niterations = 1;
      else if (error_code)
        {
          input_error (_("Bad iterations number"));
          goto exit_on_error;
        }

      // Obtaining best number
      input->nbest
        = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
                                          1, &error_code);
      if (error_code || !input->nbest)
        {
          input_error (_("Invalid best number"));
          goto exit_on_error;
        }

      // Obtaining tolerance
      input->tolerance
        = xml_node_get_float_with_default (node,
                                           (const xmlChar *) LABEL_TOLERANCE,
                                           0., &error_code);
      if (error_code || input->tolerance < 0.)
        {
          input_error (_("Invalid tolerance"));
          goto exit_on_error;
        }

      // Getting hill climbing method parameters
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
        {
          input->nsteps =
            xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
                               &error_code);
          if (error_code)
            {
              input_error (_("Invalid steps number"));
              goto exit_on_error;
            }
#if DEBUG_INPUT
          fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
#endif
          buffer = xmlGetProp (node, (const xmlChar *) LABEL_CLIMBING);
          if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
            input->climbing = CLIMBING_METHOD_COORDINATES;
          else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
            {
              input->climbing = CLIMBING_METHOD_RANDOM;
              input->nestimates
                = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
                                     &error_code);
              if (error_code || !input->nestimates)
                {
                  input_error (_("Invalid estimates number"));
                  goto exit_on_error;
                }
            }
          else
            {
              input_error (_("Unknown method to estimate the hill climbing"));
              goto exit_on_error;
            }
          xmlFree (buffer);
          buffer = NULL;
          input->relaxation
            = xml_node_get_float_with_default (node,
                                               (const xmlChar *)
                                               LABEL_RELAXATION,
                                               DEFAULT_RELAXATION, &error_code);
          if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
            {
              input_error (_("Invalid relaxation parameter"));
              goto exit_on_error;
            }
        }
      else
        input->nsteps = 0;
    }
  // Obtaining the threshold
  input->threshold =
    xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
                                     0., &error_code);
  if (error_code)
    {
      input_error (_("Invalid threshold"));
      goto exit_on_error;
    }

  // Reading the experimental data
  for (child = node->children; child; child = child->next)
```

```c
    {
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
        break;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
      input->experiment = (Experiment *)
        g_realloc (input->experiment,
                   (1 + input->nexperiments) * sizeof (Experiment));
      if (!input->nexperiments)
        {
          if (!experiment_open_xml (input->experiment, child, 0))
            goto exit_on_error;
        }
      else
        {
          if (!experiment_open_xml (input->experiment + input->nexperiments,
                                    child, input->experiment->ninputs))
            goto exit_on_error;
        }
      ++input->nexperiments;
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nexperiments=%u\n",
               input->nexperiments);
#endif
    }
  if (!input->nexperiments)
    {
      input_error (_("No optimization experiments"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Reading the variables data
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    input->nsimulations = 1;
  for (; child; child = child->next)
    {
#if DEBUG_INPUT
      fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
#endif
      if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
        {
          snprintf (buffer2, 64, "%s %u: %s",
                    _("Variable"), input->nvariables + 1, _("bad XML node"));
          input_error (buffer2);
          goto exit_on_error;
        }
      input->variable = (Variable *)
        g_realloc (input->variable,
                   (1 + input->nvariables) * sizeof (Variable));
      if (!variable_open_xml (input->variable + input->nvariables, child,
                              input->algorithm, input->nsteps))
        goto exit_on_error;
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        input->nsimulations *= input->variable[input->nvariables].nsweeps;
      ++input->nvariables;
    }
  if (!input->nvariables)
    {
      input_error (_("No optimization variables"));
      goto exit_on_error;
    }
  if (input->nbest > input->nsimulations)
    {
      input_error (_("Best number higher than simulations number"));
      goto exit_on_error;
    }
  buffer = NULL;

  // Obtaining the error norm
  if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
    {
      buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
      if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
        input->norm = ERROR_NORM_EUCLIDIAN;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
        input->norm = ERROR_NORM_MAXIMUM;
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
```

```
        {
          input->norm = ERROR_NORM_P;
          input->p
            = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
          if (error_code)
            {
              input_error (_("Bad P parameter"));
              goto exit_on_error;
            }
        }
      else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
        input->norm = ERROR_NORM_TAXICAB;
      else
        {
          input_error (_("Unknown error norm"));
          goto exit_on_error;
        }
      xmlFree (buffer);
    }
  else
    input->norm = ERROR_NORM_EUCLIDIAN;

  // Closing the XML document
  xmlFreeDoc (doc);

#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 1;

exit_on_error:
  xmlFree (buffer);
  xmlFreeDoc (doc);
#if DEBUG_INPUT
  fprintf (stderr, "input_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.10 input.h

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef INPUT__H
#define INPUT__H 1

enum ClimbingMethod
{
  CLIMBING_METHOD_COORDINATES = 0,
  CLIMBING_METHOD_RANDOM = 1,
};

enum ErrorNorm
{
  ERROR_NORM_EUCLIDIAN = 0,
  ERROR_NORM_MAXIMUM = 1,
  ERROR_NORM_P = 2,
  ERROR_NORM_TAXICAB = 3
};

typedef struct
{
  Experiment *experiment;
  Variable *variable;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  char *directory;
  char *name;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  double p;
  double threshold;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int nsimulations;
  unsigned int algorithm;
  unsigned int nsteps;
  unsigned int climbing;
  unsigned int nestimates;
  unsigned int niterations;
  unsigned int nbest;
  unsigned int norm;
  unsigned int type;
} Input;

extern Input input[1];
extern const char *result_name;
extern const char *variables_name;

// Public functions
void input_new ();
void input_free ();
void input_error (char *message);
int input_open_xml (xmlDoc * doc);
int input_open_json (JsonParser * parser);
int input_open (char *filename);

#endif
```

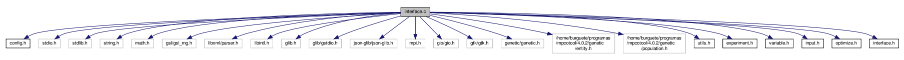
4.11 interface.c File Reference

Source file to define the graphical interface functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
```

Include dependency graph for interface.c:



Macros

- #define DEBUG_INTERFACE 0

  Macro to debug interface functions.
- #define INPUT_FILE "test-ga.xml"

  Macro to define the initial input file.

Functions

- void input_save_climbing_xml (xmlNode ∗node)
- void input_save_climbing_json (JsonNode ∗node)
- void input_save_xml (xmlDoc ∗doc)
- void input_save_json (JsonGenerator ∗generator)
- void input_save (char ∗filename)
- void options_new ()
- void running_new ()
- unsigned int window_get_algorithm ()
- unsigned int window_get_climbing ()
- unsigned int window_get_norm ()
- void window_save_climbing ()
- int window_save ()
- void window_run ()
- void window_help ()
- void window_about ()
- void window_update_climbing ()
- void window_update ()
- void window_set_algorithm ()
- void window_set_experiment ()
- void window_remove_experiment ()

- void window_add_experiment ()
- void window_name_experiment ()
- void window_weight_experiment ()
- void window_inputs_experiment ()
- void window_template_experiment (void ∗data)
- void window_set_variable ()
- void window_remove_variable ()
- void window_add_variable ()
- void window_label_variable ()
- void window_precision_variable ()
- void window_rangemin_variable ()
- void window_rangemax_variable ()
- void window_rangeminabs_variable ()
- void window_rangemaxabs_variable ()
- void window_step_variable ()
- void window_update_variable ()
- int window_read (char ∗filename)
- void window_open ()
- void window_new (GtkApplication ∗application)

Variables

- 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.

4.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file interface.c.

4.11.2 Function Documentation

4.11.2.1 input_save()

```
void input_save (
            char * filename )
```

Function to save the input file.

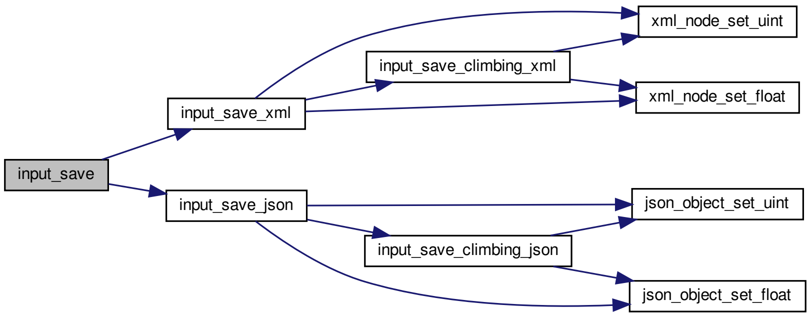
Parameters

filename	Input file name.

Definition at line 584 of file interface.c.

```
{
  xmlDoc *doc;
  JsonGenerator *generator;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);

  if (input->type == INPUT_TYPE_XML)
    {
      // Opening the input file
      doc = xmlNewDoc ((const xmlChar *) "1.0");
      input_save_xml (doc);

      // Saving the XML file
      xmlSaveFormatFile (filename, doc, 1);

      // Freeing memory
      xmlFreeDoc (doc);
    }
  else
    {
      // Opening the input file
      generator = json_generator_new ();
      json_generator_set_pretty (generator, TRUE);
      input_save_json (generator);

      // Saving the JSON file
      json_generator_to_file (generator, filename, NULL);

      // Freeing memory
      g_object_unref (generator);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.2 input_save_climbing_json()

```
void input_save_climbing_json (
            JsonNode * node )
```

Function to save the hill climbing method data in a JSON node.

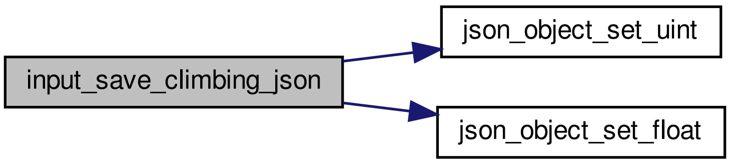
Parameters

node	JSON node.

Definition at line 201 of file interface.c.

```
{
  JsonObject *object;
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_json: start\n");
#endif
  object = json_node_get_object (node);
  if (input->nsteps)
    {
      json_object_set_uint (object, LABEL_NSTEPS,
input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        json_object_set_float (object, LABEL_RELAXATION,
input->relaxation);
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          json_object_set_string_member (object, LABEL_CLIMBING,
                                         LABEL_COORDINATES);
          break;
        default:
          json_object_set_string_member (object, LABEL_CLIMBING,
LABEL_RANDOM);
          json_object_set_uint (object, LABEL_NESTIMATES,
input->nestimates);
        }
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_json: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.3 input_save_climbing_xml()

```
void input_save_climbing_xml (
            xmlNode * node )
```

Function to save the hill climbing method data in a XML node.

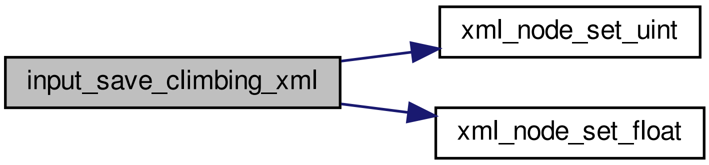
Parameters

node	XML node.

Definition at line 168 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_xml: start\n");
#endif
  if (input->nsteps)
    {
      xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        xml_node_set_float (node, (const xmlChar *)
LABEL_RELAXATION,
                            input->relaxation);
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
                      (const xmlChar *) LABEL_COORDINATES);
          break;
        default:
          xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
                      (const xmlChar *) LABEL_RANDOM);
          xml_node_set_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
                             input->nestimates);
        }
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_xml: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.4 input_save_json()

```
void input_save_json (
            JsonGenerator * generator )
```

Function to save the input file in JSON format.

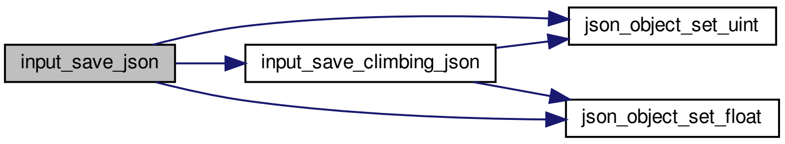
Parameters

generator	JsonGenerator struct.

Definition at line 413 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer;
  JsonNode *node, *child;
  JsonObject *object;
  JsonArray *array;
  GFile *file, *file2;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_json: start\n");
#endif

  // Setting root JSON node
  node = json_node_new (JSON_NODE_OBJECT);
  object = json_node_get_object (node);
  json_generator_set_root (generator, node);

  // Adding properties to the root JSON node
  if (strcmp (input->result, result_name))
    json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
  if (strcmp (input->variables, variables_name))
    json_object_set_string_member (object, LABEL_VARIABLES_FILE,
                                   input->variables);
  file = g_file_new_for_path (input->directory);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (strlen (buffer))
        json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    json_object_set_uint (object, LABEL_SEED,
input->seed);

  // Setting the algorithm
  buffer = (char *) g_slice_alloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      json_object_set_string_member (object, LABEL_ALGORITHM,
                                     LABEL_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    case ALGORITHM_SWEEP:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    case ALGORITHM_ORTHOGONAL:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_ORTHOGONAL);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    default:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      json_object_set_string_member (object, LABEL_MUTATION, buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
      break;
    }
  g_slice_free1 (64, buffer);
  if (input->threshold != 0.)
    json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);

  // Setting the experimental data
  array = json_array_new ();
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = json_node_new (JSON_NODE_OBJECT);
      object = json_node_get_object (child);
      json_object_set_string_member (object, LABEL_NAME,
                                     input->experiment[i].name);
      if (input->experiment[i].weight != 1.)
        json_object_set_float (object, LABEL_WEIGHT,
                               input->experiment[i].weight);
      for (j = 0; j < input->experiment->ninputs; ++j)
        json_object_set_string_member (object, stencil[j],
                                       input->experiment[i].
stencil[j]);
      json_array_add_element (array, child);
    }
  json_object_set_array_member (object, LABEL_EXPERIMENTS, array);

  // Setting the variables data
  array = json_array_new ();
  for (i = 0; i < input->nvariables; ++i)
    {
      child = json_node_new (JSON_NODE_OBJECT);
      object = json_node_get_object (child);
      json_object_set_string_member (object, LABEL_NAME,
                                     input->variable[i].name);
      json_object_set_float (object, LABEL_MINIMUM,
                             input->variable[i].rangemin);
      if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
        json_object_set_float (object,
LABEL_ABSOLUTE_MINIMUM,
                               input->variable[i].rangeminabs);
      json_object_set_float (object, LABEL_MAXIMUM,
                             input->variable[i].rangemax);
      if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
        json_object_set_float (object,
LABEL_ABSOLUTE_MAXIMUM,
                               input->variable[i].rangemaxabs);
      if (input->variable[i].precision !=
DEFAULT_PRECISION)
        json_object_set_uint (object, LABEL_PRECISION,
                              input->variable[i].precision);
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        json_object_set_uint (object, LABEL_NSWEEPS,
                              input->variable[i].nsweeps);
      else if (input->algorithm == ALGORITHM_GENETIC)
        json_object_set_uint (object, LABEL_NBITS,
input->variable[i].nbits);
      if (input->nsteps)
        json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
      json_array_add_element (array, child);
    }
  json_object_set_array_member (object, LABEL_VARIABLES, array);

  // Saving the error norm
  switch (input->norm)
    {
    case ERROR_NORM_MAXIMUM:
      json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
      break;
    case ERROR_NORM_P:
      json_object_set_string_member (object, LABEL_NORM, LABEL_P);
      json_object_set_float (object, LABEL_P, input->
p);
      break;
    case ERROR_NORM_TAXICAB:
      json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_json: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.5 input_save_xml()

```
void input_save_xml (
              xmlDoc * doc )
```

Function to save the input file in XML format.

Parameters

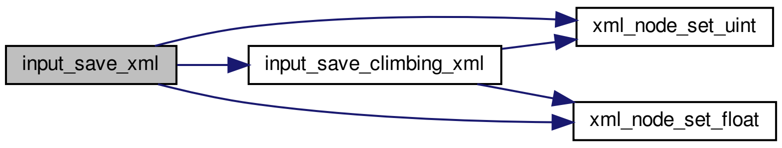
doc	xmlDoc struct.

Definition at line 233 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_xml: start\n");
#endif

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp
      ((const xmlChar *) input->result, (const xmlChar *) result_name))
    xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
                (xmlChar *) input->result);
  if (xmlStrcmp
      ((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
    xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
                (xmlChar *) input->variables);
  file = g_file_new_for_path (input->directory);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
                    (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);

  // Setting the algorithm
  buffer = (char *) g_slice_alloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    case ALGORITHM_ORTHOGONAL:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_ORTHOGONAL);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    default:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_slice_free1 (64, buffer);
  if (input->threshold != 0.)
```

```
    xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
                        input->threshold);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
      xmlSetProp (child, (const xmlChar *) LABEL_NAME,
                  (xmlChar *) input->experiment[i].name);
      if (input->experiment[i].weight != 1.)
        xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
                            input->experiment[i].weight);
      for (j = 0; j < input->experiment->ninputs; ++j)
        xmlSetProp (child, (const xmlChar *) stencil[j],
                    (xmlChar *) input->experiment[i].stencil[j]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
      xmlSetProp (child, (const xmlChar *) LABEL_NAME,
                  (xmlChar *) input->variable[i].name);
      xml_node_set_float (child, (const xmlChar *) LABEL_MINIMUM,
                          input->variable[i].rangemin);
      if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
        xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM,
                            input->variable[i].rangeminabs);
      xml_node_set_float (child, (const xmlChar *) LABEL_MAXIMUM,
                          input->variable[i].rangemax);
      if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
        xml_node_set_float (child, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM,
                            input->variable[i].rangemaxabs);
      if (input->variable[i].precision != DEFAULT_PRECISION)
        xml_node_set_uint (child, (const xmlChar *) LABEL_PRECISION,
                           input->variable[i].precision);
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        xml_node_set_uint (child, (const xmlChar *) LABEL_NSWEEPS,
                           input->variable[i].nsweeps);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
                           input->variable[i].nbits);
      if (input->nsteps)
        xml_node_set_float (child, (const xmlChar *) LABEL_STEP,
                            input->variable[i].step);
    }

  // Saving the error norm
  switch (input->norm)
    {
    case ERROR_NORM_MAXIMUM:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_MAXIMUM);
      break;
    case ERROR_NORM_P:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_P);
      xml_node_set_float (node, (const xmlChar *) LABEL_P, input->p);
      break;
    case ERROR_NORM_TAXICAB:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_TAXICAB);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.6 options_new()

```
void options_new ( )
```

Function to open the options dialog.

Definition at line 632 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: start\n");
#endif
  options->label_seed = (GtkLabel *)
    gtk_label_new (_("Pseudo-random numbers generator seed"));
  options->spin_seed = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_seed),
     _("Seed to init the pseudo-random numbers generator"));
  gtk_spin_button_set_value (options->spin_seed, (gdouble) input->seed);
  options->label_threads = (GtkLabel *)
    gtk_label_new (_("Threads number for the stochastic algorithm"));
  options->spin_threads
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_threads),
     _("Number of threads to perform the calibration/optimization for "
       "the stochastic algorithm"));
  gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
  options->label_climbing = (GtkLabel *)
    gtk_label_new (_("Threads number for the hill climbing method"));
  options->spin_climbing =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_climbing),
     _("Number of threads to perform the calibration/optimization for the "
       "hill climbing method"));
  gtk_spin_button_set_value (options->spin_climbing,
                             (gdouble) nthreads_climbing);
  options->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
                   0, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
                   1, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
                   1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
                           1);
  gtk_widget_show_all (GTK_WIDGET (options->grid));
  options->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Options"),
                                 window->window,
                                 GTK_DIALOG_MODAL,
                                 _("_OK"), GTK_RESPONSE_OK,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
     GTK_WIDGET (options->grid));
  if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
    {
      input->seed
        = (unsigned long int) gtk_spin_button_get_value (options->
spin_seed);
      nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
      nthreads_climbing
        = gtk_spin_button_get_value_as_int (options->spin_climbing);
    }
  gtk_widget_destroy (GTK_WIDGET (options->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: end\n");
#endif
}
```

4.11.2.7 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: start\n");
#endif
  running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
  running->spinner = (GtkSpinner *) gtk_spinner_new ();
  running->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
  gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
  running->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Calculating"),
                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
  gtk_container_add (GTK_CONTAINER
                     (gtk_dialog_get_content_area (running->dialog)),
                     GTK_WIDGET (running->grid));
  gtk_spinner_start (running->spinner);
  gtk_widget_show_all (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: end\n");
#endif
}
```

4.11.2.8 window_about()

```
void window_about ( )
```

Function to show an about dialog.

Definition at line 1057 of file interface.c.

```
{
  static const gchar *authors[] = {
    "Javier Burguete Tolosa <jburguete@eead.csic.es>",
    "Borja Latorre Garcés <borja.latorre@csic.es>",
    NULL
  };
#if DEBUG_INTERFACE
  fprintf (stderr, "window_about: start\n");
#endif
  gtk_show_about_dialog
    (window->window,
     "program_name", "MPCOTool",
     "comments",
     _("The Multi-Purposes Calibration and Optimization Tool.\n"
       "A software to perform calibrations or optimizations of empirical "
       "parameters"),
     "authors", authors,
     "translator-credits",
     "Javier Burguete Tolosa <jburguete@eead.csic.es> "
     "(english, french and spanish)\n"
     "Uğur Çayoğlu (german)",
     "version", "4.0.1",
     "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
     "logo", window->logo,
     "website", "https://github.com/jburguete/mpcotool",
     "license-type", GTK_LICENSE_BSD, NULL);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_about: end\n");
#endif
}
```

4.11.2.9 window_add_experiment()

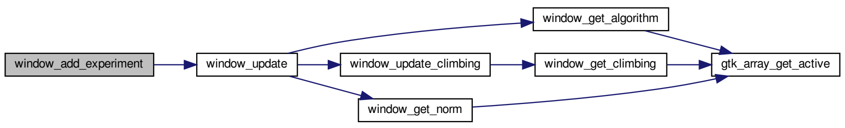
```
void window_add_experiment ( )
```

Function to add an experiment in the main window.

Definition at line 1392 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_insert_text
    (window->combo_experiment, i, input->experiment[i].name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  input->experiment = (Experiment *) g_realloc
    (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
  for (j = input->nexperiments - 1; j > i; --j)
    memcpy (input->experiment + j + 1, input->experiment + j,
            sizeof (Experiment));
  input->experiment[j + 1].weight = input->experiment[j].weight;
  input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
  if (input->type == INPUT_TYPE_XML)
    {
      input->experiment[j + 1].name
        = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
    }
  else
    {
      input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = g_strdup (input->experiment[i].stencil[j]);
    }
  ++input->nexperiments;
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_block (window->button_template[j],
window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->
id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.10 window_add_variable()

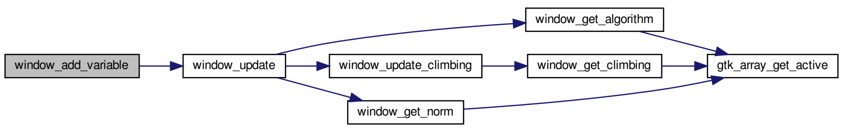
```
void window_add_variable ( )
```

Function to add a variable in the main window.

Definition at line 1655 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  gtk_combo_box_text_insert_text (window->combo_variable, i,
                                  input->variable[i].name);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
  input->variable = (Variable *) g_realloc
    (input->variable, (input->nvariables + 1) * sizeof (
Variable));
  for (j = input->nvariables - 1; j > i; --j)
    memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
  memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
  if (input->type == INPUT_TYPE_XML)
    input->variable[j + 1].name
      = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
  else
    input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
  ++input->nvariables;
  g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
  g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.11 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

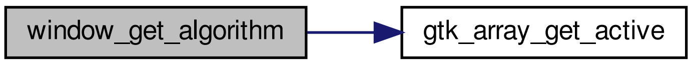
Returns

Stochastic algorithm number.

Definition at line 732 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.11.2.12 window_get_climbing()

`unsigned int window_get_climbing ( )`

Function to get the hill climbing method number.

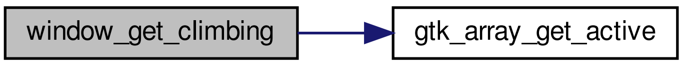
Returns

Hill climbing method number.

Definition at line 752 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: start\n");
#endif
  i = gtk_array_get_active (window->button_climbing,
    NCLIMBINGS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: %u\n", i);
  fprintf (stderr, "window_get_climbing: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.11.2.13 window_get_norm()

`unsigned int window_get_norm ( )`

Function to get the norm method number.

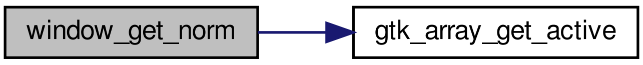
Returns

Norm method number.

Definition at line 772 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: start\n");
#endif
  i = gtk_array_get_active (window->button_norm,
NNORMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: %u\n", i);
  fprintf (stderr, "window_get_norm: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.11.2.14 window_help()

```
void window_help ( )
```

Function to show a help dialog.

Definition at line 1029 of file interface.c.

```
{
  char *buffer, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: start\n");
#endif
  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
                              _("user-manual.pdf"), NULL);
  buffer = g_filename_to_uri (buffer2, NULL, NULL);
  g_free (buffer2);
#if GTK_MINOR_VERSION >= 22
  gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
#else
  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
#endif
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: uri=%s\n", buffer);
#endif
  g_free (buffer);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: end\n");
#endif
}
```

4.11.2.15 window_inputs_experiment()

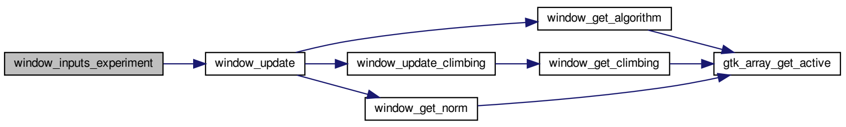
```
void window_inputs_experiment ( )
```

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

Definition at line 1492 of file interface.c.

```
{
  unsigned int j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: start\n");
#endif
  j = input->experiment->ninputs - 1;
  if (j
      && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                        (window->check_template[j])))
    --input->experiment->ninputs;
  if (input->experiment->ninputs < MAX_NINPUTS
      && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                       (window->check_template[j])))
    ++input->experiment->ninputs;
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.16 window_label_variable()

```
void window_label_variable ( )
```

Function to set the variable label in the main window.

Definition at line 1690 of file interface.c.

```
{
  unsigned int i;
  const char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  buffer = gtk_entry_get_text (window->entry_variable);
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: end\n");
#endif
}
```

4.11.2.17 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1445 of file interface.c.

```
{
  unsigned int i;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
button_experiment));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: end\n");
#endif
}
```

4.11.2.18 window_new()

```
void window_new (
            GtkApplication * application )
```

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2065 of file interface.c.

```
{
  unsigned int i;
  char *buffer, *buffer2, buffer3[64];
  char *label_algorithm[NALGORITHMS] = {
    "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
  };
  char *tip_algorithm[NALGORITHMS] = {
    _("Monte-Carlo brute force algorithm"),
    _("Sweep brute force algorithm"),
    _("Genetic algorithm"),
    _("Orthogonal sampling brute force algorithm"),
  };
  char *label_climbing[NCLIMBINGS] = {
    _("_Coordinates climbing"), _("_Random climbing")
  };
  char *tip_climbing[NCLIMBINGS] = {
    _("Coordinates climbing estimate method"),
    _("Random climbing estimate method")
  };
  char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
  char *tip_norm[NNORMS] = {
    _("Euclidean error norm (L2)"),
    _("Maximum error norm (L)"),
    _("P error norm (Lp)"),
    _("Taxicab error norm (L1)")
  };

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif

  // Creating the window
  window->window = main_window
    = (GtkWindow *) gtk_application_window_new (application);

  // Finish when closing the window
  g_signal_connect_swapped (window->window, "delete-event",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Setting the window title
  gtk_window_set_title (window->window, "MPCOTool");

  // Creating the open button
  window->button_open = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-open",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
  g_signal_connect (window->button_open, "clicked", window_open, NULL);

  // Creating the save button
  window->button_save = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-save",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
  g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
                    NULL);

  // Creating the run button
  window->button_run = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("system-run",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
  g_signal_connect (window->button_run, "clicked", window_run, NULL);

  // Creating the options button
  window->button_options = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("preferences-system",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
  g_signal_connect (window->button_options, "clicked",
options_new, NULL);

  // Creating the help button
  window->button_help = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-browser",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
  g_signal_connect (window->button_help, "clicked", window_help, NULL);

  // Creating the about button
  window->button_about = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-about",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
  g_signal_connect (window->button_about, "clicked",
window_about, NULL);

  // Creating the exit button
  window->button_exit = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("application-exit",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
  g_signal_connect_swapped (window->button_exit, "clicked",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Creating the buttons bar
  window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_open), 0);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_save), 1);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_run), 2);
```

```
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
  gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);

  // Creating the simulator program label and entry
  window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
  window->button_simulator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Simulator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
                               _("Simulator program executable file"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);

  // Creating the evaluator program label and entry
  window->check_evaluator = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
  g_signal_connect (window->check_evaluator, "toggled", window_update, NULL);
  window->button_evaluator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Evaluator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_evaluator),
     _("Optional evaluator program executable file"));

  // Creating the results files labels and entries
  window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
  window->entry_result = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_result), _("Best results file"));
  window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
  window->entry_variables = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variables), _("All simulated results file"));

  // Creating the files grid and attaching widgets
  window->grid_files = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
                   0, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
                   1, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
                   0, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
                   1, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
                   0, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
                   1, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
                   0, 3, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
                   1, 3, 1, 1);

  // Creating the algorithm properties
  window->label_simulations = (GtkLabel *) gtk_label_new
    (_("Simulations number"));
  window->spin_simulations
      = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_simulations),
     _("Number of simulations to perform for each iteration"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
  window->label_iterations = (GtkLabel *)
    gtk_label_new (_("Iterations number"));
  window->spin_iterations
      = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
  g_signal_connect
    (window->spin_iterations, "value-changed",
window_update, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
  window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
  window->spin_tolerance =
    (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_tolerance),
     _("Tolerance to set the variable interval on the next iteration"));
  window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
  window->spin_bests
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bests),
     _("Number of best simulations used to set the variable interval "
       "on the next iteration"));
  window->label_population
    = (GtkLabel *) gtk_label_new (_("Population number"));
  window->spin_population
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_population),
     _("Number of population for the genetic algorithm"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
  window->label_generations
    = (GtkLabel *) gtk_label_new (_("Generations number"));
  window->spin_generations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_generations),
     _("Number of generations for the genetic algorithm"));
  window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
  window->spin_mutation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_mutation),
     _("Ratio of mutation for the genetic algorithm"));
  window->label_reproduction
    = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
  window->spin_reproduction
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_reproduction),
     _("Ratio of reproduction for the genetic algorithm"));
  window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
  window->spin_adaptation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_adaptation),
     _("Ratio of adaptation for the genetic algorithm"));
  window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
  window->spin_threshold = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
                                    precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_threshold),
     _("Threshold in the objective function to finish the simulations"));
  window->scrolled_threshold =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
                     GTK_WIDGET (window->spin_threshold));
//  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
//  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
//                               GTK_ALIGN_FILL);

  // Creating the hill climbing method properties
  window->check_climbing = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
  g_signal_connect (window->check_climbing, "clicked",
window_update, NULL);
  window->grid_climbing = (GtkGrid *) gtk_grid_new ();
  window->button_climbing[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
  gtk_grid_attach (window->grid_climbing,
                   GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_climbing[0], "clicked",
window_update, NULL);
  for (i = 0; ++i < NCLIMBINGS;)
    {
      window->button_climbing[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_climbing[0]),
         label_climbing[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
                                   tip_climbing[i]);
      gtk_grid_attach (window->grid_climbing,
                       GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
      g_signal_connect (window->button_climbing[i], "clicked", window_update,
                        NULL);
    }
  window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
  window->spin_steps = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
  window->label_estimates
    = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
  window->spin_estimates = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e3, 1.);
  window->label_relaxation
    = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
  window->spin_relaxation = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., 2., 0.001);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
                   0, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
                   1, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
                   0, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
                   1, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
                   0, NCLIMBINGS + 2, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
                   1, NCLIMBINGS + 2, 1, 1);

  // Creating the array of algorithms
  window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
  window->button_algorithm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
                               tip_algorithm[0]);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_algorithm[0], "clicked",
                    window_set_algorithm, NULL);
  for (i = 0; ++i < NALGORITHMS;)
    {
      window->button_algorithm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_algorithm[0]),
         label_algorithm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
                                   tip_algorithm[i]);
      gtk_grid_attach (window->grid_algorithm,
                       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_algorithm[i], "clicked",
                        window_set_algorithm, NULL);
    }
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_simulations),
                   0, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_iterations),
                   0, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
                   1, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
                   0, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
                   1, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
                   0, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
                   1, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_population),
                   0, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
                        1, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->label_generations),
                        0, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->spin_generations),
                        1, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_mutation),
                        0, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_mutation),
                        1, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->label_reproduction),
                        0, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->spin_reproduction),
                        1, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->label_adaptation),
                        0, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_adaptation),
                        1, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->check_climbing),
                        0, NALGORITHMS + 9, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->grid_climbing),
                        0, NALGORITHMS + 10, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_threshold),
                        0, NALGORITHMS + 11, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                        GTK_WIDGET (window->scrolled_threshold),
                        1, NALGORITHMS + 11, 1, 1);
  window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
  gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
                          GTK_WIDGET (window->grid_algorithm));

  // Creating the variable widgets
  window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->combo_variable), _("Variables selector"));
  window->id_variable = g_signal_connect
    (window->combo_variable, "changed", window_set_variable, NULL);
  window->button_add_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_add_variable, "clicked",
window_add_variable,
                         NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
                                    _("Add variable"));
  window->button_remove_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_variable, "clicked",
                         window_remove_variable, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
                                    _("Remove variable"));
  window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
  window->entry_variable = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variable), _("Variable name"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
  window->id_variable_label = g_signal_connect
    (window->entry_variable, "changed",
window_label_variable, NULL);
  window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
  window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
  window->scrolled_min
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_min),
                          GTK_WIDGET (window->spin_min));
  g_signal_connect (window->spin_min, "value-changed",
                         window_rangemin_variable, NULL);
  window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
  window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
  window->scrolled_max
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_max),
                     GTK_WIDGET (window->spin_max));
  g_signal_connect (window->spin_max, "value-changed",
                    window_rangemax_variable, NULL);
  window->check_minabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
  g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
  window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_minabs),
     _("Minimum allowed value of the variable"));
  window->scrolled_minabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
                     GTK_WIDGET (window->spin_minabs));
  g_signal_connect (window->spin_minabs, "value-changed",
                    window_rangeminabs_variable, NULL);
  window->check_maxabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
  g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
  window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_maxabs),
     _("Maximum allowed value of the variable"));
  window->scrolled_maxabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
                     GTK_WIDGET (window->spin_maxabs));
  g_signal_connect (window->spin_maxabs, "value-changed",
                    window_rangemaxabs_variable, NULL);
  window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
  window->spin_precision = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_precision),
     _("Number of precision floating point digits\n"
       "0 is for integer numbers"));
  g_signal_connect (window->spin_precision, "value-changed",
                    window_precision_variable, NULL);
  window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
  window->spin_sweeps =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
                               _("Number of steps sweeping the variable"));
  g_signal_connect (window->spin_sweeps, "value-changed",
                    window_update_variable, NULL);
  window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
  window->spin_bits
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bits),
     _("Number of bits to encode the variable"));
  g_signal_connect
    (window->spin_bits, "value-changed", window_update_variable, NULL)
;
  window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
  window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_step),
     _("Initial step size for the hill climbing method"));
  window->scrolled_step
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_step),
                     GTK_WIDGET (window->spin_step));
  g_signal_connect
    (window->spin_step, "value-changed", window_step_variable, NULL);
  window->grid_variable = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
  window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
  gtk_container_add (GTK_CONTAINER (window->frame_variable),
                     GTK_WIDGET (window->grid_variable));

  // Creating the experiment widgets
  window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
                               _("Experiment selector"));
  window->id_experiment = g_signal_connect
    (window->combo_experiment, "changed",
window_set_experiment, NULL);
  window->button_add_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
                               _("Add experiment"));
  window->button_remove_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_experiment, "clicked",
                    window_remove_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),
                               _("Remove experiment"));
  window->label_experiment
    = (GtkLabel *) gtk_label_new (_("Experimental data file"));
  window->button_experiment = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Experimental data file"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
                               _("Experimental data file"));
  window->id_experiment_name
    = g_signal_connect (window->button_experiment, "selection-changed",
                        window_name_experiment, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
  window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
  window->spin_weight
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_weight),
     _("Weight factor to build the objective function"));
  g_signal_connect
    (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
  window->grid_experiment = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
;
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
  for (i = 0; i < MAX_NINPUTS; ++i)
    {
      snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
      window->check_template[i] = (GtkCheckButton *)
        gtk_check_button_new_with_label (buffer3);
      window->id_template[i]
        = g_signal_connect (window->check_template[i], "toggled",
                            window_inputs_experiment, NULL);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
      window->button_template[i] = (GtkFileChooserButton *)
        gtk_file_chooser_button_new (_("Input template"),
                                     GTK_FILE_CHOOSER_ACTION_OPEN);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
                                   _("Experimental input template file"));
      window->id_input[i] =
        g_signal_connect_swapped (window->button_template[i],
                                  "selection-changed",
                                  (GCallback) window_template_experiment,
                                  (void *) (size_t) i);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
    }
  window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
  gtk_container_add (GTK_CONTAINER (window->frame_experiment),
                     GTK_WIDGET (window->grid_experiment));

  // Creating the error norm widgets
  window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
  window->grid_norm = (GtkGrid *) gtk_grid_new ();
  gtk_container_add (GTK_CONTAINER (window->frame_norm),
                     GTK_WIDGET (window->grid_norm));
  window->button_norm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
                               tip_norm[0]);
  gtk_grid_attach (window->grid_norm,
                   GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_norm[0], "clicked",
window_update, NULL);
  for (i = 0; ++i < NNORMS;)
    {
      window->button_norm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
                                   tip_norm[i]);
      gtk_grid_attach (window->grid_norm,
                       GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
    }
  window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
label_p), 1, 1, 1, 1);
  window->spin_p = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
                               _("P parameter for the P error norm"));
  window->scrolled_p =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_p),
                     GTK_WIDGET (window->spin_p));
  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
scrolled_p),
                   1, 2, 1, 2);

  // Creating the grid and attaching the widgets to the grid
  window->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
window->grid));

  // Setting the window logo
  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
  gtk_window_set_icon (window->window, window->logo);

  // Showing the window
  gtk_widget_show_all (GTK_WIDGET (window->window));

  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
#if GTK_MINOR_VERSION >= 16
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
#endif

  // Reading initial example
  input_new ();
  buffer2 = g_get_current_dir ();
  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
  g_free (buffer2);
  window_read (buffer);
  g_free (buffer);

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif
}
```

4.11.2.19 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1979 of file interface.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer, *directory, *name;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: start\n");
#endif

  // Saving a backup of the current input file
  directory = g_strdup (input->directory);
  name = g_strdup (input->name);

  // Opening dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Open input file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_OPEN,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // Adding JSON filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "JSON");
  gtk_file_filter_add_pattern (filter, "*.json");
  gtk_file_filter_add_pattern (filter, "*.JSON");
  gtk_file_filter_add_pattern (filter, "*.js");
  gtk_file_filter_add_pattern (filter, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK saving
  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Traying to open the input file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      if (!window_read (buffer))
        {
#if DEBUG_INTERFACE
          fprintf (stderr, "window_open: error reading input file\n");
#endif
          g_free (buffer);

          // Reading backup file on error
          buffer = g_build_filename (directory, name, NULL);
          input->result = input->variables = NULL;
          if (!input_open (buffer))
            {

              // Closing on backup file reading error
#if DEBUG_INTERFACE
              fprintf (stderr, "window_read: error reading backup file\n");
#endif
              g_free (buffer);
              break;
            }
          g_free (buffer);
        }
      else
        {
          g_free (buffer);
          break;
        }
    }

  // Freeing and closing
  g_free (name);
  g_free (directory);
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: end\n");
#endif
}
```

4.11.2.20 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1713 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].precision
    = (unsigned int) gtk_spin_button_get_value_as_int (window->
spin_precision);
  gtk_spin_button_set_digits (window->spin_min, input->
variable[i].precision);
  gtk_spin_button_set_digits (window->spin_max, input->
variable[i].precision);
  gtk_spin_button_set_digits (window->spin_minabs,
                              input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_maxabs,
                              input->variable[i].precision);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: end\n");
#endif
}
```

4.11.2.21 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1754 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemax = gtk_spin_button_get_value (
window->spin_max);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: end\n");
#endif
}
```

4.11.2.22 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1789 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemaxabs
    = gtk_spin_button_get_value (window->spin_maxabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: end\n");
#endif
}
```

4.11.2.23 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1737 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemin = gtk_spin_button_get_value (
window->spin_min);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: end\n");
#endif
}
```

4.11.2.24 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1771 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangeminabs
    = gtk_spin_button_get_value (window->spin_minabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: end\n");
#endif
}
```

4.11.2.25 window_read()

```
int window_read (
            char * filename )
```

Function to read the input data of a file.

Returns

1 on succes, 0 on error.

Parameters

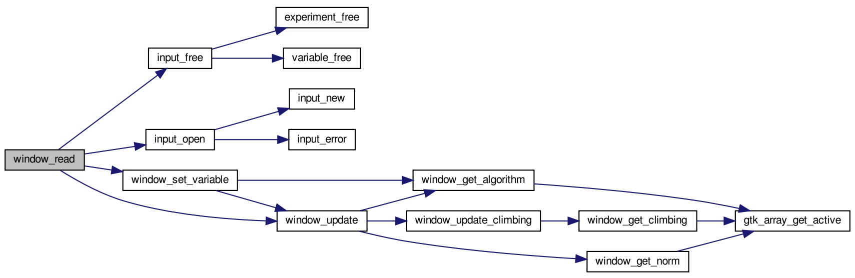
filename	File name.

Definition at line 1863 of file interface.c.

```
{
  unsigned int i;
  char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  input->result = input->variables = NULL;
  if (!input_open (filename))
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_read: end\n");
#endif
      return 0;
    }

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->
variables);
```

```
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
      // fallthrough
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_climbing), input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_climbing[input->climbing]),
             TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->climbing)
            {
            case CLIMBING_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
  gtk_spin_button_set_value (window->spin_p, input->p);
  gtk_spin_button_set_value (window->spin_threshold, input->threshold);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i].name);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01959   gtk_combo_box_text_remove_all (window->combo_variable);
01960   for (i = 0; i < input->nvariables; ++i)
01961     gtk_combo_box_text_append_text (window->combo_variable,
01962                                     input->variable[i].name);
01963   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01964   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01965   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01966   window_set_variable ();
01967   window_update ();
01968
01969 #if DEBUG_INTERFACE
01970   fprintf (stderr, "window_read: end\n");
01971 #endif
01972   return 1;
01973 }
```

Here is the call graph for this function:



4.11.2.26 window_remove_experiment()

```
void window_remove_experiment ( )
```

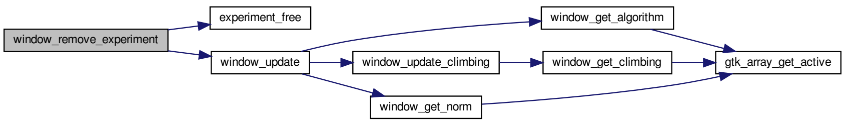
Function to remove an experiment in the main window.

Definition at line 1355 of file interface.c.

```
01356 {
01357   unsigned int i, j;
01358 #if DEBUG_INTERFACE
01359   fprintf (stderr, "window_remove_experiment: start\n");
01360 #endif
01361   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01362   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01363   gtk_combo_box_text_remove (window->combo_experiment, i);
01364   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01365   experiment_free (input->experiment + i, input->
      type);
01366   --input->nexperiments;
01367   for (j = i; j < input->nexperiments; ++j)
01368     memcpy (input->experiment + j, input->experiment + j + 1,
01369             sizeof (Experiment));
01370   j = input->nexperiments - 1;
01371   if (i > j)
01372     i = j;
01373   for (j = 0; j < input->experiment->ninputs; ++j)
01374     g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01375   g_signal_handler_block
```

```
    (window->button_experiment, window->
id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.27 window_remove_variable()

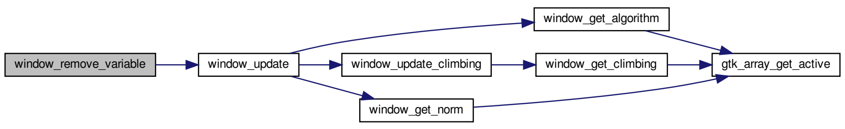
```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

Definition at line 1625 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
  xmlFree (input->variable[i].name);
  --input->nvariables;
  for (j = i; j < input->nvariables; ++j)
    memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
  j = input->nvariables - 1;
  if (i > j)
    i = j;
  g_signal_handler_block (window->entry_variable, window->
id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.28 window_run()

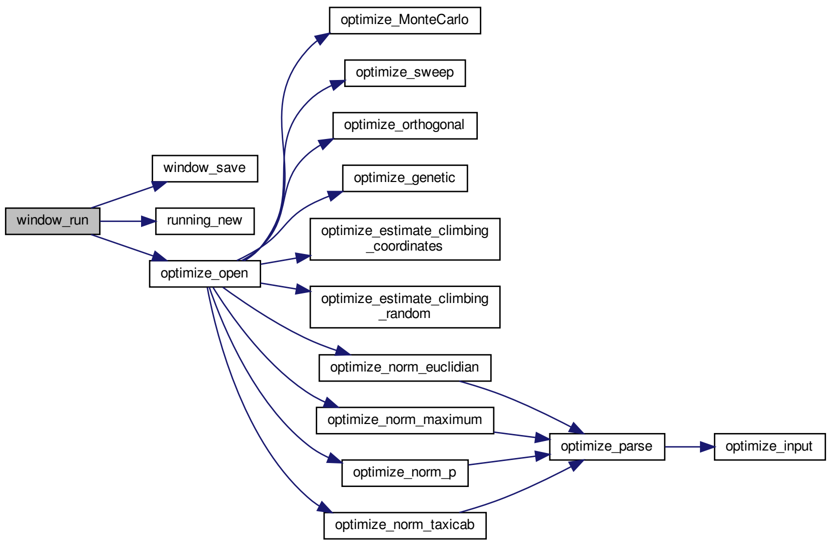
```
void window_run ( )
```

Function to run a optimization.

Definition at line 974 of file interface.c.

```
{
  unsigned int i;
  char *msg, *msg2, buffer[64], buffer2[64];
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: start\n");
#endif
  if (!window_save ())
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_run: end\n");
#endif
      return;
    }
  running_new ();
  while (gtk_events_pending ())
    gtk_main_iteration ();
  optimize_open ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: closing running dialog\n");
#endif
  gtk_spinner_stop (running->spinner);
  gtk_widget_destroy (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: displaying results\n");
#endif
  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
  msg2 = g_strdup (buffer);
  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
    {
      snprintf (buffer, 64, "%s = %s\n",
                input->variable[i].name, format[input->
variable[i].precision]);
      snprintf (buffer2, 64, buffer, optimize->value_old[i]);
      msg = g_strconcat (msg2, buffer2, NULL);
      g_free (msg2);
    }
  snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
            optimize->calculation_time);
  msg = g_strconcat (msg2, buffer, NULL);
  g_free (msg2);
  show_message (_("Best result"), msg, INFO_TYPE);
  g_free (msg);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: freeing memory\n");
#endif
  optimize_free ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.29 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 823 of file interface.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter1, *filter2;
  char *buffer;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter1 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter1, "XML");
  gtk_file_filter_add_pattern (filter1, "*.xml");
  gtk_file_filter_add_pattern (filter1, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);

  // Adding JSON filter
  filter2 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter2, "JSON");
  gtk_file_filter_add_pattern (filter2, "*.json");
  gtk_file_filter_add_pattern (filter2, "*.JSON");
  gtk_file_filter_add_pattern (filter2, "*.js");
  gtk_file_filter_add_pattern (filter2, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);

  if (input->type == INPUT_TYPE_XML)
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
  else
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {
      // Setting input file type
      filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
      buffer = (char *) gtk_file_filter_get_name (filter1);
      if (!strcmp (buffer, "XML"))
        input->type = INPUT_TYPE_XML;
      else
        input->type = INPUT_TYPE_JSON;

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      if (input->type == INPUT_TYPE_XML)
        {
          input->result
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->entry_result));
          input->variables
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->
entry_variables));
        }
      else
        {
          input->result = g_strdup (gtk_entry_get_text (window->
entry_result));
          input->variables =
            g_strdup (gtk_entry_get_text (window->entry_variables));
        }

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_ORTHOGONAL:
          input->algorithm = ALGORITHM_ORTHOGONAL;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
```

```
          input->nbest = gtk_spin_button_get_value_as_int (window->spin_bests);
          window_save_climbing ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }
      input->norm = window_get_norm ();
      input->p = gtk_spin_button_get_value (window->spin_p);
      input->threshold = gtk_spin_button_get_value (window->spin_threshold);

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}
```

4.11.2.30 window_save_climbing()

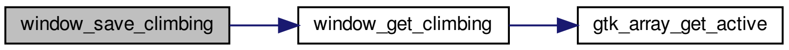
`void window_save_climbing ( )`

Function to save the hill climbing method data in the input file.

Definition at line 790 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: start\n");
#endif
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
    {
      input->nsteps = gtk_spin_button_get_value_as_int (window->spin_steps);
      input->relaxation = gtk_spin_button_get_value (window->spin_relaxation);
      switch (window_get_climbing ())
        {
        case CLIMBING_METHOD_COORDINATES:
          input->climbing = CLIMBING_METHOD_COORDINATES;
          break;
        default:
          input->climbing = CLIMBING_METHOD_RANDOM;
          input->nestimates
            = gtk_spin_button_get_value_as_int (window->spin_estimates);
        }
    }
  else
    input->nsteps = 0;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.31 window_set_algorithm()

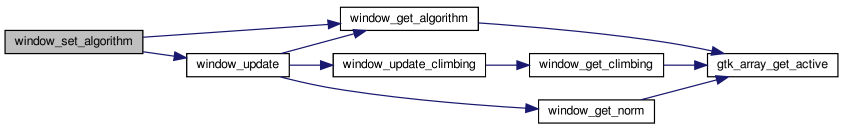
```
void window_set_algorithm ( )
```

Function to avoid memory errors changing the algorithm.

Definition at line 1281 of file interface.c.

```
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: start\n");
#endif
  i = window_get_algorithm ();
  switch (i)
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].
nsweeps);
      break;
    case ALGORITHM_GENETIC:
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_bits,
                                 (gdouble) input->variable[i].nbits);
    }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.32 window_set_experiment()

```
void window_set_experiment ( )
```

Function to set the experiment data in the main window.

Definition at line 1315 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer1, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
  buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
  buffer2 = g_build_filename (input->directory, buffer1, NULL);
  g_free (buffer1);
  g_signal_handler_block
    (window->button_experiment, window->
id_experiment_name);
  gtk_file_chooser_set_filename
    (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  g_free (buffer2);
  for (j = 0; j < input->experiment->ninputs; ++j)
    {
      g_signal_handler_block (window->button_template[j],
window->id_input[j]);
      buffer2 =
        g_build_filename (input->directory, input->experiment[i].
stencil[j],
                          NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_template[j]), buffer2);
      g_free (buffer2);
      g_signal_handler_unblock
        (window->button_template[j], window->id_input[j]);
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: end\n");
#endif
}
```

4.11.2.33 window_set_variable()

```
void window_set_variable ( )
```

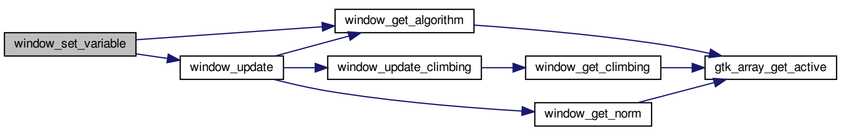
Function to set the variable data in the main window.

Definition at line 1548 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->entry_variable, window->
id_variable_label);
  gtk_entry_set_text (window->entry_variable, input->
variable[i].name);
  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
  gtk_spin_button_set_value (window->spin_min, input->
variable[i].rangemin);
```

```
  gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
  if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_minabs,
                                 input->variable[i].rangeminabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
    }
  if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_maxabs,
                                 input->variable[i].rangemaxabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
    }
  gtk_spin_button_set_value (window->spin_precision,
                             input->variable[i].precision);
  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
  if (input->nsteps)
    gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
           input->variable[i].precision);
#endif
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].nsweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      gtk_spin_button_set_value (window->spin_bits,
                                 (gdouble) input->variable[i].nbits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
               input->variable[i].nbits);
#endif
      break;
    }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.34 window_step_variable()

```
void window_step_variable ( )
```

Function to update the variable step in the main window.

Definition at line 1807 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_step_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].step = gtk_spin_button_get_value (window->
spin_step);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_step_variable: end\n");
#endif
}
```

4.11.2.35 window_template_experiment()

```
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 1517 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  if (input->type == INPUT_TYPE_XML)
    input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
  else
    input->experiment[j].stencil[i] = g_strdup (buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}
```

4.11.2.36 window_update()

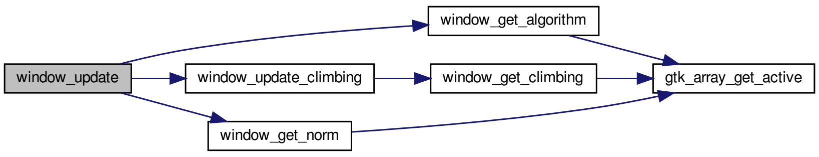
```
void window_update ( )
```

Function to update the main window view.

Definition at line 1124 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: start\n");
#endif
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_evaluator),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                   (window->check_evaluator)));
  gtk_widget_hide (GTK_WIDGET (window->label_simulations));
  gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
  gtk_widget_hide (GTK_WIDGET (window->label_iterations));
  gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
  gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->label_bests));
  gtk_widget_hide (GTK_WIDGET (window->spin_bests));
  gtk_widget_hide (GTK_WIDGET (window->label_population));
  gtk_widget_hide (GTK_WIDGET (window->spin_population));
  gtk_widget_hide (GTK_WIDGET (window->label_generations));
  gtk_widget_hide (GTK_WIDGET (window->spin_generations));
  gtk_widget_hide (GTK_WIDGET (window->label_mutation));
  gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
  gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->label_bits));
  gtk_widget_hide (GTK_WIDGET (window->spin_bits));
  gtk_widget_hide (GTK_WIDGET (window->check_climbing));
  gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
  gtk_widget_hide (GTK_WIDGET (window->label_step));
  gtk_widget_hide (GTK_WIDGET (window->spin_step));
  gtk_widget_hide (GTK_WIDGET (window->label_p));
  gtk_widget_hide (GTK_WIDGET (window->spin_p));
  i = gtk_spin_button_get_value_as_int (window->spin_iterations);
  switch (window_get_algorithm ())
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_widget_show (GTK_WIDGET (window->label_simulations));
      gtk_widget_show (GTK_WIDGET (window->spin_simulations));
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      window_update_climbing ();
      break;
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      gtk_widget_show (GTK_WIDGET (window->label_sweeps));
      gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
      gtk_widget_show (GTK_WIDGET (window->check_climbing));
      window_update_climbing ();
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_population));
      gtk_widget_show (GTK_WIDGET (window->spin_population));
      gtk_widget_show (GTK_WIDGET (window->label_generations));
      gtk_widget_show (GTK_WIDGET (window->spin_generations));
      gtk_widget_show (GTK_WIDGET (window->label_mutation));
      gtk_widget_show (GTK_WIDGET (window->spin_mutation));
      gtk_widget_show (GTK_WIDGET (window->label_reproduction));
      gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
      gtk_widget_show (GTK_WIDGET (window->label_adaptation));
      gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
      gtk_widget_show (GTK_WIDGET (window->label_bits));
      gtk_widget_show (GTK_WIDGET (window->spin_bits));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_variable),
input->nvariables > 1);
  for (i = 0; i < input->experiment->ninputs; ++i)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
      g_signal_handler_block
        (window->check_template[i], window->
id_template[i]);
      g_signal_handler_block (window->button_template[i],
window->id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 1);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  if (i > 0)
    {
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
                                gtk_toggle_button_get_active
                                GTK_TOGGLE_BUTTON (window->check_template
                                                   [i - 1]));
    }
  if (i < MAX_NINPUTS)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i]));
      g_signal_handler_block
        (window->check_template[i], window->
id_template[i]);
      g_signal_handler_block (window->button_template[i],
window->id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 0);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  while (++i < MAX_NINPUTS)
    {
      gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
      gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_minabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_maxabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
  if (window_get_norm () == ERROR_NORM_P)
    {
      gtk_widget_show (GTK_WIDGET (window->label_p));
      gtk_widget_show (GTK_WIDGET (window->spin_p));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.37 window_update_climbing()

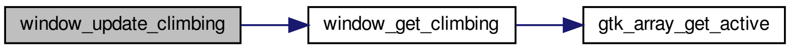
```
void window_update_climbing ( )
```

Function to update hill climbing method widgets view in the main window.

Definition at line 1093 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: start\n");
#endif
  gtk_widget_show (GTK_WIDGET (window->check_climbing));
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
check_climbing)))
    {
      gtk_widget_show (GTK_WIDGET (window->grid_climbing));
      gtk_widget_show (GTK_WIDGET (window->label_step));
      gtk_widget_show (GTK_WIDGET (window->spin_step));
    }
  switch (window_get_climbing ())
    {
    case CLIMBING_METHOD_COORDINATES:
      gtk_widget_hide (GTK_WIDGET (window->label_estimates));
      gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_estimates));
      gtk_widget_show (GTK_WIDGET (window->spin_estimates));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.38 window_update_variable()

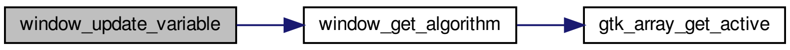
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1824 of file interface.c.

```
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  if (i < 0)
    i = 0;
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      input->variable[i].nsweeps
        = gtk_spin_button_get_value_as_int (window->spin_sweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      input->variable[i].nbits
        = gtk_spin_button_get_value_as_int (window->spin_bits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
               input->variable[i].nbits);
#endif
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.11.2.39 window_weight_experiment()

```
void window_weight_experiment ( )
```

Function to update the experiment weight in the main window.

Definition at line 1475 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  input->experiment[i].weight = gtk_spin_button_get_value (
window->spin_weight);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: end\n");
#endif
}
```

4.12 interface.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"

#define DEBUG_INTERFACE 0


#ifdef G_OS_WIN32
#define INPUT_FILE "test-ga-win.xml"
#else
#define INPUT_FILE "test-ga.xml"
#endif

const char *logo[] = {
  "32 32 3 1",
  "     c None",
  ".    c #0000FF",
  "+    c #FF0000",
  "                                ",
  "                                ",
  "                                ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .     +++     .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "     .      .    +++++    .     ",
  "    +++     .     +++    +++    ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "   +++++    .      .    +++++   ",
  "    +++     .      .     +++    ",
  "     .      .      .      .     ",
  "     .     +++     .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .    +++++    .      .     ",
  "     .     +++     .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "     .      .      .      .     ",
  "                                ",
  "                                ",
  "                                "
};

/*
const char * logo[] = {
"32 32 3 1",
"     c #FFFFFFFFFFFF",
".    c #00000000FFFF",
"X    c #FFFF00000000",
"                                ",
"                                ",
"                                ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .     XXX     .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"     .      .    XXXXX    .     ",
"    XXX     .     XXX    XXX    ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"   XXXXX    .      .    XXXXX   ",
"    XXX     .      .     XXX    ",
"     .      .      .      .     ",
"     .     XXX     .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .    XXXXX    .      .     ",
"     .     XXX     .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"     .      .      .      .     ",
"                                ",
"                                ",
"                                "};
*/

Options options[1];
Running running[1];
Window window[1];

void
input_save_climbing_xml (xmlNode * node)
{
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_xml: start\n");
#endif
  if (input->nsteps)
    {
      xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        xml_node_set_float (node, (const xmlChar *)
LABEL_RELAXATION,
                            input->relaxation);
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
                      (const xmlChar *) LABEL_COORDINATES);
          break;
        default:
          xmlSetProp (node, (const xmlChar *) LABEL_CLIMBING,
                      (const xmlChar *) LABEL_RANDOM);
          xml_node_set_uint (node, (const xmlChar *) LABEL_NESTIMATES,
                             input->nestimates);
        }
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_xml: end\n");
#endif
}

void
input_save_climbing_json (JsonNode * node)
{
  JsonObject *object;
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_json: start\n");
#endif
  object = json_node_get_object (node);
  if (input->nsteps)
    {
      json_object_set_uint (object, LABEL_NSTEPS, input->nsteps);
      if (input->relaxation != DEFAULT_RELAXATION)
        json_object_set_float (object, LABEL_RELAXATION, input->relaxation);
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          json_object_set_string_member (object, LABEL_CLIMBING,
                                         LABEL_COORDINATES);
          break;
        default:
          json_object_set_string_member (object, LABEL_CLIMBING, LABEL_RANDOM);
          json_object_set_uint (object, LABEL_NESTIMATES, input->nestimates);
        }
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_climbing_json: end\n");
#endif
}

void
input_save_xml (xmlDoc * doc)
{
  unsigned int i, j;
  char *buffer;
  xmlNode *node, *child;
  GFile *file, *file2;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_xml: start\n");
#endif

  // Setting root XML node
  node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
  xmlDocSetRootElement (doc, node);

  // Adding properties to the root XML node
  if (xmlStrcmp
      ((const xmlChar *) input->result, (const xmlChar *) result_name))
    xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
                (xmlChar *) input->result);
  if (xmlStrcmp
      ((const xmlChar *) input->variables, (const xmlChar *) variables_name))
    xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
                (xmlChar *) input->variables);
  file = g_file_new_for_path (input->directory);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (xmlStrlen ((xmlChar *) buffer))
        xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
                    (xmlChar *) buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    xml_node_set_uint (node, (const xmlChar *) LABEL_SEED, input->seed);

  // Setting the algorithm
  buffer = (char *) g_slice_alloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    case ALGORITHM_SWEEP:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    case ALGORITHM_ORTHOGONAL:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_ORTHOGONAL);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
      input_save_climbing_xml (node);
      break;
    default:
      xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
                  (const xmlChar *) LABEL_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
                  (xmlChar *) buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
      break;
    }
  g_slice_free1 (64, buffer);
  if (input->threshold != 0.)
    xml_node_set_float (node, (const xmlChar *) LABEL_THRESHOLD,
                        input->threshold);

  // Setting the experimental data
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
      xmlSetProp (child, (const xmlChar *) LABEL_NAME,
                  (xmlChar *) input->experiment[i].name);
      if (input->experiment[i].weight != 1.)
        xml_node_set_float (child, (const xmlChar *) LABEL_WEIGHT,
                            input->experiment[i].weight);
      for (j = 0; j < input->experiment->ninputs; ++j)
        xmlSetProp (child, (const xmlChar *) stencil[j],
                    (xmlChar *) input->experiment[i].stencil[j]);
    }

  // Setting the variables data
  for (i = 0; i < input->nvariables; ++i)
    {
      child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
      xmlSetProp (child, (const xmlChar *) LABEL_NAME,
                  (xmlChar *) input->variable[i].name);
      xml_node_set_float (child, (const xmlChar *)
LABEL_MINIMUM,
                          input->variable[i].rangemin);
      if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
        xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MINIMUM,
                            input->variable[i].rangeminabs);
      xml_node_set_float (child, (const xmlChar *)
LABEL_MAXIMUM,
                          input->variable[i].rangemax);
      if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
        xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MAXIMUM,
                            input->variable[i].rangemaxabs);
      if (input->variable[i].precision !=
DEFAULT_PRECISION)
        xml_node_set_uint (child, (const xmlChar *)
LABEL_PRECISION,
                           input->variable[i].precision);
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        xml_node_set_uint (child, (const xmlChar *)
LABEL_NSWEEPS,
                           input->variable[i].nsweeps);
      else if (input->algorithm == ALGORITHM_GENETIC)
        xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
                           input->variable[i].nbits);
      if (input->nsteps)
        xml_node_set_float (child, (const xmlChar *)
LABEL_STEP,
                            input->variable[i].step);
    }

  // Saving the error norm
  switch (input->norm)
    {
    case ERROR_NORM_MAXIMUM:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_MAXIMUM);
      break;
    case ERROR_NORM_P:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_P);
      xml_node_set_float (node, (const xmlChar *) LABEL_P,
input->p);
      break;
    case ERROR_NORM_TAXICAB:
      xmlSetProp (node, (const xmlChar *) LABEL_NORM,
                  (const xmlChar *) LABEL_TAXICAB);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: end\n");
#endif
}

void
input_save_json (JsonGenerator * generator)
{
  unsigned int i, j;
  char *buffer;
  JsonNode *node, *child;
  JsonObject *object;
  JsonArray *array;
  GFile *file, *file2;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_json: start\n");
#endif

  // Setting root JSON node
  node = json_node_new (JSON_NODE_OBJECT);
  object = json_node_get_object (node);
  json_generator_set_root (generator, node);

  // Adding properties to the root JSON node
  if (strcmp (input->result, result_name))
    json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
  if (strcmp (input->variables, variables_name))
    json_object_set_string_member (object, LABEL_VARIABLES_FILE,
                                    input->variables);
  file = g_file_new_for_path (input->directory);
  file2 = g_file_new_for_path (input->simulator);
  buffer = g_file_get_relative_path (file, file2);
  g_object_unref (file2);
  json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
  g_free (buffer);
  if (input->evaluator)
    {
      file2 = g_file_new_for_path (input->evaluator);
      buffer = g_file_get_relative_path (file, file2);
      g_object_unref (file2);
      if (strlen (buffer))
        json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
      g_free (buffer);
    }
  if (input->seed != DEFAULT_RANDOM_SEED)
    json_object_set_uint (object, LABEL_SEED,
input->seed);

  // Setting the algorithm
  buffer = (char *) g_slice_alloc (64);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      json_object_set_string_member (object, LABEL_ALGORITHM,
                                     LABEL_MONTE_CARLO);
      snprintf (buffer, 64, "%u", input->nsimulations);
      json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    case ALGORITHM_SWEEP:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    case ALGORITHM_ORTHOGONAL:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_ORTHOGONAL);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->tolerance);
      json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
      snprintf (buffer, 64, "%u", input->nbest);
      json_object_set_string_member (object, LABEL_NBEST, buffer);
      input_save_climbing_json (node);
      break;
    default:
      json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
      snprintf (buffer, 64, "%u", input->nsimulations);
      json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
      snprintf (buffer, 64, "%u", input->niterations);
      json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
      snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
      json_object_set_string_member (object, LABEL_MUTATION, buffer);
      snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
      json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
      snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
      json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
      break;
    }
  g_slice_free1 (64, buffer);
  if (input->threshold != 0.)
    json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);

  // Setting the experimental data
  array = json_array_new ();
  for (i = 0; i < input->nexperiments; ++i)
    {
      child = json_node_new (JSON_NODE_OBJECT);
      object = json_node_get_object (child);
      json_object_set_string_member (object, LABEL_NAME,
                                     input->experiment[i].name);
      if (input->experiment[i].weight != 1.)
        json_object_set_float (object, LABEL_WEIGHT,
                               input->experiment[i].weight);
      for (j = 0; j < input->experiment->ninputs; ++j)
        json_object_set_string_member (object, stencil[j],
                                       input->experiment[i].
stencil[j]);
      json_array_add_element (array, child);
    }
  json_object_set_array_member (object, LABEL_EXPERIMENTS, array);

  // Setting the variables data
  array = json_array_new ();
  for (i = 0; i < input->nvariables; ++i)
    {
      child = json_node_new (JSON_NODE_OBJECT);
      object = json_node_get_object (child);
      json_object_set_string_member (object, LABEL_NAME,
                                     input->variable[i].name);
      json_object_set_float (object, LABEL_MINIMUM,
                             input->variable[i].rangemin);
      if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
        json_object_set_float (object,
LABEL_ABSOLUTE_MINIMUM,
                               input->variable[i].rangeminabs);
      json_object_set_float (object, LABEL_MAXIMUM,
                             input->variable[i].rangemax);
      if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
        json_object_set_float (object,
LABEL_ABSOLUTE_MAXIMUM,
                               input->variable[i].rangemaxabs);
      if (input->variable[i].precision !=
DEFAULT_PRECISION)
        json_object_set_uint (object, LABEL_PRECISION,
                              input->variable[i].precision);
      if (input->algorithm == ALGORITHM_SWEEP
          || input->algorithm == ALGORITHM_ORTHOGONAL)
        json_object_set_uint (object, LABEL_NSWEEPS,
                              input->variable[i].nsweeps);
      else if (input->algorithm == ALGORITHM_GENETIC)
        json_object_set_uint (object, LABEL_NBITS,
input->variable[i].nbits);
      if (input->nsteps)
        json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
      json_array_add_element (array, child);
    }
  json_object_set_array_member (object, LABEL_VARIABLES, array);

  // Saving the error norm
  switch (input->norm)
    {
    case ERROR_NORM_MAXIMUM:
      json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
      break;
    case ERROR_NORM_P:
      json_object_set_string_member (object, LABEL_NORM, LABEL_P);
      json_object_set_float (object, LABEL_P, input->
p);
      break;
    case ERROR_NORM_TAXICAB:
      json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save_json: end\n");
#endif
}

void
input_save (char *filename)
{
  xmlDoc *doc;
  JsonGenerator *generator;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);

  if (input->type == INPUT_TYPE_XML)
    {
      // Opening the input file
      doc = xmlNewDoc ((const xmlChar *) "1.0");
      input_save_xml (doc);

      // Saving the XML file
      xmlSaveFormatFile (filename, doc, 1);

      // Freeing memory
      xmlFreeDoc (doc);
    }
  else
    {
      // Opening the input file
      generator = json_generator_new ();
      json_generator_set_pretty (generator, TRUE);
      input_save_json (generator);

      // Saving the JSON file
      json_generator_to_file (generator, filename, NULL);

      // Freeing memory
      g_object_unref (generator);
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: end\n");
#endif
}

void
options_new ()
{
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: start\n");
#endif
  options->label_seed = (GtkLabel *)
    gtk_label_new (_("Pseudo-random numbers generator seed"));
  options->spin_seed = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_seed),
     _("Seed to init the pseudo-random numbers generator"));
  gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
  options->label_threads = (GtkLabel *)
    gtk_label_new (_("Threads number for the stochastic algorithm"));
  options->spin_threads
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_threads),
     _("Number of threads to perform the calibration/optimization for "
       "the stochastic algorithm"));
  gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
  options->label_climbing = (GtkLabel *)
    gtk_label_new (_("Threads number for the hill climbing method"));
  options->spin_climbing =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_climbing),
     _("Number of threads to perform the calibration/optimization for the "
       "hill climbing method"));
  gtk_spin_button_set_value (options->spin_climbing,
                             (gdouble) nthreads_climbing);
  options->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
                   0, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
                   1, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->label_climbing), 0, 2, 1,
                   1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_climbing), 1, 2, 1,
                   1);
  gtk_widget_show_all (GTK_WIDGET (options->grid));
  options->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Options"),
                                 window->window,
                                 GTK_DIALOG_MODAL,
                                 _("_OK"), GTK_RESPONSE_OK,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
     GTK_WIDGET (options->grid));
  if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
    {
      input->seed
        = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
      nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
      nthreads_climbing
        = gtk_spin_button_get_value_as_int (options->spin_climbing);
    }
  gtk_widget_destroy (GTK_WIDGET (options->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: end\n");
#endif
}

void
running_new ()
{
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: start\n");
#endif
  running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
  running->spinner = (GtkSpinner *) gtk_spinner_new ();
  running->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
  gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
  running->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Calculating"),
                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
  gtk_container_add (GTK_CONTAINER
                     (gtk_dialog_get_content_area (running->dialog)),
                     GTK_WIDGET (running->grid));
  gtk_spinner_start (running->spinner);
  gtk_widget_show_all (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: end\n");
#endif
}

unsigned int
window_get_algorithm ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  i = gtk_array_get_active (window->button_algorithm,
NALGORITHMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}

unsigned int
window_get_climbing ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: start\n");
#endif
  i = gtk_array_get_active (window->button_climbing,
NCLIMBINGS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: %u\n", i);
  fprintf (stderr, "window_get_climbing: end\n");
#endif
  return i;
}

unsigned int
window_get_norm ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: start\n");
#endif
  i = gtk_array_get_active (window->button_norm,
NNORMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: %u\n", i);
  fprintf (stderr, "window_get_norm: end\n");
#endif
  return i;
}

void
window_save_climbing ()
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: start\n");
```

```
#endif
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
    {
      input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
      input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
      switch (window_get_climbing ())
        {
        case CLIMBING_METHOD_COORDINATES:
          input->climbing = CLIMBING_METHOD_COORDINATES;
          break;
        default:
          input->climbing = CLIMBING_METHOD_RANDOM;
          input->nestimates
            = gtk_spin_button_get_value_as_int (window->spin_estimates);
        }
    }
  else
    input->nsteps = 0;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: end\n");
#endif
}

int
window_save ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter1, *filter2;
  char *buffer;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter1 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter1, "XML");
  gtk_file_filter_add_pattern (filter1, "*.xml");
  gtk_file_filter_add_pattern (filter1, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);

  // Adding JSON filter
  filter2 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter2, "JSON");
  gtk_file_filter_add_pattern (filter2, "*.json");
  gtk_file_filter_add_pattern (filter2, "*.JSON");
  gtk_file_filter_add_pattern (filter2, "*.js");
  gtk_file_filter_add_pattern (filter2, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);

  if (input->type == INPUT_TYPE_XML)
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
  else
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {
      // Setting input file type
      filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
      buffer = (char *) gtk_file_filter_get_name (filter1);
      if (!strcmp (buffer, "XML"))
        input->type = INPUT_TYPE_XML;
      else
        input->type = INPUT_TYPE_JSON;

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      if (input->type == INPUT_TYPE_XML)
        {
          input->result
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->entry_result));
          input->variables
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->entry_variables));
        }
      else
        {
          input->result = g_strdup (gtk_entry_get_text (window->
entry_result));
          input->variables =
            g_strdup (gtk_entry_get_text (window->entry_variables));
        }

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_ORTHOGONAL:
          input->algorithm = ALGORITHM_ORTHOGONAL;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }
      input->norm = window_get_norm ();
      input->p = gtk_spin_button_get_value (window->spin_p);
      input->threshold = gtk_spin_button_get_value (window->
spin_threshold);

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}

void
window_run ()
{
  unsigned int i;
  char *msg, *msg2, buffer[64], buffer2[64];
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: start\n");
#endif
  if (!window_save ())
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_run: end\n");
#endif
      return;
    }
  running_new ();
  while (gtk_events_pending ())
    gtk_main_iteration ();
  optimize_open ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: closing running dialog\n");
#endif
  gtk_spinner_stop (running->spinner);
  gtk_widget_destroy (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: displaying results\n");
#endif
  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
  msg2 = g_strdup (buffer);
  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
    {
      snprintf (buffer, 64, "%s = %s\n",
                input->variable[i].name, format[input->
variable[i].precision]);
      snprintf (buffer2, 64, buffer, optimize->value_old[i]);
      msg = g_strconcat (msg2, buffer2, NULL);
      g_free (msg2);
    }
  snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
            optimize->calculation_time);
  msg = g_strconcat (msg2, buffer, NULL);
  g_free (msg2);
  show_message (_("Best result"), msg, INFO_TYPE);
  g_free (msg);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: freeing memory\n");
#endif
  optimize_free ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: end\n");
#endif
}

void
window_help ()
{
  char *buffer, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: start\n");
#endif
  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
                              _("user-manual.pdf"), NULL);
  buffer = g_filename_to_uri (buffer2, NULL, NULL);
  g_free (buffer2);
#if GTK_MINOR_VERSION >= 22
  gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
#else
  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
#endif
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: uri=%s\n", buffer);
#endif
  g_free (buffer);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: end\n");
#endif
}

void
window_about ()
```

```
{
  static const gchar *authors[] = {
    "Javier Burguete Tolosa <jburguete@eead.csic.es>",
    "Borja Latorre Garcés <borja.latorre@csic.es>",
    NULL
  };
#if DEBUG_INTERFACE
  fprintf (stderr, "window_about: start\n");
#endif
  gtk_show_about_dialog
    (window->window,
     "program_name", "MPCOTool",
     "comments",
     _("The Multi-Purposes Calibration and Optimization Tool.\n"
       "A software to perform calibrations or optimizations of empirical "
       "parameters"),
     "authors", authors,
     "translator-credits",
     "Javier Burguete Tolosa <jburguete@eead.csic.es> "
     "(english, french and spanish)\n"
     "Uğur Çayoğlu (german)",
     "version", "4.0.1",
     "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
     "logo", window->logo,
     "website", "https://github.com/jburguete/mpcotool",
     "license-type", GTK_LICENSE_BSD, NULL);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_about: end\n");
#endif
}

void
window_update_climbing ()
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: start\n");
#endif
  gtk_widget_show (GTK_WIDGET (window->check_climbing));
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_climbing)))
    {
      gtk_widget_show (GTK_WIDGET (window->grid_climbing));
      gtk_widget_show (GTK_WIDGET (window->label_step));
      gtk_widget_show (GTK_WIDGET (window->spin_step));
    }
  switch (window_get_climbing ())
    {
    case CLIMBING_METHOD_COORDINATES:
      gtk_widget_hide (GTK_WIDGET (window->label_estimates));
      gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_estimates));
      gtk_widget_show (GTK_WIDGET (window->spin_estimates));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: end\n");
#endif
}

void
window_update ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: start\n");
#endif
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_evaluator),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                   (window->check_evaluator)));
  gtk_widget_hide (GTK_WIDGET (window->label_simulations));
  gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
  gtk_widget_hide (GTK_WIDGET (window->label_iterations));
  gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
  gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->label_bests));
  gtk_widget_hide (GTK_WIDGET (window->spin_bests));
  gtk_widget_hide (GTK_WIDGET (window->label_population));
  gtk_widget_hide (GTK_WIDGET (window->spin_population));
  gtk_widget_hide (GTK_WIDGET (window->label_generations));
  gtk_widget_hide (GTK_WIDGET (window->spin_generations));
  gtk_widget_hide (GTK_WIDGET (window->label_mutation));
  gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
  gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->label_bits));
  gtk_widget_hide (GTK_WIDGET (window->spin_bits));
  gtk_widget_hide (GTK_WIDGET (window->check_climbing));
  gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
  gtk_widget_hide (GTK_WIDGET (window->label_step));
  gtk_widget_hide (GTK_WIDGET (window->spin_step));
  gtk_widget_hide (GTK_WIDGET (window->label_p));
  gtk_widget_hide (GTK_WIDGET (window->spin_p));
  i = gtk_spin_button_get_value_as_int (window->spin_iterations);
  switch (window_get_algorithm ())
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_widget_show (GTK_WIDGET (window->label_simulations));
      gtk_widget_show (GTK_WIDGET (window->spin_simulations));
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      window_update_climbing ();
      break;
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      gtk_widget_show (GTK_WIDGET (window->label_sweeps));
      gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
      gtk_widget_show (GTK_WIDGET (window->check_climbing));
      window_update_climbing ();
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_population));
      gtk_widget_show (GTK_WIDGET (window->spin_population));
      gtk_widget_show (GTK_WIDGET (window->label_generations));
      gtk_widget_show (GTK_WIDGET (window->spin_generations));
      gtk_widget_show (GTK_WIDGET (window->label_mutation));
      gtk_widget_show (GTK_WIDGET (window->spin_mutation));
      gtk_widget_show (GTK_WIDGET (window->label_reproduction));
      gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
      gtk_widget_show (GTK_WIDGET (window->label_adaptation));
      gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
      gtk_widget_show (GTK_WIDGET (window->label_bits));
      gtk_widget_show (GTK_WIDGET (window->spin_bits));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_experiment), input->nexperiments > 1);
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_variable), input->nvariables > 1);
  for (i = 0; i < input->experiment->ninputs; ++i)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 1);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  if (i > 0)
    {
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
                                gtk_toggle_button_get_active
                                GTK_TOGGLE_BUTTON (window->check_template
                                                        [i - 1]));
    }
  if (i < MAX_NINPUTS)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i]));
      g_signal_handler_block
        (window->check_template[i], window->id_template[i]);
      g_signal_handler_block (window->button_template[i], window->
id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 0);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  while (++i < MAX_NINPUTS)
    {
      gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
      gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_minabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_maxabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
  if (window_get_norm () == ERROR_NORM_P)
    {
      gtk_widget_show (GTK_WIDGET (window->label_p));
      gtk_widget_show (GTK_WIDGET (window->spin_p));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: end\n");
#endif
}

void
window_set_algorithm ()
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: start\n");
#endif
  i = window_get_algorithm ();
  switch (i)
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].
nsweeps);
      break;
    case ALGORITHM_GENETIC:
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_bits,
                                 (gdouble) input->variable[i].nbits);
    }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: end\n");
#endif
}

void
window_set_experiment ()
{
  unsigned int i, j;
  char *buffer1, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
  buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
  buffer2 = g_build_filename (input->directory, buffer1, NULL);
  g_free (buffer1);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_file_chooser_set_filename
    (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_free (buffer2);
  for (j = 0; j < input->experiment->ninputs; ++j)
    {
      g_signal_handler_block (window->button_template[j], window->id_input[j]);
      buffer2 =
        g_build_filename (input->directory, input->experiment[i].stencil[j],
                          NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_template[j]), buffer2);
      g_free (buffer2);
      g_signal_handler_unblock
        (window->button_template[j], window->id_input[j]);
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: end\n");
#endif
}

void
window_remove_experiment ()
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  experiment_free (input->experiment + i, input->type);
  --input->nexperiments;
  for (j = i; j < input->nexperiments; ++j)
    memcpy (input->experiment + j, input->experiment + j + 1,
            sizeof (Experiment));
  j = input->nexperiments - 1;
  if (i > j)
    i = j;
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_experiment: end\n");
#endif
}

void
window_add_experiment ()
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_insert_text
    (window->combo_experiment, i, input->experiment[i].name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  input->experiment = (Experiment *) g_realloc
    (input->experiment, (input->nexperiments + 1) * sizeof (Experiment));
  for (j = input->nexperiments - 1; j > i; --j)
    memcpy (input->experiment + j + 1, input->experiment + j,
            sizeof (Experiment));
  input->experiment[j + 1].weight = input->experiment[j].weight;
  input->experiment[j + 1].ninputs = input->experiment[j].ninputs;
  if (input->type == INPUT_TYPE_XML)
    {
      input->experiment[j + 1].name
        = (char *) xmlStrdup ((xmlChar *) input->experiment[j].name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = (char *) xmlStrdup ((xmlChar *) input->experiment[i].stencil[j]);
    }
  else
    {
      input->experiment[j + 1].name = g_strdup (input->experiment[j].name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = g_strdup (input->experiment[i].stencil[j]);
    }
  ++input->nexperiments;
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_block (window->button_template[j], window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j], window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: end\n");
#endif
}

void
window_name_experiment ()
{
  unsigned int i;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: end\n");
#endif
}

void
window_weight_experiment ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  input->experiment[i].weight = gtk_spin_button_get_value (window->spin_weight);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: end\n");
#endif
}

void
window_inputs_experiment ()
```

```
{
  unsigned int j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: start\n");
#endif
  j = input->experiment->ninputs - 1;
  if (j
      && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                        (window->check_template[j])))
    --input->experiment->ninputs;
  if (input->experiment->ninputs < MAX_NINPUTS
      && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                       (window->check_template[j])))
    ++input->experiment->ninputs;
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: end\n");
#endif
}

void
window_template_experiment (void *data)
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  if (input->type == INPUT_TYPE_XML)
    input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
  else
    input->experiment[j].stencil[i] = g_strdup (buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}

void
window_set_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->entry_variable, window->
id_variable_label);
  gtk_entry_set_text (window->entry_variable, input->variable[i].
name);
  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
  gtk_spin_button_set_value (window->spin_min, input->variable[i].
rangemin);
  gtk_spin_button_set_value (window->spin_max, input->variable[i].
rangemax);
  if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_minabs,
                                 input->variable[i].rangeminabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
    }
  if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_maxabs,
                                 input->variable[i].rangemaxabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
    }
  else
    {
```

```c
      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
    }
  gtk_spin_button_set_value (window->spin_precision,
                             input->variable[i].precision);
  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
  if (input->nsteps)
    gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
           input->variable[i].precision);
#endif
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].nsweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      gtk_spin_button_set_value (window->spin_bits,
                                 (gdouble) input->variable[i].nbits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
               input->variable[i].nbits);
#endif
      break;
    }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: end\n");
#endif
}

void
window_remove_variable ()
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  xmlFree (input->variable[i].name);
  --input->nvariables;
  for (j = i; j < input->nvariables; ++j)
    memcpy (input->variable + j, input->variable + j + 1, sizeof (Variable));
  j = input->nvariables - 1;
  if (i > j)
    i = j;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: end\n");
#endif
}

void
window_add_variable ()
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_insert_text (window->combo_variable, i,
                                  input->variable[i].name);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
  input->variable = (Variable *) g_realloc
    (input->variable, (input->nvariables + 1) * sizeof (Variable));
  for (j = input->nvariables - 1; j > i; --j)
    memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
  memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
  if (input->type == INPUT_TYPE_XML)
    input->variable[j + 1].name
      = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
  else
    input->variable[j + 1].name = g_strdup (input->variable[j].name);
  ++input->nvariables;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: end\n");
#endif
}

void
window_label_variable ()
{
  unsigned int i;
  const char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  buffer = gtk_entry_get_text (window->entry_variable);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: end\n");
#endif
}

void
window_precision_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].precision
    = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
  gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_minabs,
                              input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_maxabs,
                              input->variable[i].precision);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: end\n");
#endif
}

void
window_rangemin_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemin = gtk_spin_button_get_value (window->spin_min);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: end\n");
#endif
}

```

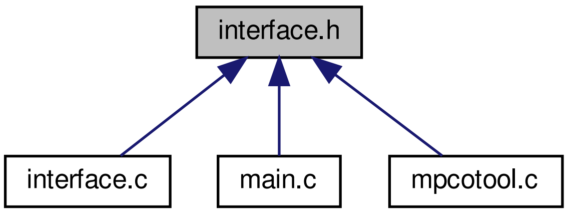
```
void
window_rangemax_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemax = gtk_spin_button_get_value (window->
spin_max);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: end\n");
#endif
}

void
window_rangeminabs_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangeminabs
    = gtk_spin_button_get_value (window->spin_minabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: end\n");
#endif
}

void
window_rangemaxabs_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemaxabs
    = gtk_spin_button_get_value (window->spin_maxabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: end\n");
#endif
}

void
window_step_variable ()
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_step_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].step = gtk_spin_button_get_value (window->
spin_step);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_step_variable: end\n");
#endif
}

void
window_update_variable ()
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  if (i < 0)
    i = 0;
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      input->variable[i].nsweeps
        = gtk_spin_button_get_value_as_int (window->spin_sweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      input->variable[i].nbits
        = gtk_spin_button_get_value_as_int (window->spin_bits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
               input->variable[i].nbits);
```

```
#endif
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: end\n");
#endif
}

int
window_read (char *filename)
{
  unsigned int i;
  char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  input->result = input->variables = NULL;
  if (!input_open (filename))
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_read: end\n");
#endif
      return 0;
    }

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->
variables);
  buffer = g_build_filename (input->directory, input->
simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->
evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
      // fallthrough
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_climbing),
input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_climbing[input->
climbing]),
             TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->climbing)
            {
            case CLIMBING_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
  gtk_spin_button_set_value (window->spin_p, input->p);
  gtk_spin_button_set_value (window->spin_threshold, input->threshold);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i].name);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable,
                                    input->variable[i].name);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG_INTERFACE
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}

void
window_open ()
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer, *directory, *name;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: start\n");
#endif

  // Saving a backup of the current input file
  directory = g_strdup (input->directory);
  name = g_strdup (input->name);

  // Opening dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Open input file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_OPEN,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // Adding JSON filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "JSON");
  gtk_file_filter_add_pattern (filter, "*.json");
  gtk_file_filter_add_pattern (filter, "*.JSON");
  gtk_file_filter_add_pattern (filter, "*.js");
  gtk_file_filter_add_pattern (filter, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK saving
  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Traying to open the input file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      if (!window_read (buffer))
        {
#if DEBUG_INTERFACE
          fprintf (stderr, "window_open: error reading input file\n");
#endif
          g_free (buffer);

          // Reading backup file on error
          buffer = g_build_filename (directory, name, NULL);
          input->result = input->variables = NULL;
          if (!input_open (buffer))
            {

              // Closing on backup file reading error
#if DEBUG_INTERFACE
              fprintf (stderr, "window_read: error reading backup file\n");
#endif
              g_free (buffer);
              break;
            }
          g_free (buffer);
        }
      else
        {
          g_free (buffer);
          break;
        }
    }

  // Freeing and closing
  g_free (name);
  g_free (directory);
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: end\n");
#endif
}

void
window_new (GtkApplication * application)
{
  unsigned int i;
  char *buffer, *buffer2, buffer3[64];
  char *label_algorithm[NALGORITHMS] = {
    "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
  };
  char *tip_algorithm[NALGORITHMS] = {
    _("Monte-Carlo brute force algorithm"),
    _("Sweep brute force algorithm"),
    _("Genetic algorithm"),
    _("Orthogonal sampling brute force algorithm"),
  };
  char *label_climbing[NCLIMBINGS] = {
    _("_Coordinates climbing"), _("_Random climbing")
  };
  char *tip_climbing[NCLIMBINGS] = {
    _("Coordinates climbing estimate method"),
    _("Random climbing estimate method")
  };
  char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
  char *tip_norm[NNORMS] = {
    _("Euclidean error norm (L2)"),
    _("Maximum error norm (L)"),
    _("P error norm (Lp)"),
    _("Taxicab error norm (L1)")
  };

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif

  // Creating the window
  window->window = main_window
    = (GtkWindow *) gtk_application_window_new (application);

  // Finish when closing the window
  g_signal_connect_swapped (window->window, "delete-event",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Setting the window title
  gtk_window_set_title (window->window, "MPCOTool");

  // Creating the open button
  window->button_open = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-open",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
  g_signal_connect (window->button_open, "clicked", window_open, NULL);

  // Creating the save button
  window->button_save = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-save",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
  g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
                    NULL);

  // Creating the run button
  window->button_run = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("system-run",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
  g_signal_connect (window->button_run, "clicked", window_run, NULL);

  // Creating the options button
  window->button_options = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("preferences-system",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
  g_signal_connect (window->button_options, "clicked", options_new, NULL);

  // Creating the help button
  window->button_help = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-browser",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
  g_signal_connect (window->button_help, "clicked", window_help, NULL);

  // Creating the about button
  window->button_about = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-about",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
  g_signal_connect (window->button_about, "clicked", window_about, NULL);

  // Creating the exit button
  window->button_exit = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("application-exit",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
  g_signal_connect_swapped (window->button_exit, "clicked",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Creating the buttons bar
  window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
  gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);

  // Creating the simulator program label and entry
  window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
  window->button_simulator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Simulator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
                               _("Simulator program executable file"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);

  // Creating the evaluator program label and entry
  window->check_evaluator = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
  g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
  window->button_evaluator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Evaluator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_evaluator),
     _("Optional evaluator program executable file"));

  // Creating the results files labels and entries
  window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
  window->entry_result = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_result), _("Best results file"));
  window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
  window->entry_variables = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variables), _("All simulated results file"));

  // Creating the files grid and attaching widgets
  window->grid_files = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_simulator),
                   0, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_simulator),
                   1, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->check_evaluator),
                   0, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->button_evaluator),
                   1, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_result),
                   0, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_result),
                   1, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->label_variables),
                   0, 3, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->entry_variables),
                   1, 3, 1, 1);

  // Creating the algorithm properties
  window->label_simulations = (GtkLabel *) gtk_label_new
    (_("Simulations number"));
  window->spin_simulations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_simulations),
     _("Number of simulations to perform for each iteration"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
  window->label_iterations = (GtkLabel *)
    gtk_label_new (_("Iterations number"));
  window->spin_iterations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
  g_signal_connect
    (window->spin_iterations, "value-changed", window_update, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
  window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
  window->spin_tolerance =
    (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_tolerance),
     _("Tolerance to set the variable interval on the next iteration"));
  window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
  window->spin_bests
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bests),
     _("Number of best simulations used to set the variable interval "
       "on the next iteration"));
  window->label_population
    = (GtkLabel *) gtk_label_new (_("Population number"));
  window->spin_population
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_population),
     _("Number of population for the genetic algorithm"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
  window->label_generations
    = (GtkLabel *) gtk_label_new (_("Generations number"));
  window->spin_generations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_generations),
     _("Number of generations for the genetic algorithm"));
  window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
  window->spin_mutation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_mutation),
     _("Ratio of mutation for the genetic algorithm"));
  window->label_reproduction
    = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
  window->spin_reproduction
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_reproduction),
     _("Ratio of reproduction for the genetic algorithm"));
  window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
  window->spin_adaptation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_adaptation),
     _("Ratio of adaptation for the genetic algorithm"));
  window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
  window->spin_threshold = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
                                    precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_threshold),
     _("Threshold in the objective function to finish the simulations"));
  window->scrolled_threshold =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
                     GTK_WIDGET (window->spin_threshold));
//  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
//  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
//                               GTK_ALIGN_FILL);

  // Creating the hill climbing method properties
  window->check_climbing = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
  g_signal_connect (window->check_climbing, "clicked",
window_update, NULL);
  window->grid_climbing = (GtkGrid *) gtk_grid_new ();
  window->button_climbing[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
  gtk_grid_attach (window->grid_climbing,
                   GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_climbing[0], "clicked",
window_update, NULL);
  for (i = 0; ++i < NCLIMBINGS;)
    {
      window->button_climbing[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_climbing[0]),
         label_climbing[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
                                   tip_climbing[i]);
      gtk_grid_attach (window->grid_climbing,
                       GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
      g_signal_connect (window->button_climbing[i], "clicked",
window_update,
                        NULL);
    }
  window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
  window->spin_steps = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
  window->label_estimates
    = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
  window->spin_estimates = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e3, 1.);
  window->label_relaxation
    = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
  window->spin_relaxation = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., 2., 0.001);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_steps),
                   0, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_steps),
                   1, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_estimates),
                   0, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_estimates),
                   1, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
label_relaxation),
                   0, NCLIMBINGS + 2, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->
spin_relaxation),
                   1, NCLIMBINGS + 2, 1, 1);

  // Creating the array of algorithms
  window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
  window->button_algorithm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
                               tip_algorithm[0]);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_algorithm[0], "clicked",
                    window_set_algorithm, NULL);
  for (i = 0; ++i < NALGORITHMS;)
    {
      window->button_algorithm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_algorithm[0]),
         label_algorithm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
                                   tip_algorithm[i]);
      gtk_grid_attach (window->grid_algorithm,
                       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_algorithm[i], "clicked",
                        window_set_algorithm, NULL);
    }
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_simulations),
                   0, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_iterations),
                   0, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_iterations),
                   1, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_tolerance),
                   0, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_tolerance),
                   1, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_bests),
                   0, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_bests),
                   1, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_population),
                   0, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_population),
                   1, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_generations),
                   0, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_generations),
                   1, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_mutation),
                   0, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_mutation),
                   1, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_reproduction),
                   0, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_reproduction),
                   1, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_adaptation),
                   0, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->spin_adaptation),
                   1, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->check_climbing),
                   0, NALGORITHMS + 9, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->grid_climbing),
                   0, NALGORITHMS + 10, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->label_threshold),
                   0, NALGORITHMS + 11, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->scrolled_threshold),
                   1, NALGORITHMS + 11, 1, 1);
  window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
  gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
                     GTK_WIDGET (window->grid_algorithm));

  // Creating the variable widgets
  window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->combo_variable), _("Variables selector"));
  window->id_variable = g_signal_connect
    (window->combo_variable, "changed", window_set_variable, NULL);
  window->button_add_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_add_variable, "clicked",
window_add_variable,
                    NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
                               _("Add variable"));
  window->button_remove_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_variable, "clicked",
                    window_remove_variable, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
                               _("Remove variable"));
  window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
  window->entry_variable = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variable), _("Variable name"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
  window->id_variable_label = g_signal_connect
    (window->entry_variable, "changed", window_label_variable, NULL);
  window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
  window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
  window->scrolled_min
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_min),
                     GTK_WIDGET (window->spin_min));
  g_signal_connect (window->spin_min, "value-changed",
                    window_rangemin_variable, NULL);
  window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
  window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
  window->scrolled_max
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_max),
                       GTK_WIDGET (window->spin_max));
  g_signal_connect (window->spin_max, "value-changed",
                    window_rangemax_variable, NULL);
  window->check_minabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
  g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
  window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_minabs),
     _("Minimum allowed value of the variable"));
  window->scrolled_minabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
                     GTK_WIDGET (window->spin_minabs));
  g_signal_connect (window->spin_minabs, "value-changed",
                    window_rangeminabs_variable, NULL);
  window->check_maxabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
  g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
  window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_maxabs),
     _("Maximum allowed value of the variable"));
  window->scrolled_maxabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
                     GTK_WIDGET (window->spin_maxabs));
  g_signal_connect (window->spin_maxabs, "value-changed",
                    window_rangemaxabs_variable, NULL);
  window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
  window->spin_precision = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_precision),
    _("Number of precision floating point digits\n"
      "0 is for integer numbers"));
  g_signal_connect (window->spin_precision, "value-changed",
                    window_precision_variable, NULL);
  window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
  window->spin_sweeps =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
                               _("Number of steps sweeping the variable"));
  g_signal_connect (window->spin_sweeps, "value-changed",
                    window_update_variable, NULL);
  window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
  window->spin_bits
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bits),
     _("Number of bits to encode the variable"));
  g_signal_connect
    (window->spin_bits, "value-changed", window_update_variable, NULL);
  window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
  window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_step),
     _("Initial step size for the hill climbing method"));
  window->scrolled_step
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_step),
                     GTK_WIDGET (window->spin_step));
  g_signal_connect
    (window->spin_step, "value-changed", window_step_variable, NULL);
  window->grid_variable = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
  window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
  gtk_container_add (GTK_CONTAINER (window->frame_variable),
                     GTK_WIDGET (window->grid_variable));

  // Creating the experiment widgets
  window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
                               _("Experiment selector"));
  window->id_experiment = g_signal_connect
    (window->combo_experiment, "changed", window_set_experiment, NULL)
;
  window->button_add_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
                               _("Add experiment"));
  window->button_remove_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_experiment, "clicked",
                    window_remove_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
                               _("Remove experiment"));
  window->label_experiment
    = (GtkLabel *) gtk_label_new (_("Experimental data file"));
  window->button_experiment = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Experimental data file"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
                               _("Experimental data file"));
  window->id_experiment_name
    = g_signal_connect (window->button_experiment, "selection-changed",
                        window_name_experiment, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
  window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
  window->spin_weight
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_weight),
     _("Weight factor to build the objective function"));
  g_signal_connect
    (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
  window->grid_experiment = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
  for (i = 0; i < MAX_NINPUTS; ++i)
    {
      snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
      window->check_template[i] = (GtkCheckButton *)
        gtk_check_button_new_with_label (buffer3);
      window->id_template[i]
        = g_signal_connect (window->check_template[i], "toggled",
                            window_inputs_experiment, NULL);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
      window->button_template[i] = (GtkFileChooserButton *)
        gtk_file_chooser_button_new (_("Input template"),
                                     GTK_FILE_CHOOSER_ACTION_OPEN);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
                                   _("Experimental input template file"));
      window->id_input[i] =
        g_signal_connect_swapped (window->button_template[i],
                                  "selection-changed",
                                  (GCallback) window_template_experiment,
                                  (void *) (size_t) i);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
    }
  window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
  gtk_container_add (GTK_CONTAINER (window->frame_experiment),
                     GTK_WIDGET (window->grid_experiment));

  // Creating the error norm widgets
  window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
  window->grid_norm = (GtkGrid *) gtk_grid_new ();
  gtk_container_add (GTK_CONTAINER (window->frame_norm),
                     GTK_WIDGET (window->grid_norm));
  window->button_norm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
                               tip_norm[0]);
  gtk_grid_attach (window->grid_norm,
                   GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
  for (i = 0; ++i < NNORMS;)
    {
      window->button_norm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
                                   tip_norm[i]);
      gtk_grid_attach (window->grid_norm,
                       GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
    }
  window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
  window->spin_p = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
                               _("P parameter for the P error norm"));
  window->scrolled_p =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_p),
                     GTK_WIDGET (window->spin_p));
  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
                   1, 2, 1, 2);

  // Creating the grid and attaching the widgets to the grid
  window->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));

  // Setting the window logo
  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
  gtk_window_set_icon (window->window, window->logo);

  // Showing the window
  gtk_widget_show_all (GTK_WIDGET (window->window));

  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
#if GTK_MINOR_VERSION >= 16
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
#endif

  // Reading initial example
  input_new ();
  buffer2 = g_get_current_dir ();
  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
  g_free (buffer2);
  window_read (buffer);
  g_free (buffer);

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif
}
```

4.13 interface.h File Reference

Header file to define the graphical interface functions.

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



Data Structures

- 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

- unsigned int gtk_array_get_active (GtkRadioButton ∗array[ ], unsigned int n)
- void input_save (char ∗filename)
- void options_new ()
- void running_new ()
- unsigned int window_get_algorithm ()
- unsigned int window_get_climbing ()
- unsigned int window_get_norm ()
- void window_save_climbing ()
- int window_save ()
- void window_run ()
- void window_help ()
- void window_update_climbing ()
- void window_update ()
- void window_set_algorithm ()
- void window_set_experiment ()
- void window_remove_experiment ()
- void window_add_experiment ()
- void window_name_experiment ()
- void window_weight_experiment ()

- void window_inputs_experiment ()
- void window_template_experiment (void ∗data)
- void window_set_variable ()
- void window_remove_variable ()
- void window_add_variable ()
- void window_label_variable ()
- void window_precision_variable ()
- void window_rangemin_variable ()
- void window_rangemax_variable ()
- void window_rangeminabs_variable ()
- void window_rangemaxabs_variable ()
- void window_update_variable ()
- int window_read (char ∗filename)
- void window_open ()
- void window_new (GtkApplication ∗application)

Variables

- 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.

4.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file interface.h.

4.13.2 Function Documentation

4.13.2.1 gtk_array_get_active()

```
unsigned int gtk_array_get_active (
              GtkRadioButton * array[ ],
              unsigned int n )
```

Function to get the active GtkRadioButton.

Returns

Active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Definition at line 469 of file utils.c.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

4.13.2.2 input_save()

```
void input_save (
            char * filename )
```

Function to save the input file.

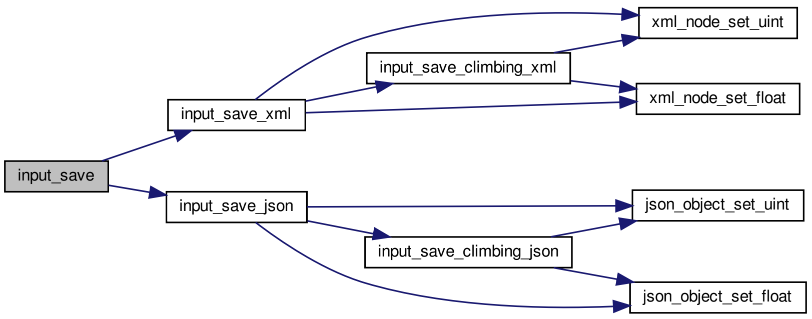
Parameters

filename	Input file name.

Definition at line 584 of file interface.c.

```
{
  xmlDoc *doc;
  JsonGenerator *generator;

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: start\n");
#endif

  // Getting the input file directory
  input->name = g_path_get_basename (filename);
  input->directory = g_path_get_dirname (filename);

  if (input->type == INPUT_TYPE_XML)
    {
      // Opening the input file
      doc = xmlNewDoc ((const xmlChar *) "1.0");
      input_save_xml (doc);

      // Saving the XML file
      xmlSaveFormatFile (filename, doc, 1);

      // Freeing memory
      xmlFreeDoc (doc);
    }
  else
    {
      // Opening the input file
      generator = json_generator_new ();
      json_generator_set_pretty (generator, TRUE);
      input_save_json (generator);

      // Saving the JSON file
      json_generator_to_file (generator, filename, NULL);

      // Freeing memory
      g_object_unref (generator);
    }

#if DEBUG_INTERFACE
  fprintf (stderr, "input_save: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.3 options_new()

```
void options_new ( )
```

Function to open the options dialog.

Definition at line 632 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: start\n");
#endif
  options->label_seed = (GtkLabel *)
    gtk_label_new (_("Pseudo-random numbers generator seed"));
  options->spin_seed = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_seed),
     _("Seed to init the pseudo-random numbers generator"));
  gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
  options->label_threads = (GtkLabel *)
    gtk_label_new (_("Threads number for the stochastic algorithm"));
  options->spin_threads
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_threads),
     _("Number of threads to perform the calibration/optimization for "
       "the stochastic algorithm"));
  gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
  options->label_climbing = (GtkLabel *)
    gtk_label_new (_("Threads number for the hill climbing method"));
  options->spin_climbing =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (options->spin_climbing),
     _("Number of threads to perform the calibration/optimization for the "
       "hill climbing method"));
  gtk_spin_button_set_value (options->spin_climbing,
                                (gdouble) nthreads_climbing);
  options->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_seed), 0, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_seed), 1, 0, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_threads),
                   0, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_threads),
                   1, 1, 1, 1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
label_climbing), 0, 2, 1,
                   1);
  gtk_grid_attach (options->grid, GTK_WIDGET (options->
spin_climbing), 1, 2, 1,
                   1);
  gtk_widget_show_all (GTK_WIDGET (options->grid));
  options->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Options"),
                                 window->window,
                                 GTK_DIALOG_MODAL,
                                 _("_OK"), GTK_RESPONSE_OK,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
  gtk_container_add
    (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
     GTK_WIDGET (options->grid));
  if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
    {
      input->seed
        = (unsigned long int) gtk_spin_button_get_value (options->
spin_seed);
      nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
      nthreads_climbing
        = gtk_spin_button_get_value_as_int (options->spin_climbing);
    }
  gtk_widget_destroy (GTK_WIDGET (options->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "options_new: end\n");
#endif
}
```

4.13.2.4 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: start\n");
#endif
  running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
  running->spinner = (GtkSpinner *) gtk_spinner_new ();
  running->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
  gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
  running->dialog = (GtkDialog *)
    gtk_dialog_new_with_buttons (_("Calculating"),
                                 window->window, GTK_DIALOG_MODAL, NULL, NULL);
  gtk_container_add (GTK_CONTAINER
                     (gtk_dialog_get_content_area (running->dialog)),
                     GTK_WIDGET (running->grid));
  gtk_spinner_start (running->spinner);
  gtk_widget_show_all (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "running_new: end\n");
#endif
}
```

4.13.2.5 window_add_experiment()

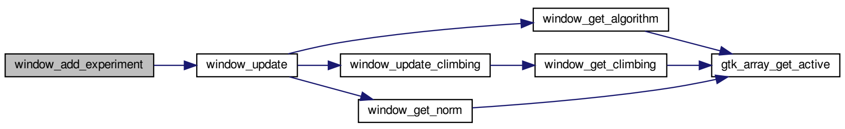
```
void window_add_experiment ( )
```

Function to add an experiment in the main window.

Definition at line 1392 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  gtk_combo_box_text_insert_text
    (window->combo_experiment, i, input->experiment[i].
name);
  g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
  input->experiment = (Experiment *) g_realloc
    (input->experiment, (input->nexperiments + 1) * sizeof (
Experiment));
  for (j = input->nexperiments - 1; j > i; --j)
    memcpy (input->experiment + j + 1, input->experiment + j,
            sizeof (Experiment));
  input->experiment[j + 1].weight = input->experiment[j].
weight;
  input->experiment[j + 1].ninputs = input->
experiment[j].ninputs;
  if (input->type == INPUT_TYPE_XML)
    {
      input->experiment[j + 1].name
        = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
stencil[j]);
    }
  else
    {
      input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
      for (j = 0; j < input->experiment->ninputs; ++j)
        input->experiment[i + 1].stencil[j]
          = g_strdup (input->experiment[i].stencil[j]);
    }
  ++input->nexperiments;
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_block (window->button_template[j],
window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->
id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.6 window_add_variable()

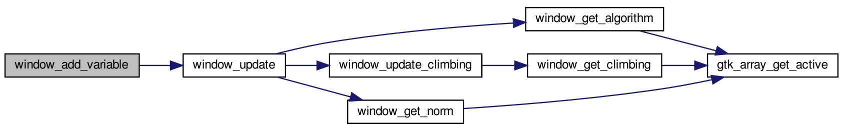
```
void window_add_variable ( )
```

Function to add a variable in the main window.

Definition at line 1655 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->id_variable);
  gtk_combo_box_text_insert_text (window->combo_variable, i,
                                  input->variable[i].name);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  input->variable = (Variable *) g_realloc
    (input->variable, (input->nvariables + 1) * sizeof (Variable));
  for (j = input->nvariables - 1; j > i; --j)
    memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
  memcpy (input->variable + j + 1, input->variable + j, sizeof (Variable));
  if (input->type == INPUT_TYPE_XML)
    input->variable[j + 1].name
      = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
  else
    input->variable[j + 1].name = g_strdup (input->variable[j].name);
  ++input->nvariables;
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_add_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.7 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

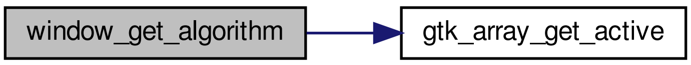
Returns

Stochastic algorithm number.

Definition at line 732 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: start\n");
#endif
  i = gtk_array_get_active (window->button_algorithm,
NALGORITHMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_algorithm: %u\n", i);
  fprintf (stderr, "window_get_algorithm: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.13.2.8 window_get_climbing()

```
unsigned int window_get_climbing ( )
```

Function to get the hill climbing method number.

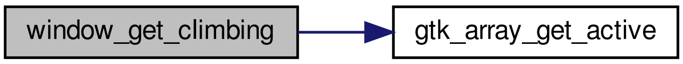
Returns

Hill climbing method number.

Definition at line 752 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: start\n");
#endif
  i = gtk_array_get_active (window->button_climbing,
    NCLIMBINGS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_climbing: %u\n", i);
  fprintf (stderr, "window_get_climbing: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.13.2.9 window_get_norm()

`unsigned int window_get_norm ( )`

Function to get the norm method number.

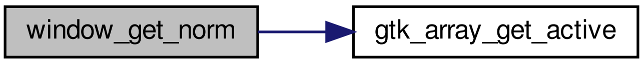
Returns

Norm method number.

Definition at line 772 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: start\n");
#endif
  i = gtk_array_get_active (window->button_norm,
    NNORMS);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_get_norm: %u\n", i);
  fprintf (stderr, "window_get_norm: end\n");
#endif
  return i;
}
```

Here is the call graph for this function:



4.13.2.10 window_help()

```
void window_help ( )
```

Function to show a help dialog.

Definition at line 1029 of file interface.c.

```
{
  char *buffer, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: start\n");
#endif
  buffer2 = g_build_filename (window->application_directory, "..", "manuals",
                              _("user-manual.pdf"), NULL);
  buffer = g_filename_to_uri (buffer2, NULL, NULL);
  g_free (buffer2);
#if GTK_MINOR_VERSION >= 22
  gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
#else
  gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
#endif
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: uri=%s\n", buffer);
#endif
  g_free (buffer);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_help: end\n");
#endif
}
```

4.13.2.11 window_inputs_experiment()

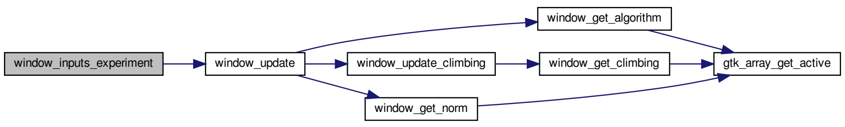
```
void window_inputs_experiment ( )
```

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

Definition at line 1492 of file interface.c.

```
{
  unsigned int j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: start\n");
#endif
  j = input->experiment->ninputs - 1;
  if (j
      && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                         (window->check_template[j])))
    --input->experiment->ninputs;
  if (input->experiment->ninputs < MAX_NINPUTS
      && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                        (window->check_template[j])))
    ++input->experiment->ninputs;
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_inputs_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.12 window_label_variable()

```
void window_label_variable ( )
```

Function to set the variable label in the main window.

Definition at line 1690 of file interface.c.

```
{
  unsigned int i;
  const char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  buffer = gtk_entry_get_text (window->entry_variable);
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_label_variable: end\n");
#endif
}
```

4.13.2.13 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1445 of file interface.c.

```
{
  unsigned int i;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
button_experiment));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_name_experiment: end\n");
#endif
}
```

4.13.2.14 window_new()

```
void window_new (
              GtkApplication * application )
```

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2065 of file interface.c.

```
{
  unsigned int i;
  char *buffer, *buffer2, buffer3[64];
  char *label_algorithm[NALGORITHMS] = {
    "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
  };
  char *tip_algorithm[NALGORITHMS] = {
    _("Monte-Carlo brute force algorithm"),
    _("Sweep brute force algorithm"),
    _("Genetic algorithm"),
    _("Orthogonal sampling brute force algorithm"),
  };
  char *label_climbing[NCLIMBINGS] = {
    _("_Coordinates climbing"), _("_Random climbing")
  };
  char *tip_climbing[NCLIMBINGS] = {
    _("Coordinates climbing estimate method"),
    _("Random climbing estimate method")
  };
  char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
  char *tip_norm[NNORMS] = {
    _("Euclidean error norm (L2)"),
    _("Maximum error norm (L)"),
    _("P error norm (Lp)"),
    _("Taxicab error norm (L1)")
  };

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif

  // Creating the window
  window->window = main_window
    = (GtkWindow *) gtk_application_window_new (application);

  // Finish when closing the window
  g_signal_connect_swapped (window->window, "delete-event",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Setting the window title
  gtk_window_set_title (window->window, "MPCOTool");

  // Creating the open button
  window->button_open = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-open",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
  g_signal_connect (window->button_open, "clicked", window_open, NULL);

  // Creating the save button
  window->button_save = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("document-save",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
  g_signal_connect (window->button_save, "clicked", (GCallback)
window_save,
                    NULL);

  // Creating the run button
  window->button_run = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("system-run",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
  g_signal_connect (window->button_run, "clicked", window_run, NULL);

  // Creating the options button
  window->button_options = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("preferences-system",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
  g_signal_connect (window->button_options, "clicked",
options_new, NULL);

  // Creating the help button
  window->button_help = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-browser",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
  g_signal_connect (window->button_help, "clicked", window_help, NULL);

  // Creating the about button
  window->button_about = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("help-about",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
  g_signal_connect (window->button_about, "clicked",
window_about, NULL);

  // Creating the exit button
  window->button_exit = (GtkToolButton *) gtk_tool_button_new
    (gtk_image_new_from_icon_name ("application-exit",
                                   GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
  g_signal_connect_swapped (window->button_exit, "clicked",
                            G_CALLBACK (g_application_quit),
                            G_APPLICATION (application));

  // Creating the buttons bar
  window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_open), 0);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_save), 1);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_run), 2);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_options), 3);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_help), 4);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_about), 5);
  gtk_toolbar_insert
    (window->bar_buttons, GTK_TOOL_ITEM (window->
button_exit), 6);
  gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);

  // Creating the simulator program label and entry
  window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
  window->button_simulator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Simulator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
                               _("Simulator program executable file"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);

  // Creating the evaluator program label and entry
  window->check_evaluator = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
  g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
  window->button_evaluator = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Evaluator program"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->button_evaluator),
     _("Optional evaluator program executable file"));

  // Creating the results files labels and entries
  window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
  window->entry_result = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_result), _("Best results file"));
  window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
  window->entry_variables = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variables), _("All simulated results file"));

  // Creating the files grid and attaching widgets
  window->grid_files = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
                   0, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
                   1, 0, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
                   0, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
                   1, 1, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
                        0, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
                        1, 2, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
                        0, 3, 1, 1);
  gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
                        1, 3, 1, 1);

  // Creating the algorithm properties
  window->label_simulations = (GtkLabel *) gtk_label_new
    (_("Simulations number"));
  window->spin_simulations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_simulations),
     _("Number of simulations to perform for each iteration"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
  window->label_iterations = (GtkLabel *)
    gtk_label_new (_("Iterations number"));
  window->spin_iterations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
  g_signal_connect
    (window->spin_iterations, "value-changed",
      window_update, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
  window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
  window->spin_tolerance =
    (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_tolerance),
     _("Tolerance to set the variable interval on the next iteration"));
  window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
  window->spin_bests
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bests),
     _("Number of best simulations used to set the variable interval "
       "on the next iteration"));
  window->label_population
    = (GtkLabel *) gtk_label_new (_("Population number"));
  window->spin_population
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_population),
     _("Number of population for the genetic algorithm"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
  window->label_generations
    = (GtkLabel *) gtk_label_new (_("Generations number"));
  window->spin_generations
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_generations),
     _("Number of generations for the genetic algorithm"));
  window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
  window->spin_mutation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_mutation),
     _("Ratio of mutation for the genetic algorithm"));
  window->label_reproduction
    = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
  window->spin_reproduction
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_reproduction),
     _("Ratio of reproduction for the genetic algorithm"));
  window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
  window->spin_adaptation
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_adaptation),
     _("Ratio of adaptation for the genetic algorithm"));
  window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
  window->spin_threshold = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
                                    precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_threshold),
     _("Threshold in the objective function to finish the simulations"));
  window->scrolled_threshold =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
                     GTK_WIDGET (window->spin_threshold));
//  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
//  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
//                               GTK_ALIGN_FILL);

  // Creating the hill climbing method properties
  window->check_climbing = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Hill climbing method"));
  g_signal_connect (window->check_climbing, "clicked", window_update, NULL);
  window->grid_climbing = (GtkGrid *) gtk_grid_new ();
  window->button_climbing[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_climbing[0]);
  gtk_grid_attach (window->grid_climbing,
                   GTK_WIDGET (window->button_climbing[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_climbing[0], "clicked", window_update, NULL);
  for (i = 0; ++i < NCLIMBINGS;)
    {
      window->button_climbing[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_climbing[0]),
         label_climbing[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_climbing[i]),
                                   tip_climbing[i]);
      gtk_grid_attach (window->grid_climbing,
                       GTK_WIDGET (window->button_climbing[i]), 0, i, 1, 1);
      g_signal_connect (window->button_climbing[i], "clicked", window_update,
                        NULL);
    }
  window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
  window->spin_steps = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
  window->label_estimates
    = (GtkLabel *) gtk_label_new (_("Climbing estimates number"));
  window->spin_estimates = (GtkSpinButton *)
    gtk_spin_button_new_with_range (1., 1.e3, 1.);
  window->label_relaxation
    = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
  window->spin_relaxation = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., 2., 0.001);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_steps),
                   0, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_steps),
                   1, NCLIMBINGS, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_estimates),
                   0, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_estimates),
                   1, NCLIMBINGS + 1, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->label_relaxation),
                   0, NCLIMBINGS + 2, 1, 1);
  gtk_grid_attach (window->grid_climbing, GTK_WIDGET (window->spin_relaxation),
                   1, NCLIMBINGS + 2, 1, 1);

  // Creating the array of algorithms
  window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
  window->button_algorithm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
                               tip_algorithm[0]);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_algorithm[0], "clicked",
                    window_set_algorithm, NULL);
  for (i = 0; ++i < NALGORITHMS;)
    {
      window->button_algorithm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_algorithm[0]),
         label_algorithm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
                                   tip_algorithm[i]);
      gtk_grid_attach (window->grid_algorithm,
                       GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_algorithm[i], "clicked",
                        window_set_algorithm, NULL);
    }
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_simulations),
                   0, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_iterations),
                   0, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_iterations),
                   1, NALGORITHMS + 1, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_tolerance),
                   0, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_tolerance),
                   1, NALGORITHMS + 2, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_bests),
                   0, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_bests),
                   1, NALGORITHMS + 3, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_population),
                   0, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_population),
                   1, NALGORITHMS + 4, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_generations),
                   0, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_generations),
                   1, NALGORITHMS + 5, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_mutation),
                   0, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_mutation),
                   1, NALGORITHMS + 6, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_reproduction),
                   0, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->spin_reproduction),
                   1, NALGORITHMS + 7, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->label_adaptation),
                   0, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->spin_adaptation),
                   1, NALGORITHMS + 8, 1, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->check_climbing),
                   0, NALGORITHMS + 9, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->grid_climbing),
                   0, NALGORITHMS + 10, 2, 1);
  gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
window->label_threshold),
                   0, NALGORITHMS + 11, 1, 1);
  gtk_grid_attach (window->grid_algorithm,
                   GTK_WIDGET (window->scrolled_threshold),
                   1, NALGORITHMS + 11, 1, 1);
  window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
  gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
                     GTK_WIDGET (window->grid_algorithm));

  // Creating the variable widgets
  window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->combo_variable), _("Variables selector"));
  window->id_variable = g_signal_connect
    (window->combo_variable, "changed", window_set_variable, NULL);
  window->button_add_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_add_variable, "clicked",
window_add_variable,
                    NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_variable),
                               _("Add variable"));
  window->button_remove_variable = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_variable, "clicked",
                    window_remove_variable, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_variable),
                               _("Remove variable"));
  window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
  window->entry_variable = (GtkEntry *) gtk_entry_new ();
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->entry_variable), _("Variable name"));
  gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
  window->id_variable_label = g_signal_connect
    (window->entry_variable, "changed", window_label_variable, NULL);
  window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
  window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
  window->scrolled_min
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_min),
                     GTK_WIDGET (window->spin_min));
  g_signal_connect (window->spin_min, "value-changed",
                    window_rangemin_variable, NULL);
  window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
  window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
  window->scrolled_max
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_max),
                     GTK_WIDGET (window->spin_max));
  g_signal_connect (window->spin_max, "value-changed",
                    window_rangemax_variable, NULL);
  window->check_minabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
  g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
  window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_minabs),
     _("Minimum allowed value of the variable"));
  window->scrolled_minabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
                     GTK_WIDGET (window->spin_minabs));
  g_signal_connect (window->spin_minabs, "value-changed",
                    window_rangeminabs_variable, NULL);
  window->check_maxabs = (GtkCheckButton *)
    gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
  g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
  window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_maxabs),
     _("Maximum allowed value of the variable"));
  window->scrolled_maxabs
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
                     GTK_WIDGET (window->spin_maxabs));
  g_signal_connect (window->spin_maxabs, "value-changed",
                    window_rangemaxabs_variable, NULL);
  window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
  window->spin_precision = (GtkSpinButton *)
    gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_precision),
     _("Number of precision floating point digits\n"
       "0 is for integer numbers"));
  g_signal_connect (window->spin_precision, "value-changed",
                    window_precision_variable, NULL);
  window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
  window->spin_sweeps =
    (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
                               _("Number of steps sweeping the variable"));
  g_signal_connect (window->spin_sweeps, "value-changed",
                    window_update_variable, NULL);
  window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
  window->spin_bits
    = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_bits),
     _("Number of bits to encode the variable"));
  g_signal_connect
    (window->spin_bits, "value-changed", window_update_variable, NULL);
  window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
  window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
    (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_step),
     _("Initial step size for the hill climbing method"));
  window->scrolled_step
    = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_step),
                     GTK_WIDGET (window->spin_step));
  g_signal_connect
    (window->spin_step, "value-changed", window_step_variable, NULL);
  window->grid_variable = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_min), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_max), 0, 3, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->label_step), 0, 9, 1, 1);
  gtk_grid_attach (window->grid_variable,
                   GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
  window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
  gtk_container_add (GTK_CONTAINER (window->frame_variable),
                     GTK_WIDGET (window->grid_variable));

  // Creating the experiment widgets
  window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
                               _("Experiment selector"));
  window->id_experiment = g_signal_connect
    (window->combo_experiment, "changed",
window_set_experiment, NULL);
  window->button_add_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-add", GTK_ICON_SIZE_BUTTON);
  g_signal_connect
    (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
                               _("Add experiment"));
  window->button_remove_experiment = (GtkButton *)
    gtk_button_new_from_icon_name ("list-remove", GTK_ICON_SIZE_BUTTON);
  g_signal_connect (window->button_remove_experiment, "clicked",
                    window_remove_experiment, NULL);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),
                               _("Remove experiment"));
  window->label_experiment
    = (GtkLabel *) gtk_label_new (_("Experimental data file"));
  window->button_experiment = (GtkFileChooserButton *)
    gtk_file_chooser_button_new (_("Experimental data file"),
                                 GTK_FILE_CHOOSER_ACTION_OPEN);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
                               _("Experimental data file"));
  window->id_experiment_name
    = g_signal_connect (window->button_experiment, "selection-changed",
                        window_name_experiment, NULL);
  gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
  window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
  window->spin_weight
    = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
  gtk_widget_set_tooltip_text
    (GTK_WIDGET (window->spin_weight),
     _("Weight factor to build the objective function"));
  g_signal_connect
    (window->spin_weight, "value-changed", window_weight_experiment, NULL);
  window->grid_experiment = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
  gtk_grid_attach (window->grid_experiment,
                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
  for (i = 0; i < MAX_NINPUTS; ++i)
    {
      snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
      window->check_template[i] = (GtkCheckButton *)
        gtk_check_button_new_with_label (buffer3);
      window->id_template[i]
        = g_signal_connect (window->check_template[i], "toggled",
                            window_inputs_experiment, NULL);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
      window->button_template[i] = (GtkFileChooserButton *)
        gtk_file_chooser_button_new (_("Input template"),
                                     GTK_FILE_CHOOSER_ACTION_OPEN);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
                                   _("Experimental input template file"));
      window->id_input[i] =
        g_signal_connect_swapped (window->button_template[i],
                                  "selection-changed",
                                  (GCallback) window_template_experiment,
                                  (void *) (size_t) i);
      gtk_grid_attach (window->grid_experiment,
                       GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
    }
  window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
  gtk_container_add (GTK_CONTAINER (window->frame_experiment),
                     GTK_WIDGET (window->grid_experiment));

  // Creating the error norm widgets
  window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
  window->grid_norm = (GtkGrid *) gtk_grid_new ();
  gtk_container_add (GTK_CONTAINER (window->frame_norm),
                     GTK_WIDGET (window->grid_norm));
  window->button_norm[0] = (GtkRadioButton *)
    gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
                               tip_norm[0]);
  gtk_grid_attach (window->grid_norm,
                   GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
  g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
  for (i = 0; ++i < NNORMS;)
    {
      window->button_norm[i] = (GtkRadioButton *)
        gtk_radio_button_new_with_mnemonic
        (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
      gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
                                   tip_norm[i]);
      gtk_grid_attach (window->grid_norm,
                       GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
      g_signal_connect (window->button_norm[i], "clicked", window_update, NULL);
    }
  window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
  window->spin_p = (GtkSpinButton *)
    gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
  gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
                               _("P parameter for the P error norm"));
  window->scrolled_p =
    (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
  gtk_container_add (GTK_CONTAINER (window->scrolled_p),
                     GTK_WIDGET (window->spin_p));
  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
  gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
                   1, 2, 1, 2);

  // Creating the grid and attaching the widgets to the grid
  window->grid = (GtkGrid *) gtk_grid_new ();
  gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
  gtk_grid_attach (window->grid,
                   GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
  gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
  gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->grid));

  // Setting the window logo
  window->logo = gdk_pixbuf_new_from_xpm_data (logo);
  gtk_window_set_icon (window->window, window->logo);

  // Showing the window
  gtk_widget_show_all (GTK_WIDGET (window->window));

  // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
#if GTK_MINOR_VERSION >= 16
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
#endif

  // Reading initial example
  input_new ();
  buffer2 = g_get_current_dir ();
  buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
  g_free (buffer2);
  window_read (buffer);
  g_free (buffer);

#if DEBUG_INTERFACE
  fprintf (stderr, "window_new: start\n");
#endif
}
```

4.13.2.15 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1979 of file interface.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter;
  char *buffer, *directory, *name;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: start\n");
#endif

  // Saving a backup of the current input file
  directory = g_strdup (input->directory);
  name = g_strdup (input->name);

  // Opening dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Open input file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_OPEN,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);

  // Adding XML filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "XML");
  gtk_file_filter_add_pattern (filter, "*.xml");
  gtk_file_filter_add_pattern (filter, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // Adding JSON filter
  filter = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter, "JSON");
  gtk_file_filter_add_pattern (filter, "*.json");
  gtk_file_filter_add_pattern (filter, "*.JSON");
  gtk_file_filter_add_pattern (filter, "*.js");
  gtk_file_filter_add_pattern (filter, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);

  // If OK saving
  while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {

      // Traying to open the input file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      if (!window_read (buffer))
        {
#if DEBUG_INTERFACE
          fprintf (stderr, "window_open: error reading input file\n");
#endif
          g_free (buffer);

          // Reading backup file on error
          buffer = g_build_filename (directory, name, NULL);
          input->result = input->variables = NULL;
          if (!input_open (buffer))
            {

              // Closing on backup file reading error
#if DEBUG_INTERFACE
              fprintf (stderr, "window_read: error reading backup file\n");
#endif
              g_free (buffer);
              break;
            }
          g_free (buffer);
        }
      else
        {
          g_free (buffer);
          break;
        }
    }

  // Freeing and closing
  g_free (name);
  g_free (directory);
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_open: end\n");
#endif
}
```

4.13.2.16 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1713 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].precision
    = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
  gtk_spin_button_set_digits (window->spin_min, input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_max, input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_minabs,
                              input->variable[i].precision);
  gtk_spin_button_set_digits (window->spin_maxabs,
                              input->variable[i].precision);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_precision_variable: end\n");
#endif
}
```

4.13.2.17 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1754 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemax = gtk_spin_button_get_value (window->spin_max);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemax_variable: end\n");
#endif
}
```

4.13.2.18 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1789 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemaxabs
    = gtk_spin_button_get_value (window->spin_maxabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemaxabs_variable: end\n");
#endif
}
```

4.13.2.19 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1737 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangemin = gtk_spin_button_get_value (
window->spin_min);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangemin_variable: end\n");
#endif
}
```

4.13.2.20 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1771 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  input->variable[i].rangeminabs
    = gtk_spin_button_get_value (window->spin_minabs);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_rangeminabs_variable: end\n");
#endif
}
```

4.13.2.21 window_read()

```
int window_read (
             char * filename )
```

Function to read the input data of a file.

Returns

1 on succes, 0 on error.

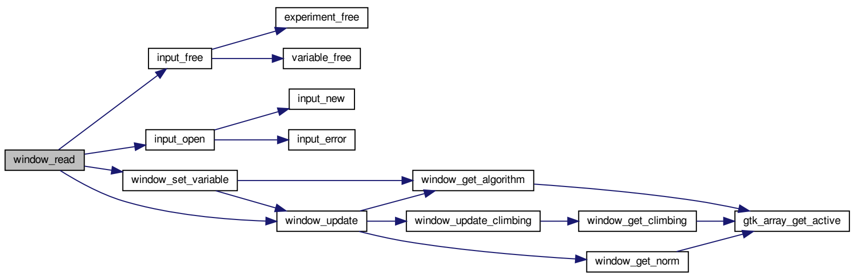
Parameters

filename	File name.

Definition at line 1863 of file interface.c.

```
{
  unsigned int i;
  char *buffer;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_read: start\n");
#endif

  // Reading new input file
  input_free ();
  input->result = input->variables = NULL;
  if (!input_open (filename))
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_read: end\n");
#endif
      return 0;
    }

  // Setting GTK+ widgets data
  gtk_entry_set_text (window->entry_result, input->result);
  gtk_entry_set_text (window->entry_variables, input->variables);
  buffer = g_build_filename (input->directory, input->simulator, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                 (window->button_simulator), buffer);
  g_free (buffer);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
                                (size_t) input->evaluator);
  if (input->evaluator)
    {
      buffer = g_build_filename (input->directory, input->evaluator, NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_evaluator), buffer);
      g_free (buffer);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_algorithm[input->algorithm]), TRUE);
  switch (input->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_spin_button_set_value (window->spin_simulations,
                                 (gdouble) input->nsimulations);
      // fallthrough
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_iterations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_bests, (gdouble) input->nbest);
      gtk_spin_button_set_value (window->spin_tolerance, input->tolerance);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_climbing), input->nsteps);
      if (input->nsteps)
        {
          gtk_toggle_button_set_active
            (GTK_TOGGLE_BUTTON (window->button_climbing[input->climbing]),
             TRUE);
          gtk_spin_button_set_value (window->spin_steps,
                                     (gdouble) input->nsteps);
          gtk_spin_button_set_value (window->spin_relaxation,
                                     (gdouble) input->relaxation);
          switch (input->climbing)
            {
            case CLIMBING_METHOD_RANDOM:
              gtk_spin_button_set_value (window->spin_estimates,
                                         (gdouble) input->nestimates);
            }
        }
      break;
    default:
      gtk_spin_button_set_value (window->spin_population,
                                 (gdouble) input->nsimulations);
      gtk_spin_button_set_value (window->spin_generations,
                                 (gdouble) input->niterations);
      gtk_spin_button_set_value (window->spin_mutation, input->mutation_ratio);
      gtk_spin_button_set_value (window->spin_reproduction,
                                 input->reproduction_ratio);
      gtk_spin_button_set_value (window->spin_adaptation,
                                 input->adaptation_ratio);
    }
  gtk_toggle_button_set_active
    (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
  gtk_spin_button_set_value (window->spin_p, input->p);
  gtk_spin_button_set_value (window->spin_threshold, input->threshold);
  g_signal_handler_block (window->combo_experiment, window->id_experiment);
  g_signal_handler_block (window->button_experiment,
                          window->id_experiment_name);
  gtk_combo_box_text_remove_all (window->combo_experiment);
  for (i = 0; i < input->nexperiments; ++i)
    gtk_combo_box_text_append_text (window->combo_experiment,
                                    input->experiment[i].name);
  g_signal_handler_unblock
    (window->button_experiment, window->id_experiment_name);
  g_signal_handler_unblock (window->combo_experiment, window->id_experiment);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
  g_signal_handler_block (window->combo_variable, window->id_variable);
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_combo_box_text_remove_all (window->combo_variable);
  for (i = 0; i < input->nvariables; ++i)
    gtk_combo_box_text_append_text (window->combo_variable,
                                    input->variable[i].name);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  g_signal_handler_unblock (window->combo_variable, window->id_variable);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
  window_set_variable ();
  window_update ();

#if DEBUG_INTERFACE
  fprintf (stderr, "window_read: end\n");
#endif
  return 1;
}
```

Here is the call graph for this function:



4.13.2.22 window_remove_experiment()

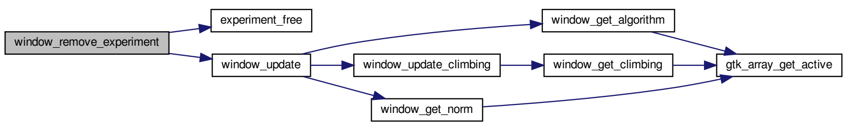
```
void window_remove_experiment ( )
```

Function to remove an experiment in the main window.

Definition at line 1355 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  g_signal_handler_block (window->combo_experiment, window->
id_experiment);
  gtk_combo_box_text_remove (window->combo_experiment, i);
  g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
  experiment_free (input->experiment + i, input->
type);
  --input->nexperiments;
  for (j = i; j < input->nexperiments; ++j)
    memcpy (input->experiment + j, input->experiment + j + 1,
            sizeof (Experiment));
  j = input->nexperiments - 1;
  if (i > j)
    i = j;
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_block (window->button_template[j],
window->id_input[j]);
  g_signal_handler_block
    (window->button_experiment, window->
id_experiment_name);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  for (j = 0; j < input->experiment->ninputs; ++j)
    g_signal_handler_unblock (window->button_template[j],
window->id_input[j]);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_experiment: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.23 window_remove_variable()

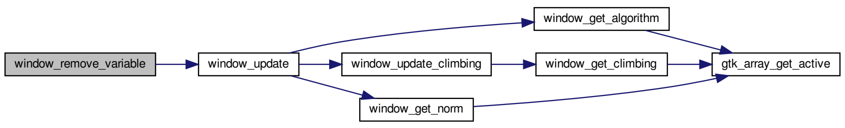
```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

Definition at line 1625 of file interface.c.

```
{
  unsigned int i, j;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->combo_variable, window->
id_variable);
  gtk_combo_box_text_remove (window->combo_variable, i);
  g_signal_handler_unblock (window->combo_variable, window->
id_variable);
  xmlFree (input->variable[i].name);
  --input->nvariables;
  for (j = i; j < input->nvariables; ++j)
    memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
  j = input->nvariables - 1;
  if (i > j)
    i = j;
  g_signal_handler_block (window->entry_variable, window->
id_variable_label);
  gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
  g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_remove_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.24 window_run()

```
void window_run ( )
```

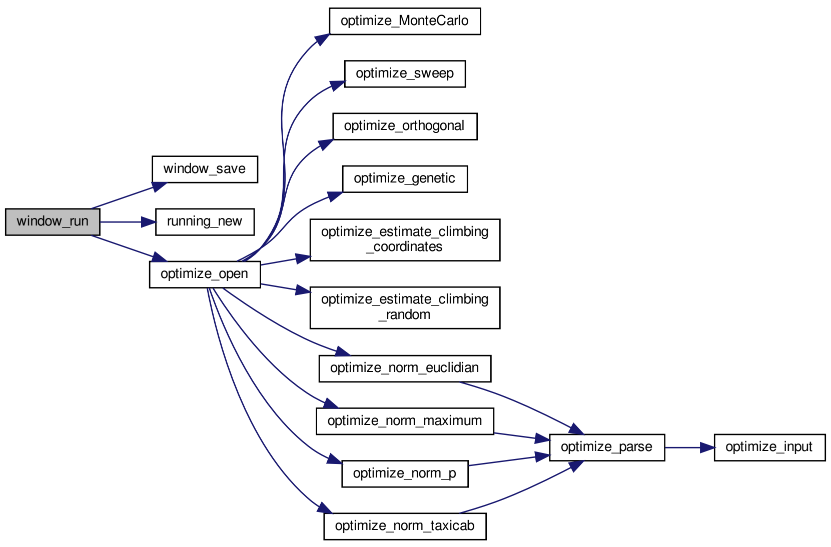
Function to run a optimization.

Definition at line 974 of file interface.c.

```
{
  unsigned int i;
  char *msg, *msg2, buffer[64], buffer2[64];
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: start\n");
#endif
  if (!window_save ())
    {
#if DEBUG_INTERFACE
      fprintf (stderr, "window_run: end\n");
#endif
      return;
    }
  running_new ();
  while (gtk_events_pending ())
    gtk_main_iteration ();
  optimize_open ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: closing running dialog\n");
#endif
```

```
  gtk_spinner_stop (running->spinner);
  gtk_widget_destroy (GTK_WIDGET (running->dialog));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: displaying results\n");
#endif
  snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
  msg2 = g_strdup (buffer);
  for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
    {
      snprintf (buffer, 64, "%s = %s\n",
                input->variable[i].name, format[input->
variable[i].precision]);
      snprintf (buffer2, 64, buffer, optimize->value_old[i]);
      msg = g_strconcat (msg2, buffer2, NULL);
      g_free (msg2);
    }
  snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
            optimize->calculation_time);
  msg = g_strconcat (msg2, buffer, NULL);
  g_free (msg2);
  show_message (_("Best result"), msg, INFO_TYPE);
  g_free (msg);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: freeing memory\n");
#endif
  optimize_free ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_run: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.25 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 823 of file interface.c.

```
{
  GtkFileChooserDialog *dlg;
  GtkFileFilter *filter1, *filter2;
  char *buffer;

#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: start\n");
#endif

  // Opening the saving dialog
  dlg = (GtkFileChooserDialog *)
    gtk_file_chooser_dialog_new (_("Save file"),
                                 window->window,
                                 GTK_FILE_CHOOSER_ACTION_SAVE,
                                 _("_Cancel"), GTK_RESPONSE_CANCEL,
                                 _("_OK"), GTK_RESPONSE_OK, NULL);
  gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
  buffer = g_build_filename (input->directory, input->name, NULL);
  gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
  g_free (buffer);

  // Adding XML filter
  filter1 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter1, "XML");
  gtk_file_filter_add_pattern (filter1, "*.xml");
  gtk_file_filter_add_pattern (filter1, "*.XML");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);

  // Adding JSON filter
  filter2 = (GtkFileFilter *) gtk_file_filter_new ();
  gtk_file_filter_set_name (filter2, "JSON");
  gtk_file_filter_add_pattern (filter2, "*.json");
  gtk_file_filter_add_pattern (filter2, "*.JSON");
  gtk_file_filter_add_pattern (filter2, "*.js");
  gtk_file_filter_add_pattern (filter2, "*.JS");
  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);

  if (input->type == INPUT_TYPE_XML)
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
  else
    gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);

  // If OK response then saving
  if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
    {
      // Setting input file type
      filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
      buffer = (char *) gtk_file_filter_get_name (filter1);
      if (!strcmp (buffer, "XML"))
        input->type = INPUT_TYPE_XML;
      else
        input->type = INPUT_TYPE_JSON;

      // Adding properties to the root XML node
      input->simulator = gtk_file_chooser_get_filename
        (GTK_FILE_CHOOSER (window->button_simulator));
      if (gtk_toggle_button_get_active
          (GTK_TOGGLE_BUTTON (window->check_evaluator)))
        input->evaluator = gtk_file_chooser_get_filename
          (GTK_FILE_CHOOSER (window->button_evaluator));
      else
        input->evaluator = NULL;
      if (input->type == INPUT_TYPE_XML)
        {
          input->result
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->entry_result));
          input->variables
            = (char *) xmlStrdup ((const xmlChar *)
                                  gtk_entry_get_text (window->
entry_variables));
        }
      else
        {
          input->result = g_strdup (gtk_entry_get_text (window->
entry_result));
          input->variables =
            g_strdup (gtk_entry_get_text (window->entry_variables));
        }

      // Setting the algorithm
      switch (window_get_algorithm ())
        {
        case ALGORITHM_MONTE_CARLO:
          input->algorithm = ALGORITHM_MONTE_CARLO;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_simulations);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_SWEEP:
          input->algorithm = ALGORITHM_SWEEP;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        case ALGORITHM_ORTHOGONAL:
          input->algorithm = ALGORITHM_ORTHOGONAL;
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_iterations);
          input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
          input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
          window_save_climbing ();
          break;
        default:
          input->algorithm = ALGORITHM_GENETIC;
          input->nsimulations
            = gtk_spin_button_get_value_as_int (window->spin_population);
          input->niterations
            = gtk_spin_button_get_value_as_int (window->spin_generations);
          input->mutation_ratio
            = gtk_spin_button_get_value (window->spin_mutation);
          input->reproduction_ratio
            = gtk_spin_button_get_value (window->spin_reproduction);
          input->adaptation_ratio
            = gtk_spin_button_get_value (window->spin_adaptation);
          break;
        }
      input->norm = window_get_norm ();
      input->p = gtk_spin_button_get_value (window->spin_p);
      input->threshold = gtk_spin_button_get_value (window->
spin_threshold);

      // Saving the XML file
      buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
      input_save (buffer);

      // Closing and freeing memory
      g_free (buffer);
      gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
      fprintf (stderr, "window_save: end\n");
#endif
      return 1;
    }

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save: end\n");
#endif
  return 0;
}
```

4.13.2.26 window_save_climbing()

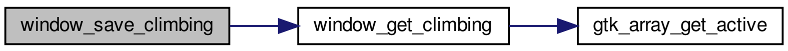
```
void window_save_climbing ( )
```

Function to save the hill climbing method data in the input file.

Definition at line 790 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: start\n");
#endif
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
check_climbing)))
    {
      input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
      input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
      switch (window_get_climbing ())
        {
        case CLIMBING_METHOD_COORDINATES:
          input->climbing = CLIMBING_METHOD_COORDINATES;
          break;
        default:
          input->climbing = CLIMBING_METHOD_RANDOM;
          input->nestimates
            = gtk_spin_button_get_value_as_int (window->spin_estimates);
        }
    }
  else
    input->nsteps = 0;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_save_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.27 window_set_algorithm()

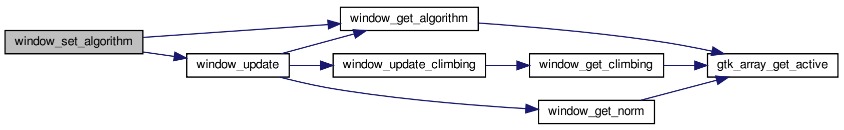
```
void window_set_algorithm ( )
```

Function to avoid memory errors changing the algorithm.

Definition at line 1281 of file interface.c.

```
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: start\n");
#endif
  i = window_get_algorithm ();
  switch (i)
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
      if (i < 0)
        i = 0;
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].
      nsweeps);
        break;
      case ALGORITHM_GENETIC:
        i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
        if (i < 0)
          i = 0;
        gtk_spin_button_set_value (window->spin_bits,
                                   (gdouble) input->variable[i].nbits);
      }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_algorithm: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.28 window_set_experiment()

```
void window_set_experiment ( )
```

Function to set the experiment data in the main window.

Definition at line 1315 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer1, *buffer2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
  buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
  buffer2 = g_build_filename (input->directory, buffer1, NULL);
  g_free (buffer1);
  g_signal_handler_block
    (window->button_experiment, window->
id_experiment_name);
  gtk_file_chooser_set_filename
    (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
  g_signal_handler_unblock
    (window->button_experiment, window->
id_experiment_name);
  g_free (buffer2);
  for (j = 0; j < input->experiment->ninputs; ++j)
    {
      g_signal_handler_block (window->button_template[j],
window->id_input[j]);
      buffer2 =
        g_build_filename (input->directory, input->experiment[i].
stencil[j],
                          NULL);
      gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
                                     (window->button_template[j]), buffer2);
      g_free (buffer2);
      g_signal_handler_unblock
        (window->button_template[j], window->id_input[j]);
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_experiment: end\n");
#endif
}
```

4.13.2.29 window_set_variable()

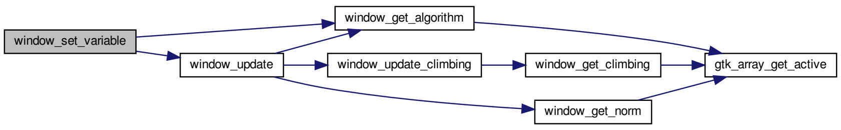
```
void window_set_variable ( )
```

Function to set the variable data in the main window.

Definition at line 1548 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  g_signal_handler_block (window->entry_variable, window->id_variable_label);
  gtk_entry_set_text (window->entry_variable, input->variable[i].name);
  g_signal_handler_unblock (window->entry_variable, window->id_variable_label);
  gtk_spin_button_set_value (window->spin_min, input->variable[i].rangemin);
  gtk_spin_button_set_value (window->spin_max, input->variable[i].rangemax);
  if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_minabs,
                                 input->variable[i].rangeminabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
    }
  if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
    {
      gtk_spin_button_set_value (window->spin_maxabs,
                                 input->variable[i].rangemaxabs);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
    }
  else
    {
      gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
      gtk_toggle_button_set_active
        (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
    }
  gtk_spin_button_set_value (window->spin_precision,
                             input->variable[i].precision);
  gtk_spin_button_set_value (window->spin_steps, (gdouble) input->nsteps);
  if (input->nsteps)
    gtk_spin_button_set_value (window->spin_step, input->variable[i].step);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
           input->variable[i].precision);
#endif
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_spin_button_set_value (window->spin_sweeps,
                                 (gdouble) input->variable[i].nsweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      gtk_spin_button_set_value (window->spin_bits,
                                 (gdouble) input->variable[i].nbits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
               input->variable[i].nbits);
#endif
      break;
    }
  window_update ();
#if DEBUG_INTERFACE
  fprintf (stderr, "window_set_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.30 window_template_experiment()

```
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 1517 of file interface.c.

```
{
  unsigned int i, j;
  char *buffer;
  GFile *file1, *file2;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: start\n");
#endif
  i = (size_t) data;
  j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  file1
    = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
  file2 = g_file_new_for_path (input->directory);
  buffer = g_file_get_relative_path (file2, file1);
  if (input->type == INPUT_TYPE_XML)
    input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
  else
    input->experiment[j].stencil[i] = g_strdup (buffer);
  g_free (buffer);
  g_object_unref (file2);
  g_object_unref (file1);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_template_experiment: end\n");
#endif
}
```

4.13.2.31 window_update()

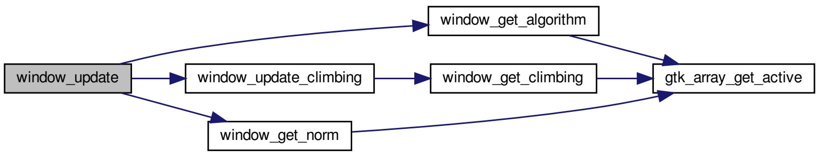
```
void window_update ( )
```

Function to update the main window view.

Definition at line 1124 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: start\n");
#endif
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_evaluator),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
                                   (window->check_evaluator)));
  gtk_widget_hide (GTK_WIDGET (window->label_simulations));
  gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
  gtk_widget_hide (GTK_WIDGET (window->label_iterations));
  gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
  gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
  gtk_widget_hide (GTK_WIDGET (window->label_bests));
  gtk_widget_hide (GTK_WIDGET (window->spin_bests));
  gtk_widget_hide (GTK_WIDGET (window->label_population));
  gtk_widget_hide (GTK_WIDGET (window->spin_population));
  gtk_widget_hide (GTK_WIDGET (window->label_generations));
  gtk_widget_hide (GTK_WIDGET (window->spin_generations));
  gtk_widget_hide (GTK_WIDGET (window->label_mutation));
  gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
  gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
  gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
  gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
  gtk_widget_hide (GTK_WIDGET (window->label_bits));
  gtk_widget_hide (GTK_WIDGET (window->spin_bits));
  gtk_widget_hide (GTK_WIDGET (window->check_climbing));
  gtk_widget_hide (GTK_WIDGET (window->grid_climbing));
  gtk_widget_hide (GTK_WIDGET (window->label_step));
  gtk_widget_hide (GTK_WIDGET (window->spin_step));
  gtk_widget_hide (GTK_WIDGET (window->label_p));
  gtk_widget_hide (GTK_WIDGET (window->spin_p));
  i = gtk_spin_button_get_value_as_int (window->spin_iterations);
  switch (window_get_algorithm ())
    {
    case ALGORITHM_MONTE_CARLO:
      gtk_widget_show (GTK_WIDGET (window->label_simulations));
      gtk_widget_show (GTK_WIDGET (window->spin_simulations));
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      window_update_climbing ();
      break;
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      gtk_widget_show (GTK_WIDGET (window->label_iterations));
      gtk_widget_show (GTK_WIDGET (window->spin_iterations));
      if (i > 1)
        {
          gtk_widget_show (GTK_WIDGET (window->label_tolerance));
          gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
          gtk_widget_show (GTK_WIDGET (window->label_bests));
          gtk_widget_show (GTK_WIDGET (window->spin_bests));
        }
      gtk_widget_show (GTK_WIDGET (window->label_sweeps));
      gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
      gtk_widget_show (GTK_WIDGET (window->check_climbing));
      window_update_climbing ();
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_population));
      gtk_widget_show (GTK_WIDGET (window->spin_population));
      gtk_widget_show (GTK_WIDGET (window->label_generations));
      gtk_widget_show (GTK_WIDGET (window->spin_generations));
      gtk_widget_show (GTK_WIDGET (window->label_mutation));
      gtk_widget_show (GTK_WIDGET (window->spin_mutation));
      gtk_widget_show (GTK_WIDGET (window->label_reproduction));
      gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
      gtk_widget_show (GTK_WIDGET (window->label_adaptation));
      gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
      gtk_widget_show (GTK_WIDGET (window->label_bits));
      gtk_widget_show (GTK_WIDGET (window->spin_bits));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_experiment),
input->nexperiments > 1);
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->button_remove_variable),
input->nvariables > 1);
  for (i = 0; i < input->experiment->ninputs; ++i)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
      g_signal_handler_block
        (window->check_template[i], window->
id_template[i]);
      g_signal_handler_block (window->button_template[i],
window->id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 1);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  if (i > 0)
    {
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
      gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
                                gtk_toggle_button_get_active
                                GTK_TOGGLE_BUTTON (window->check_template
                                                   [i - 1]));
    }
  if (i < MAX_NINPUTS)
    {
      gtk_widget_show (GTK_WIDGET (window->check_template[i]));
      gtk_widget_show (GTK_WIDGET (window->button_template[i]));
      gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
      gtk_widget_set_sensitive
        (GTK_WIDGET (window->button_template[i]),
         gtk_toggle_button_get_active
         GTK_TOGGLE_BUTTON (window->check_template[i]));
      g_signal_handler_block
        (window->check_template[i], window->
id_template[i]);
      g_signal_handler_block (window->button_template[i],
window->id_input[i]);
      gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
                                    (window->check_template[i]), 0);
      g_signal_handler_unblock (window->button_template[i],
                                window->id_input[i]);
      g_signal_handler_unblock (window->check_template[i],
                                window->id_template[i]);
    }
  while (++i < MAX_NINPUTS)
    {
      gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
      gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
    }
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_minabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
  gtk_widget_set_sensitive
    (GTK_WIDGET (window->spin_maxabs),
     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
  if (window_get_norm () == ERROR_NORM_P)
    {
      gtk_widget_show (GTK_WIDGET (window->label_p));
      gtk_widget_show (GTK_WIDGET (window->spin_p));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.32 window_update_climbing()

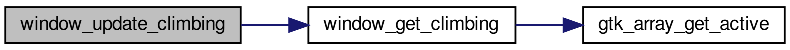
```
void window_update_climbing ( )
```

Function to update hill climbing method widgets view in the main window.

Definition at line 1093 of file interface.c.

```
{
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: start\n");
#endif
  gtk_widget_show (GTK_WIDGET (window->check_climbing));
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->
check_climbing)))
    {
      gtk_widget_show (GTK_WIDGET (window->grid_climbing));
      gtk_widget_show (GTK_WIDGET (window->label_step));
      gtk_widget_show (GTK_WIDGET (window->spin_step));
    }
  switch (window_get_climbing ())
    {
    case CLIMBING_METHOD_COORDINATES:
      gtk_widget_hide (GTK_WIDGET (window->label_estimates));
      gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
      break;
    default:
      gtk_widget_show (GTK_WIDGET (window->label_estimates));
      gtk_widget_show (GTK_WIDGET (window->spin_estimates));
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.33 window_update_variable()

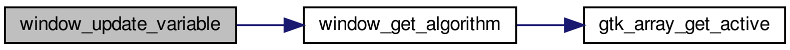
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1824 of file interface.c.

```
{
  int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
  if (i < 0)
    i = 0;
  switch (window_get_algorithm ())
    {
    case ALGORITHM_SWEEP:
    case ALGORITHM_ORTHOGONAL:
      input->variable[i].nsweeps
        = gtk_spin_button_get_value_as_int (window->spin_sweeps);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
               input->variable[i].nsweeps);
#endif
      break;
    case ALGORITHM_GENETIC:
      input->variable[i].nbits
        = gtk_spin_button_get_value_as_int (window->spin_bits);
#if DEBUG_INTERFACE
      fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
               input->variable[i].nbits);
#endif
    }
#if DEBUG_INTERFACE
  fprintf (stderr, "window_update_variable: end\n");
#endif
}
```

Here is the call graph for this function:



4.13.2.34 window_weight_experiment()

```
void window_weight_experiment ( )
```

Function to update the experiment weight in the main window.

Definition at line 1475 of file interface.c.

```
{
  unsigned int i;
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: start\n");
#endif
  i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
  input->experiment[i].weight = gtk_spin_button_get_value (
window->spin_weight);
#if DEBUG_INTERFACE
  fprintf (stderr, "window_weight_experiment: end\n");
#endif
}
```

4.14 interface.h

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef INTERFACE__H
#define INTERFACE__H 1

#define MAX_LENGTH (DEFAULT_PRECISION + 8)

typedef struct
{
  GtkDialog *dialog;
  GtkGrid *grid;
  GtkLabel *label_seed;
  GtkSpinButton *spin_seed;
  GtkLabel *label_threads;
  GtkSpinButton *spin_threads;
  GtkLabel *label_climbing;
  GtkSpinButton *spin_climbing;
} Options;

typedef struct
{
  GtkDialog *dialog;
  GtkLabel *label;
  GtkSpinner *spinner;
  GtkGrid *grid;
} Running;

typedef struct
{
  GtkWindow *window;
  GtkGrid *grid;
  GtkToolbar *bar_buttons;
  GtkToolButton *button_open;
  GtkToolButton *button_save;
  GtkToolButton *button_run;
  GtkToolButton *button_options;
  GtkToolButton *button_help;
  GtkToolButton *button_about;
  GtkToolButton *button_exit;
  GtkGrid *grid_files;
  GtkLabel *label_simulator;
  GtkFileChooserButton *button_simulator;
  GtkCheckButton *check_evaluator;
  GtkFileChooserButton *button_evaluator;
  GtkLabel *label_result;
  GtkEntry *entry_result;
  GtkLabel *label_variables;
  GtkEntry *entry_variables;
  GtkFrame *frame_norm;
  GtkGrid *grid_norm;
  GtkRadioButton *button_norm[NNORMS];
  GtkLabel *label_p;
  GtkSpinButton *spin_p;
  GtkScrolledWindow *scrolled_p;
  GtkFrame *frame_algorithm;
  GtkGrid *grid_algorithm;
  GtkRadioButton *button_algorithm[NALGORITHMS];
  GtkLabel *label_simulations;
  GtkSpinButton *spin_simulations;
  GtkLabel *label_iterations;
  GtkSpinButton *spin_iterations;
  GtkLabel *label_tolerance;
  GtkSpinButton *spin_tolerance;
  GtkLabel *label_bests;
  GtkSpinButton *spin_bests;
  GtkLabel *label_population;
  GtkSpinButton *spin_population;
  GtkLabel *label_generations;
  GtkSpinButton *spin_generations;
  GtkLabel *label_mutation;
  GtkSpinButton *spin_mutation;
  GtkLabel *label_reproduction;
  GtkSpinButton *spin_reproduction;
  GtkLabel *label_adaptation;
  GtkSpinButton *spin_adaptation;
  GtkCheckButton *check_climbing;
  GtkGrid *grid_climbing;
  GtkRadioButton *button_climbing[NCLIMBINGS];
  GtkLabel *label_steps;
  GtkSpinButton *spin_steps;
  GtkLabel *label_estimates;
  GtkSpinButton *spin_estimates;
  GtkLabel *label_relaxation;
  GtkSpinButton *spin_relaxation;
  GtkLabel *label_threshold;
  GtkSpinButton *spin_threshold;
  GtkScrolledWindow *scrolled_threshold;
  GtkFrame *frame_variable;
  GtkGrid *grid_variable;
  GtkComboBoxText *combo_variable;
  GtkButton *button_add_variable;
  GtkButton *button_remove_variable;
  GtkLabel *label_variable;
  GtkEntry *entry_variable;
  GtkLabel *label_min;
  GtkSpinButton *spin_min;
  GtkScrolledWindow *scrolled_min;
  GtkLabel *label_max;
  GtkSpinButton *spin_max;
  GtkScrolledWindow *scrolled_max;
  GtkCheckButton *check_minabs;
  GtkSpinButton *spin_minabs;
  GtkScrolledWindow *scrolled_minabs;
  GtkCheckButton *check_maxabs;
  GtkSpinButton *spin_maxabs;
  GtkScrolledWindow *scrolled_maxabs;
  GtkLabel *label_precision;
  GtkSpinButton *spin_precision;
  GtkLabel *label_sweeps;
  GtkSpinButton *spin_sweeps;
  GtkLabel *label_bits;
  GtkSpinButton *spin_bits;
  GtkLabel *label_step;
  GtkSpinButton *spin_step;
  GtkScrolledWindow *scrolled_step;
  GtkFrame *frame_experiment;
  GtkGrid *grid_experiment;
  GtkComboBoxText *combo_experiment;
  GtkButton *button_add_experiment;
  GtkButton *button_remove_experiment;
  GtkLabel *label_experiment;
  GtkFileChooserButton *button_experiment;
  GtkLabel *label_weight;
  GtkSpinButton *spin_weight;
  GtkCheckButton *check_template[MAX_NINPUTS];
  GtkFileChooserButton *button_template[MAX_NINPUTS];
  GdkPixbuf *logo;
  Experiment *experiment;
  Variable *variable;
  char *application_directory;
  gulong id_experiment;
  gulong id_experiment_name;
  gulong id_variable;
  gulong id_variable_label;
  gulong id_template[MAX_NINPUTS];
  gulong id_input[MAX_NINPUTS];
  unsigned int nexperiments;
  unsigned int nvariables;
} Window;

// Global variables
extern const char *logo[];
extern Options options[1];
extern Running running[1];
extern Window window[1];

// Inline functions
#if GTK_MINOR_VERSION < 10
static inline GtkButton *
gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
{
  GtkButton *button;
  GtkImage *image;
  button = (GtkButton *) gtk_button_new ();
  image = (GtkImage *) gtk_image_new_from_icon_name (name, size);
  gtk_button_set_image (button, GTK_WIDGET (image));
  return button;
}
#endif

// Public functions
unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
void input_save (char *filename);
void options_new ();
void running_new ();
unsigned int window_get_algorithm ();
unsigned int window_get_climbing ();
unsigned int window_get_norm ();
void window_save_climbing ();
int window_save ();
void window_run ();
void window_help ();
void window_update_climbing ();
void window_update ();
void window_set_algorithm ();
void window_set_experiment ();
void window_remove_experiment ();
void window_add_experiment ();
void window_name_experiment ();
void window_weight_experiment ();
void window_inputs_experiment ();
void window_template_experiment (void *data);
void window_set_variable ();
void window_remove_variable ();
void window_add_variable ();
void window_label_variable ();
void window_precision_variable ();
void window_rangemin_variable ();
void window_rangemax_variable ();
void window_rangeminabs_variable ();
void window_rangemaxabs_variable ();
void window_update_variable ();
int window_read (char *filename);
void window_open ();
void window_new (GtkApplication * application);

#endif
```

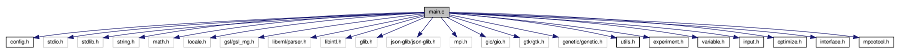
4.15 main.c File Reference

Main source file.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
```

```
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"
```

Include dependency graph for main.c:



Functions

- int **main** (int argn, char ∗∗argc)

4.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file main.c.

4.16 main.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#if HAVE_GTK
#include <gio/gio.h>
#include <gtk/gtk.h>
#endif
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#if HAVE_GTK
#include "interface.h"
#endif
#include "mpcotool.h"

int
main (int argn, char **argc)
{
#if HAVE_GTK
  show_pending = process_pending;
#endif
  return mpcotool (argn, argc);
}
```

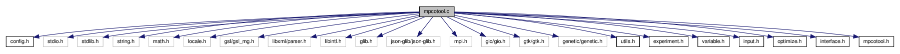
4.17 mpcotool.c File Reference

Main function source file.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <mpi.h>
#include <gio/gio.h>
#include <gtk/gtk.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
```

```
#include "input.h"
#include "optimize.h"
#include "interface.h"
#include "mpcotool.h"
```

Include dependency graph for mpcotool.c:



Macros

- #define DEBUG_MPCOTOOL 0

  Macro to debug main functions.

Functions

- int mpcotool (int argn, char ∗∗argc)

4.17.1 Detailed Description

Main function source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file mpcotool.c.

4.17.2 Function Documentation

4.17.2.1 mpcotool()

```
int mpcotool (
            int argn,
            char ** argc )
```

Main function.

Returns

0 on success, >0 on error.

Parameters

argn	Arguments number.
argc	Arguments pointer.

Definition at line 79 of file mpcotool.c.

```
{
#if HAVE_GTK
  GtkApplication *application;
  char *buffer;
#endif

  // Starting pseudo-random numbers generator
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
#endif
  optimize->rng = gsl_rng_alloc (gsl_rng_taus2);

  // Allowing spaces in the XML data file
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
#endif
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: starting MPI\n");
#endif
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
  printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

  // Resetting result and variables file names
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: resetting result and variables file names\n");
#endif
  input->result = input->variables = NULL;

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_climbing = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;

#if HAVE_GTK

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
                                     G_APPLICATION_FLAGS_NONE);
  g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);

  // Opening the main window
  g_application_run (G_APPLICATION (application), 0, NULL);

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_object_unref (application);
  g_free (window->application_directory);

#else

  // Checking syntax
  if (argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }

  // Getting threads number and pseudo-random numbers generator seed
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
           "generator seed\n");
#endif
  if (argn > 2 && !strcmp (argc[1], "-nthreads"))
    {
      nthreads_climbing = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-seed"))
        {
          optimize->seed = atoi (argc[2]);
          argc += 2;
          argn -= 2;
        }
    }
  else if (argn > 2 && !strcmp (argc[1], "-seed"))
    {
      optimize->seed = atoi (argc[2]);
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-nthreads"))
        {
          nthreads_climbing = nthreads = atoi (argc[2]);
          if (!nthreads)
            {
              printf ("Bad threads number\n");
              return 2;
            }
          argc += 2;
          argn -= 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);
  printf ("seed=%lu\n", optimize->seed);

  // Checking arguments
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: checking arguments\n");
#endif
  if (argn > 4 || argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }
  if (argn > 2)
    input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
  if (argn == 4)
    input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);

  // Making optimization
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: making optimization\n");
#endif
  if (input_open (argc[1]))
    optimize_open ();

  // Freeing memory
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: freeing memory and closing\n");
#endif
  optimize_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (optimize->rng);

  // Closing
  return 0;
}
```

4.18 mpcotool.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <locale.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#if HAVE_GTK
#include <gio/gio.h>
#include <gtk/gtk.h>
#endif
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
#if HAVE_GTK
#include "interface.h"
#endif
#include "mpcotool.h"

#define DEBUG_MPCOTOOL 0


int
mpcotool (int argn
#if HAVE_GTK
          __attribute__ ((unused))
#endif
          ,
          char **argc
#if HAVE_GTK
          __attribute__ ((unused))
```

```c
#endif
  )
{
#if HAVE_GTK
  GtkApplication *application;
  char *buffer;
#endif

  // Starting pseudo-random numbers generator
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: starting pseudo-random numbers generator\n");
#endif
  optimize->rng = gsl_rng_alloc (gsl_rng_taus2);

  // Allowing spaces in the XML data file
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: allowing spaces in the XML data file\n");
#endif
  xmlKeepBlanksDefault (0);

  // Starting MPI
#if HAVE_MPI
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: starting MPI\n");
#endif
  MPI_Init (&argn, &argc);
  MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
  MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
  printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
#else
  ntasks = 1;
#endif

  // Resetting result and variables file names
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: resetting result and variables file names\n");
#endif
  input->result = input->variables = NULL;

  // Getting threads number and pseudo-random numbers generator seed
  nthreads_climbing = nthreads = cores_number ();
  optimize->seed = DEFAULT_RANDOM_SEED;

#if HAVE_GTK

  // Setting local language and international floating point numbers notation
  setlocale (LC_ALL, "");
  setlocale (LC_NUMERIC, "C");
  window->application_directory = g_get_current_dir ();
  buffer = g_build_filename (window->application_directory,
LOCALE_DIR, NULL);
  bindtextdomain (PROGRAM_INTERFACE, buffer);
  bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
  textdomain (PROGRAM_INTERFACE);

  // Initing GTK+
  gtk_disable_setlocale ();
  application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
                                     G_APPLICATION_FLAGS_NONE);
  g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);

  // Opening the main window
  g_application_run (G_APPLICATION (application), 0, NULL);

  // Freeing memory
  input_free ();
  g_free (buffer);
  gtk_widget_destroy (GTK_WIDGET (window->window));
  g_object_unref (application);
  g_free (window->application_directory);

#else

  // Checking syntax
  if (argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }

  // Getting threads number and pseudo-random numbers generator seed
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: getting threads number and pseudo-random numbers "
           "generator seed\n");
#endif
```

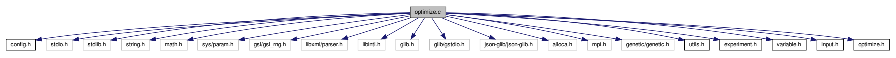
```
  if (argn > 2 && !strcmp (argc[1], "-nthreads"))
    {
      nthreads_climbing = nthreads = atoi (argc[2]);
      if (!nthreads)
        {
          printf ("Bad threads number\n");
          return 2;
        }
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-seed"))
        {
          optimize->seed = atoi (argc[2]);
          argc += 2;
          argn -= 2;
        }
    }
  else if (argn > 2 && !strcmp (argc[1], "-seed"))
    {
      optimize->seed = atoi (argc[2]);
      argc += 2;
      argn -= 2;
      if (argn > 2 && !strcmp (argc[1], "-nthreads"))
        {
          nthreads_climbing = nthreads = atoi (argc[2]);
          if (!nthreads)
            {
              printf ("Bad threads number\n");
              return 2;
            }
          argc += 2;
          argn -= 2;
        }
    }
  printf ("nthreads=%u\n", nthreads);
  printf ("seed=%lu\n", optimize->seed);

  // Checking arguments
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: checking arguments\n");
#endif
  if (argn > 4 || argn < 2)
    {
      printf ("The syntax is:\n"
              "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
              "[variables_file]\n");
      return 1;
    }
  if (argn > 2)
    input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
  if (argn == 4)
    input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);

  // Making optimization
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: making optimization\n");
#endif
  if (input_open (argc[1]))
    optimize_open ();

  // Freeing memory
#if DEBUG_MPCOTOOL
  fprintf (stderr, "mpcotool: freeing memory and closing\n");
#endif
  optimize_free ();

#endif

  // Closing MPI
#if HAVE_MPI
  MPI_Finalize ();
#endif

  // Freeing memory
  gsl_rng_free (optimize->rng);

  // Closing
  return 0;
}
```

4.19 optimize.c File Reference

Source file to define the optimization functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
```

Include dependency graph for optimize.c:



Macros

- #define DEBUG_OPTIMIZE 0

    Macro to debug optimize functions.
- #define RM "rm"

    Macro to define the shell remove command.

Functions

- void optimize_input (unsigned int simulation, char ∗input, GMappedFile ∗stencil)
- double optimize_parse (unsigned int simulation, unsigned int experiment)
- double optimize_norm_euclidian (unsigned int simulation)
- double optimize_norm_maximum (unsigned int simulation)
- double optimize_norm_p (unsigned int simulation)
- double optimize_norm_taxicab (unsigned int simulation)
- void optimize_print ()
- void optimize_save_variables (unsigned int simulation, double error)
- void optimize_best (unsigned int simulation, double value)
- void optimize_sequential ()
- void ∗ optimize_thread (ParallelData ∗data)
- void optimize_merge (unsigned int nsaveds, unsigned int ∗simulation_best, double ∗error_best)
- void optimize_synchronise ()
- void optimize_sweep ()
- void optimize_MonteCarlo ()
- void optimize_orthogonal ()
- void optimize_best_climbing (unsigned int simulation, double value)
- void optimize_climbing_sequential (unsigned int simulation)

- void ∗ optimize_climbing_thread (ParallelData ∗data)
- double optimize_estimate_climbing_random (unsigned int variable, unsigned int estimate)
- double optimize_estimate_climbing_coordinates (unsigned int variable, unsigned int estimate)
- void optimize_step_climbing (unsigned int simulation)
- void optimize_climbing ()
- double optimize_genetic_objective (**Entity** ∗entity)
- void optimize_genetic ()
- void optimize_save_old ()
- void optimize_merge_old ()
- void optimize_refine ()
- void optimize_step ()
- void optimize_iterate ()
- void optimize_free ()
- void optimize_open ()

Variables

- unsigned int nthreads_climbing

  Number of threads for the hill climbing method.
- void(∗ optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(∗ optimize_estimate_climbing)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the climbing.
- double(∗ optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Optimize optimize [1]

  Optimization data.

4.19.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file optimize.c.

4.19.2 Function Documentation

4.19.2.1 optimize_best()

```
void optimize_best (
            unsigned int simulation,
            double value )
```

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 444 of file optimize.c.

```
{
  unsigned int i, j;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->
      error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
              optimize->error_best[i] = optimize->
      error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: end\n");
#endif
}
```

4.19.2.2 optimize_best_climbing()

```
void optimize_best_climbing (
            unsigned int simulation,
            double value )
```

Function to save the best simulation in a hill climbing method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 806 of file optimize.c.

```
{
#if DEBUG_OPTIMIZE
```

```
  fprintf (stderr, "optimize_best_climbing: start\n");
  fprintf (stderr,
           "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG_OPTIMIZE
      fprintf (stderr,
               "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best_climbing: end\n");
#endif
}
```

4.19.2.3 optimize_climbing()

```
void optimize_climbing ( )
```

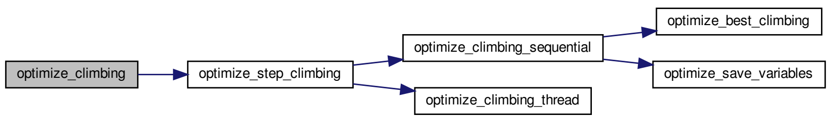
Function to optimize with a hill climbing method.

Definition at line 1034 of file optimize.c.

```
{
  unsigned int i, j, k, b, s, adjust;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    optimize->climbing[i] = 0.;
  b = optimize->simulation_best[0] * optimize->nvariables;
  s = optimize->nsimulations;
  adjust = 1;
  for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      optimize_step_climbing (s);
      k = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      if (k == b)
        {
          if (adjust)
            for (j = 0; j < optimize->nvariables; ++j)
              optimize->step[j] *= 0.5;
          for (j = 0; j < optimize->nvariables; ++j)
            optimize->climbing[j] = 0.;
          adjust = 1;
        }
      else
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
#if DEBUG_OPTIMIZE
              fprintf (stderr,
                       "optimize_climbing: best%u=%.14le old%u=%.14le\n",
                       j, optimize->value[k + j], j, optimize->value[b + j]);
#endif
              optimize->climbing[j]
                = (1. - optimize->relaxation) * optimize->climbing[j]
```

```
                    + optimize->relaxation
                    * (optimize->value[k + j] - optimize->value[b + j]);
#if DEBUG_OPTIMIZE
                  fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
                           j, optimize->climbing[j]);
#endif
                }
              adjust = 0;
            }
        }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.4 optimize_climbing_sequential()

```
void optimize_climbing_sequential (
              unsigned int simulation )
```

Function to estimate the hill climbing sequentially.

Parameters

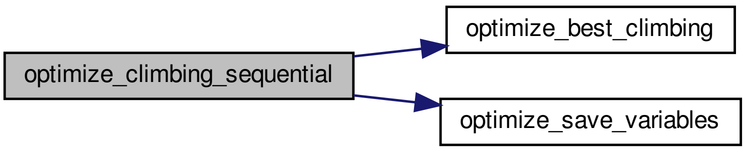
simulation	Simulation number.

Definition at line 834 of file optimize.c.

```
{
  double e;
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: start\n");
  fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
           "nend_climbing=%u\n",
           optimize->nstart_climbing, optimize->
      nend_climbing);
#endif
  for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
    {
      j = simulation + i;
      e = optimize_norm (j);
      optimize_best_climbing (j, e);
      optimize_save_variables (j, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
#endif
```

```
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.5 optimize_climbing_thread()

```
void* optimize_climbing_thread (
            ParallelData * data )
```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

data	Function data.

Definition at line 870 of file optimize.c.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
           thread,
           optimize->thread_climbing[thread],
           optimize->thread_climbing[thread + 1]);
#endif
  for (i = optimize->thread_climbing[thread];
       i < optimize->thread_climbing[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best_climbing (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

4.19.2.6 optimize_estimate_climbing_coordinates()

```
double optimize_estimate_climbing_coordinates (
            unsigned int variable,
            unsigned int estimate )
```

Function to estimate a component of the hill climbing vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 935 of file optimize.c.

```
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
#endif
  x = optimize->climbing[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr,
           "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
#endif
  return x;
}
```

4.19.2.7 optimize_estimate_climbing_random()

```
double optimize_estimate_climbing_random (
            unsigned int variable,
            unsigned int estimate )
```

Function to estimate a component of the hill climbing vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 911 of file optimize.c.

```
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: start\n");
#endif
  x = optimize->climbing[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
step[variable];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_random: end\n");
#endif
  return x;
}
```

4.19.2.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the Optimize struct.

Definition at line 1391 of file optimize.c.

```
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: start\n");
#endif
  for (j = 0; j < optimize->ninputs; ++j)
    {
      for (i = 0; i < optimize->nexperiments; ++i)
        g_mapped_file_unref (optimize->file[j][i]);
      g_free (optimize->file[j]);
    }
  g_free (optimize->error_old);
  g_free (optimize->value_old);
  g_free (optimize->value);
  g_free (optimize->genetic_variable);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: end\n");
#endif
}
```

4.19.2.9 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1131 of file optimize.c.

```
{
  double *best_variable = NULL;
  char *best_genome = NULL;
  double best_objective = 0.;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: start\n");
  fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
           nthreads);
  fprintf (stderr,
           "optimize_genetic: nvariables=%u population=%u generations=%u\n",
           optimize->nvariables, optimize->
nsimulations, optimize->niterations);
  fprintf (stderr,
           "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
           optimize->mutation_ratio, optimize->
reproduction_ratio,
           optimize->adaptation_ratio);
#endif
  genetic_algorithm_default (optimize->nvariables,
                             optimize->genetic_variable,
                             optimize->nsimulations,
                             optimize->niterations,
                             optimize->mutation_ratio,
                             optimize->reproduction_ratio,
                             optimize->adaptation_ratio,
                             optimize->seed,
                             optimize->threshold,
                             &optimize_genetic_objective,
                             &best_genome, &best_variable, &best_objective);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: the best\n");
#endif
  optimize->error_old = (double *) g_malloc (sizeof (double));
  optimize->value_old
    = (double *) g_malloc (optimize->nvariables * sizeof (double));
  optimize->error_old[0] = best_objective;
  memcpy (optimize->value_old, best_variable,
          optimize->nvariables * sizeof (double));
  g_free (best_genome);
  g_free (best_variable);
  optimize_print ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: end\n");
#endif
}
```

4.19.2.10 optimize_genetic_objective()

```
double optimize_genetic_objective (
               Entity * entity )
```

Function to calculate the objective function of an entity.

Returns

objective function value.

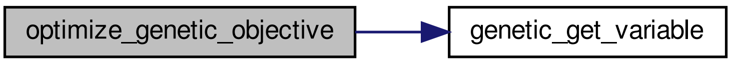
Parameters

entity	entity data.

Definition at line 1098 of file optimize.c.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: start\n");
#endif
  for (j = 0; j < optimize->nvariables; ++j)
    {
      optimize->value[entity->id * optimize->nvariables + j]
        = genetic_get_variable (entity, optimize->genetic_variable + j);
    }
  objective = optimize_norm (entity->id);
  g_mutex_lock (mutex);
  for (j = 0; j < optimize->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
      fprintf (optimize->file_variables, buffer,
               genetic_get_variable (entity, optimize->genetic_variable + j));
    }
  fprintf (optimize->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: end\n");
#endif
  return objective;
}
```

Here is the call graph for this function:



4.19.2.11 optimize_input()

```
void optimize_input (
             unsigned int simulation,
             char * input,
             GMappedFile * stencil )
```

Function to write the simulation input file.

Parameters

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

Definition at line 93 of file optimize.c.

```
{
  char buffer[32], value[32];
  GRegex *regex;
  FILE *file;
  char *buffer2, *buffer3 = NULL, *content;
  gsize length;
  unsigned int i;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: start\n");
#endif

  // Checking the file
  if (!stencil)
    goto optimize_input_end;

  // Opening stencil
  content = g_mapped_file_get_contents (stencil);
  length = g_mapped_file_get_length (stencil);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
#endif
  file = g_fopen (input, "w");

  // Parsing stencil
  for (i = 0; i < optimize->nvariables; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      snprintf (value, 32, format[optimize->precision[i]],
                optimize->value[simulation * optimize->
nvariables + i]);

#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         (GRegexMatchFlags) 0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

optimize_input_end:
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: end\n");
#endif
  return;
}
```

4.19.2.12 optimize_iterate()

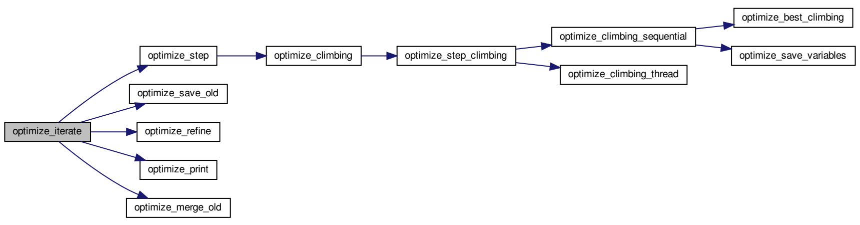
```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1361 of file optimize.c.

```
{
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: start\n");
#endif
  optimize->error_old = (double *) g_malloc (optimize->
nbest * sizeof (double));
  optimize->value_old =
    (double *) g_malloc (optimize->nbest * optimize->
nvariables *
                         sizeof (double));
  optimize_step ();
  optimize_save_old ();
  optimize_refine ();
  optimize_print ();
  for (i = 1; i < optimize->niterations && !optimize->
stop; ++i)
    {
      optimize_step ();
      optimize_merge_old ();
      optimize_refine ();
      optimize_print ();
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.13 optimize_merge()

```
void optimize_merge (
            unsigned int nsaveds,
            unsigned int * simulation_best,
            double * error_best )
```

Function to merge the 2 optimization 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 557 of file optimize.c.

```
{
  unsigned int i, j, k, s[optimize->nbest];
  double e[optimize->nbest];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == optimize->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: end\n");
#endif
}
```

4.19.2.14 optimize_merge_old()

```
void optimize_merge_old ( )
```

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

Definition at line 1212 of file optimize.c.

```
{
  unsigned int i, j, k;
  double v[optimize->nbest * optimize->nvariables], e[optimize->nbest],
    *enew, *eold;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: start\n");
#endif
  enew = optimize->error_best;
  eold = optimize->error_old;
  i = j = k = 0;
  do
    {
      if (*enew < *eold)
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value
                  + optimize->simulation_best[i] * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *enew;
          ++k;
          ++enew;
          ++i;
        }
      else
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value_old + j * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *eold;
          ++k;
          ++eold;
          ++j;
        }
    }
  while (k < optimize->nbest);
  memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
  memcpy (optimize->error_old, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: end\n");
#endif
}
```

4.19.2.15 optimize_MonteCarlo()

```
void optimize_MonteCarlo ( )
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file optimize.c.

```
{
  unsigned int i, j;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    for (j = 0; j < optimize->nvariables; ++j)
      optimize->value[i * optimize->nvariables + j]
        = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
        * (optimize->rangemax[j] - optimize->rangemin[j]);
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: end\n");
#endif
}
```

4.19.2.16 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
               unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

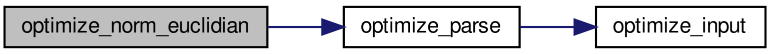
Parameters

simulation	simulation number.

Definition at line 292 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.19.2.17 optimize_norm_maximum()

```
double optimize_norm_maximum (
            unsigned int simulation )
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

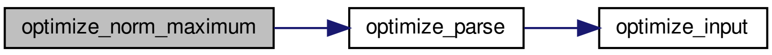
Parameters

simulation	simulation number.

Definition at line 319 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e = fmax (e, ei);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_maximum: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.19.2.18 optimize_norm_p()

```
double optimize_norm_p (
            unsigned int simulation )
```

Function to calculate the P error norm.

Returns

P error norm.

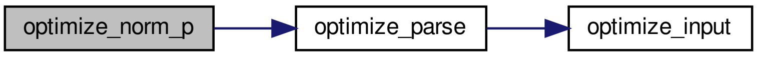
Parameters

simulation	simulation number.

Definition at line 345 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_p: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e += pow (ei, optimize->p);
    }
  e = pow (e, 1. / optimize->p);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_p: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.19.2.19 optimize_norm_taxicab()

```
double optimize_norm_taxicab (
              unsigned int simulation )
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

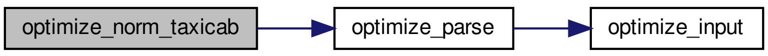
Parameters

simulation	simulation number.

Definition at line 372 of file optimize.c.

```
{
  double e;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.19.2.20 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

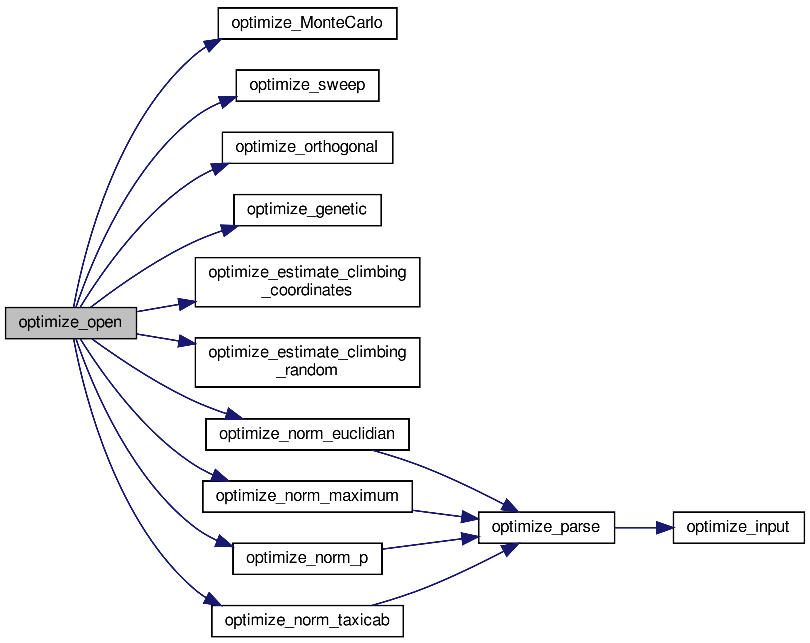
Definition at line 1416 of file optimize.c.

```
{
  GTimeZone *tz;
  GDateTime *t0, *t;
  unsigned int i, j;

#if DEBUG_OPTIMIZE
  char *buffer;
  fprintf (stderr, "optimize_open: start\n");
#endif

  // Getting initial time
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial time\n");
#endif
  tz = g_time_zone_new_utc ();
  t0 = g_date_time_new_now (tz);

  // Obtaining and initing the pseudo-random numbers generator seed
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial seed\n");
#endif
  if (optimize->seed == DEFAULT_RANDOM_SEED)
    optimize->seed = input->seed;
  gsl_rng_set (optimize->rng, optimize->seed);

  // Replacing the working directory
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: replacing the working directory\n");
#endif
  g_chdir (input->directory);

  // Getting results file names
  optimize->result = input->result;
  optimize->variables = input->variables;
```

```c

  // Obtaining the simulator file
  optimize->simulator = input->simulator;

  // Obtaining the evaluator file
  optimize->evaluator = input->evaluator;

  // Reading the algorithm
  optimize->algorithm = input->algorithm;
  switch (optimize->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      optimize_algorithm = optimize_MonteCarlo;
      break;
    case ALGORITHM_SWEEP:
      optimize_algorithm = optimize_sweep;
      break;
    case ALGORITHM_ORTHOGONAL:
      optimize_algorithm = optimize_orthogonal;
      break;
    default:
      optimize_algorithm = optimize_genetic;
      optimize->mutation_ratio = input->mutation_ratio;
      optimize->reproduction_ratio = input->reproduction_ratio;
      optimize->adaptation_ratio = input->adaptation_ratio;
    }
  optimize->nvariables = input->nvariables;
  optimize->nsimulations = input->nsimulations;
  optimize->niterations = input->niterations;
  optimize->nbest = input->nbest;
  optimize->tolerance = input->tolerance;
  optimize->nsteps = input->nsteps;
  optimize->nestimates = 0;
  optimize->threshold = input->threshold;
  optimize->stop = 0;
  if (input->nsteps)
    {
      optimize->relaxation = input->relaxation;
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          optimize->nestimates = 2 * optimize->nvariables;
          optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
          break;
        default:
          optimize->nestimates = input->nestimates;
          optimize_estimate_climbing = optimize_estimate_climbing_random;
        }
    }

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
#endif
  optimize->simulation_best
    = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
  optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));

  // Reading the experimental data
#if DEBUG_OPTIMIZE
  buffer = g_get_current_dir ();
  fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
  g_free (buffer);
#endif
  optimize->nexperiments = input->nexperiments;
  optimize->ninputs = input->experiment->ninputs;
  optimize->experiment
    = (char **) alloca (input->nexperiments * sizeof (char *));
  optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
  for (i = 0; i < input->experiment->ninputs; ++i)
    optimize->file[i] = (GMappedFile **)
      g_malloc (input->nexperiments * sizeof (GMappedFile *));
  for (i = 0; i < input->nexperiments; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u\n", i);
#endif
      optimize->experiment[i] = input->experiment[i].name;
      optimize->weight[i] = input->experiment[i].
weight;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
               optimize->experiment[i], optimize->weight[i]);
#endif
      for (j = 0; j < input->experiment->ninputs; ++j)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
#endif
          optimize->file[j][i]
            = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
        }
    }

  // Reading the variables data
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: reading variables\n");
#endif
  optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
  j = input->nvariables * sizeof (double);
  optimize->rangemin = (double *) alloca (j);
  optimize->rangeminabs = (double *) alloca (j);
  optimize->rangemax = (double *) alloca (j);
  optimize->rangemaxabs = (double *) alloca (j);
  optimize->step = (double *) alloca (j);
  j = input->nvariables * sizeof (unsigned int);
  optimize->precision = (unsigned int *) alloca (j);
  optimize->nsweeps = (unsigned int *) alloca (j);
  optimize->nbits = (unsigned int *) alloca (j);
  for (i = 0; i < input->nvariables; ++i)
    {
      optimize->label[i] = input->variable[i].name;
      optimize->rangemin[i] = input->variable[i].rangemin;
      optimize->rangeminabs[i] = input->variable[i].rangeminabs;
      optimize->rangemax[i] = input->variable[i].rangemax;
      optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
      optimize->precision[i] = input->variable[i].precision;
      optimize->step[i] = input->variable[i].step;
      optimize->nsweeps[i] = input->variable[i].nsweeps;
      optimize->nbits[i] = input->variable[i].nbits;
    }
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {
      optimize->nsimulations = 1;
      for (i = 0; i < input->nvariables; ++i)
        {
          optimize->nsimulations *= optimize->nsweeps[i];
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
                   optimize->nsweeps[i], optimize->nsimulations);
#endif
        }
    }
  if (optimize->nsteps)
    optimize->climbing
      = (double *) alloca (optimize->nvariables * sizeof (double));

  // Setting error norm
  switch (input->norm)
    {
    case ERROR_NORM_EUCLIDIAN:
      optimize_norm = optimize_norm_euclidian;
      break;
    case ERROR_NORM_MAXIMUM:
      optimize_norm = optimize_norm_maximum;
      break;
    case ERROR_NORM_P:
      optimize_norm = optimize_norm_p;
      optimize->p = input->p;
      break;
    default:
      optimize_norm = optimize_norm_taxicab;
    }

  // Allocating values
#if DEBUG_OPTIMIZE
```

```
  fprintf (stderr, "optimize_open: allocating variables\n");
  fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
           optimize->nvariables, optimize->algorithm);
#endif
  optimize->genetic_variable = NULL;
  if (optimize->algorithm == ALGORITHM_GENETIC)
    {
      optimize->genetic_variable = (GeneticVariable *)
        g_malloc (optimize->nvariables * sizeof (
GeneticVariable));
      for (i = 0; i < optimize->nvariables; ++i)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
                   i, optimize->rangemin[i], optimize->
rangemax[i],
                   optimize->nbits[i]);
#endif
          optimize->genetic_variable[i].minimum =
optimize->rangemin[i];
          optimize->genetic_variable[i].maximum =
optimize->rangemax[i];
          optimize->genetic_variable[i].nbits = optimize->
nbits[i];
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
           optimize->nvariables, optimize->
nsimulations);
#endif
  optimize->value = (double *)
    g_malloc ((optimize->nsimulations
               + optimize->nestimates * optimize->
nsteps)
              * optimize->nvariables * sizeof (double));

  // Calculating simulations to perform for each task
#if HAVE_MPI
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
           optimize->mpi_rank, ntasks);
#endif
  optimize->nstart = optimize->mpi_rank * optimize->
nsimulations / ntasks;
  optimize->nend = (1 + optimize->mpi_rank) *
optimize->nsimulations / ntasks;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing
        = optimize->mpi_rank * optimize->nestimates /
ntasks;
      optimize->nend_climbing
        = (1 + optimize->mpi_rank) * optimize->
nestimates / ntasks;
    }
#else
  optimize->nstart = 0;
  optimize->nend = optimize->nsimulations;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing = 0;
      optimize->nend_climbing = optimize->
nestimates;
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
nstart,
           optimize->nend);
#endif

  // Calculating simulations to perform for each thread
  optimize->thread
    = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
  for (i = 0; i <= nthreads; ++i)
    {
      optimize->thread[i] = optimize->nstart
        + i * (optimize->nend - optimize->nstart) / nthreads;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
               optimize->thread[i]);
#endif
    }
  if (optimize->nsteps)
    optimize->thread_climbing = (unsigned int *)
      alloca ((1 + nthreads_climbing) * sizeof (unsigned int));

  // Opening result files
  optimize->file_result = g_fopen (optimize->result, "w");
  optimize->file_variables = g_fopen (optimize->variables, "w");

  // Performing the algorithm
  switch (optimize->algorithm)
    {
      // Genetic algorithm
    case ALGORITHM_GENETIC:
      optimize_genetic ();
      break;

      // Iterative algorithm
    default:
      optimize_iterate ();
    }

  // Getting calculation time
  t = g_date_time_new_now (tz);
  optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
  g_date_time_unref (t);
  g_date_time_unref (t0);
  g_time_zone_unref (tz);
  printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
  fprintf (optimize->file_result, "%s = %.6lg s\n",
           _("Calculation time"), optimize->calculation_time);

  // Closing result files
  fclose (optimize->file_variables);
  fclose (optimize->file_result);

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.21 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file optimize.c.

```
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += (l + gsl_rng_uniform (optimize->rng))
              * (optimize->rangemax[j] - optimize->
rangemin[j])
              / optimize->nsweeps[j];
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: end\n");
#endif
}
```

4.19.2.22 optimize_parse()

```
double optimize_parse (
            unsigned int simulation,
            unsigned int experiment )
```

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

Returns

Objective function value.

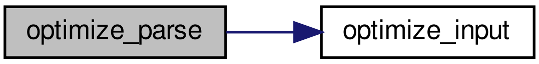
Parameters

simulation	Simulation number.
experiment	Experiment number.

Definition at line 182 of file optimize.c.

```
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: start\n");
  fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
           simulation, experiment);
#endif

  // Opening input files
  for (i = 0; i < optimize->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (optimize->simulator);
  buffer3 = g_path_get_basename (optimize->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4],
            input[5], input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (optimize->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (optimize->evaluator);
      buffer3 = g_path_get_basename (optimize->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, optimize->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: %s\n", buffer);
      fprintf (stderr, "optimize_parse: result=%s\n", result);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: output=%s\n", output);
#endif
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG_OPTIMIZE
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

  // Processing pending events
  if (show_pending)
    show_pending ();

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}
```

Here is the call graph for this function:



4.19.2.23 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file optimize.c.

```
{
  unsigned int i;
  char buffer[512];
#if HAVE_MPI
  if (optimize->mpi_rank)
    return;
#endif
  printf ("%s\n", _("Best result"));
  fprintf (optimize->file_result, "%s\n", _("Best result"));
  printf ("error = %.15le\n", optimize->error_old[0]);
  fprintf (optimize->file_result, "error = %.15le\n",
optimize->error_old[0]);
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 512, "%s = %s\n",
                optimize->label[i], format[optimize->
precision[i]]);
      printf (buffer, optimize->value_old[i]);
      fprintf (optimize->file_result, buffer, optimize->
value_old[i]);
    }
  fflush (optimize->file_result);
}
```

4.19.2.24 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1260 of file optimize.c.

```
{
  unsigned int i, j;
  double d;
#if HAVE_MPI
  MPI_Status mpi_stat;
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: start\n");
#endif
#if HAVE_MPI
  if (!optimize->mpi_rank)
    {
#endif
      for (j = 0; j < optimize->nvariables; ++j)
        {
          optimize->rangemin[j] = optimize->rangemax[j]
            = optimize->value_old[j];
        }
      for (i = 0; ++i < optimize->nbest;)
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
              optimize->rangemin[j]
                = fmin (optimize->rangemin[j],
                        optimize->value_old[i * optimize->
nvariables + j]);
              optimize->rangemax[j]
                = fmax (optimize->rangemax[j],
                        optimize->value_old[i * optimize->
nvariables + j]);
            }
        }
      for (j = 0; j < optimize->nvariables; ++j)
        {
          d = optimize->tolerance
            * (optimize->rangemax[j] - optimize->
rangemin[j]);
          switch (optimize->algorithm)
            {
            case ALGORITHM_MONTE_CARLO:
              d *= 0.5;
              break;
            default:
              if (optimize->nsweeps[j] > 1)
                d /= optimize->nsweeps[j] - 1;
              else
                d = 0.;
            }
          optimize->rangemin[j] -= d;
          optimize->rangemin[j]
            = fmax (optimize->rangemin[j], optimize->
rangeminabs[j]);
          optimize->rangemax[j] += d;
          optimize->rangemax[j]
            = fmin (optimize->rangemax[j], optimize->
rangemaxabs[j]);
          printf ("%s min=%lg max=%lg\n", optimize->label[j],
                  optimize->rangemin[j], optimize->
rangemax[j]);
          fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
                   optimize->label[j], optimize->rangemin[j],
                   optimize->rangemax[j]);
        }
#if HAVE_MPI
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Send (optimize->rangemin, optimize->
nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
          MPI_Send (optimize->rangemax, optimize->
nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
        }
    }
  else
    {
      MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
1,
                MPI_COMM_WORLD, &mpi_stat);
      MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
1,
                MPI_COMM_WORLD, &mpi_stat);
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: end\n");
#endif
}
```

4.19.2.25 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1180 of file optimize.c.

```
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_old: start\n");
  fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
#endif
  memcpy (optimize->error_old, optimize->error_best,
          optimize->nbest * sizeof (double));
  for (i = 0; i < optimize->nbest; ++i)
    {
      j = optimize->simulation_best[i];
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
#endif
      memcpy (optimize->value_old + i * optimize->
nvariables,
              optimize->value + j * optimize->nvariables,
              optimize->nvariables * sizeof (double));
    }
#if DEBUG_OPTIMIZE
  for (i = 0; i < optimize->nvariables; ++i)
    fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
             i, optimize->value_old[i]);
  fprintf (stderr, "optimize_save_old: end\n");
#endif
}
```

4.19.2.26 optimize_save_variables()

```
void optimize_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 419 of file optimize.c.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->
nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
  fflush (optimize->file_variables);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}
```

4.19.2.27 optimize_sequential()

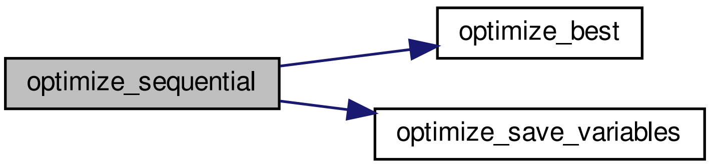
```
void optimize_sequential ( )
```

Function to optimize sequentially.

Definition at line 485 of file optimize.c.

```
{
  unsigned int i;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: start\n");
  fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
           optimize->nstart, optimize->nend);
#endif
  for (i = optimize->nstart; i < optimize->nend; ++i)
    {
      e = optimize_norm (i);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.28 optimize_step()

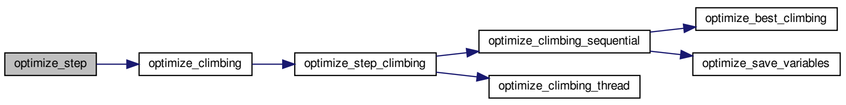
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1344 of file optimize.c.

```
{
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: start\n");
#endif
  optimize_algorithm ();
  if (optimize->nsteps)
    optimize_climbing ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.29 optimize_step_climbing()

```
void optimize_step_climbing (
            unsigned int simulation )
```

Function to do a step of the hill climbing method.

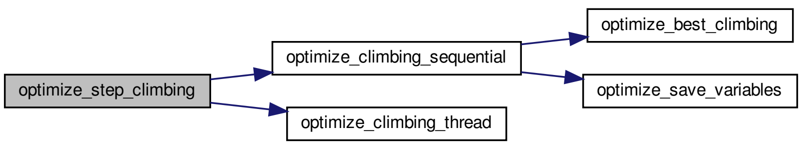
Parameters

simulation	Simulation number.

Definition at line 965 of file optimize.c.

```
{
  GThread *thread[nthreads_climbing];
  ParallelData data[nthreads_climbing];
  unsigned int i, j, k, b;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: start\n");
#endif
  for (i = 0; i < optimize->nestimates; ++i)
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->
nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_climbing (j, i)
;
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_climbing == 1)
    optimize_climbing_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_climbing; ++i)
        {
          optimize->thread_climbing[i]
            = simulation + optimize->nstart_climbing
            + i * (optimize->nend_climbing - optimize->
nstart_climbing)
            / nthreads_climbing;
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: i=%u thread_climbing=%u\n",
                   i, optimize->thread_climbing[i]);
#endif
        }
      for (i = 0; i < nthreads_climbing; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
        }
      for (i = 0; i < nthreads_climbing; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.19.2.30 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file optimize.c.

```
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += l * (optimize->rangemax[j] - optimize->
rangemin[j])
              / (optimize->nsweeps[j] - 1);
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: end\n");
#endif
}
```

4.19.2.31 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file optimize.c.

```
{
  unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
  double error_best[optimize->nbest];
  MPI_Status mpi_stat;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: start\n");
#endif
  if (optimize->mpi_rank == 0)
    {
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          optimize_merge (nsaveds, simulation_best, error_best);
          MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
          if (stop)
            optimize->stop = 1;
        }
      for (i = 1; i < ntasks; ++i)
        MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
    }
  else
    {
      MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
      MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
      if (stop)
        optimize->stop = 1;
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: end\n");
#endif
}
```

4.19.2.32 optimize_thread()

```
void* optimize_thread (
            ParallelData * data )
```

Function to optimize on a thread.

Returns

NULL.

Parameters

data	Function data.

Definition at line 519 of file optimize.c.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

4.20 optimize.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
#include <alloca.h>
#endif
#if HAVE_MPI
#include <mpi.h>
#endif
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"

#define DEBUG_OPTIMIZE 0


#ifdef G_OS_WIN32
#define RM "del"
#else
#define RM "rm"
#endif
```

```

unsigned int nthreads_climbing;
void (*optimize_algorithm) ();
double (*optimize_estimate_climbing) (unsigned int variable,
                                      unsigned int estimate);
double (*optimize_norm) (unsigned int simulation);
Optimize optimize[1];

void
optimize_input (unsigned int simulation,
                char *input,
                GMappedFile * stencil)
{
  char buffer[32], value[32];
  GRegex *regex;
  FILE *file;
  char *buffer2, *buffer3 = NULL, *content;
  gsize length;
  unsigned int i;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: start\n");
#endif

  // Checking the file
  if (!stencil)
    goto optimize_input_end;

  // Opening stencil
  content = g_mapped_file_get_contents (stencil);
  length = g_mapped_file_get_length (stencil);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
#endif
  file = g_fopen (input, "w");

  // Parsing stencil
  for (i = 0; i < optimize->nvariables; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      snprintf (value, 32, format[optimize->precision[i]],
                optimize->value[simulation * optimize->nvariables + i]);

#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         (GRegexMatchFlags) 0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

optimize_input_end:
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: end\n");
```

```
#endif
  return;
}

double
optimize_parse (unsigned int simulation,
                unsigned int experiment)
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: start\n");
  fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
           simulation, experiment);
#endif

  // Opening input files
  for (i = 0; i < optimize->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (optimize->simulator);
  buffer3 = g_path_get_basename (optimize->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4],
            input[5], input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (optimize->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (optimize->evaluator);
      buffer3 = g_path_get_basename (optimize->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, optimize->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: %s\n", buffer);
      fprintf (stderr, "optimize_parse: result=%s\n", result);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: output=%s\n", output);
#endif
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG_OPTIMIZE
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

  // Processing pending events
  if (show_pending)
    show_pending ();

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}

double
optimize_norm_euclidian (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}

double
optimize_norm_maximum (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e = fmax (e, ei);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_maximum: end\n");
#endif
  return e;
}

double
optimize_norm_p (unsigned int simulation)
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_p: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e += pow (ei, optimize->p);
    }
  e = pow (e, 1. / optimize->p);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_p: end\n");
#endif
  return e;
}

```

```c
double
optimize_norm_taxicab (unsigned int simulation)
{
  double e;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}

void
optimize_print ()
{
  unsigned int i;
  char buffer[512];
#if HAVE_MPI
  if (optimize->mpi_rank)
    return;
#endif
  printf ("%s\n", _("Best result"));
  fprintf (optimize->file_result, "%s\n", _("Best result"));
  printf ("error = %.15le\n", optimize->error_old[0]);
  fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 512, "%s = %s\n",
                optimize->label[i], format[optimize->precision[i]]);
      printf (buffer, optimize->value_old[i]);
      fprintf (optimize->file_result, buffer, optimize->value_old[i]);
    }
  fflush (optimize->file_result);
}

void
optimize_save_variables (unsigned int simulation,
                         double error)
{
  unsigned int i;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
  fflush (optimize->file_variables);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}

void
optimize_best (unsigned int simulation,
               double value)
{
  unsigned int i, j;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->simulation_best[i - 1];
              optimize->error_best[i] = optimize->error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: end\n");
#endif
}

void
optimize_sequential ()
{
  unsigned int i;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: start\n");
  fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
           optimize->nstart, optimize->nend);
#endif
  for (i = optimize->nstart; i < optimize->nend; ++i)
    {
      e = optimize_norm (i);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: end\n");
#endif
}

void *
optimize_thread (ParallelData * data)
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

void
optimize_merge (unsigned int nsaveds,
                unsigned int *simulation_best,
                double *error_best)
{
  unsigned int i, j, k, s[optimize->nbest];
  double e[optimize->nbest];
```

```
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == optimize->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: end\n");
#endif
}

#if HAVE_MPI
void
optimize_synchronise ()
{
  unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
  double error_best[optimize->nbest];
  MPI_Status mpi_stat;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: start\n");
#endif
  if (optimize->mpi_rank == 0)
    {
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          optimize_merge (nsaveds, simulation_best, error_best);
          MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
          if (stop)
            optimize->stop = 1;
        }
      for (i = 1; i < ntasks; ++i)
        MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
    }
  else
    {
      MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
      MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
      if (stop)
        optimize->stop = 1;
    }
```

```
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: end\n");
#endif
}
#endif

void
optimize_sweep ()
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += l * (optimize->rangemax[j] - optimize->rangemin[j])
              / (optimize->nsweeps[j] - 1);
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: end\n");
#endif
}

void
optimize_MonteCarlo ()
{
  unsigned int i, j;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    for (j = 0; j < optimize->nvariables; ++j)
      optimize->value[i * optimize->nvariables + j]
        = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
        * (optimize->rangemax[j] - optimize->rangemin[j]);
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: end\n");
```

```
#endif
}

void
optimize_orthogonal ()
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += (l + gsl_rng_uniform (optimize->rng))
              * (optimize->rangemax[j] - optimize->rangemin[j])
              / optimize->nsweeps[j];
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: end\n");
#endif
}

void
optimize_best_climbing (unsigned int simulation,
                        double value)
{
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best_climbing: start\n");
  fprintf (stderr,
           "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG_OPTIMIZE
      fprintf (stderr,
               "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best_climbing: end\n");
#endif
}

void
optimize_climbing_sequential (unsigned int simulation)
{
  double e;
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: start\n");
  fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
           "nend_climbing=%u\n",
           optimize->nstart_climbing, optimize->nend_climbing);
#endif
```

```
  for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
    {
      j = simulation + i;
      e = optimize_norm (j);
      optimize_best_climbing (j, e);
      optimize_save_variables (j, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: end\n");
#endif
}

void *
optimize_climbing_thread (ParallelData * data)
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
           thread,
           optimize->thread_climbing[thread],
           optimize->thread_climbing[thread + 1]);
#endif
  for (i = optimize->thread_climbing[thread];
       i < optimize->thread_climbing[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best_climbing (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}

double
optimize_estimate_climbing_random (unsigned int variable,
                                   unsigned int estimate
                                   __attribute__ ((unused)))
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: start\n");
#endif
  x = optimize->climbing[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_random: end\n");
#endif
  return x;
}

double
optimize_estimate_climbing_coordinates (unsigned int variable,
                                        unsigned int estimate)
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
#endif
  x = optimize->climbing[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
```

```
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr,
           "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
#endif
  return x;
}

void
optimize_step_climbing (unsigned int simulation)
{
  GThread *thread[nthreads_climbing];
  ParallelData data[nthreads_climbing];
  unsigned int i, j, k, b;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: start\n");
#endif
  for (i = 0; i < optimize->nestimates; ++i)
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_climbing (j, i);
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_climbing == 1)
    optimize_climbing_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_climbing; ++i)
        {
          optimize->thread_climbing[i]
            = simulation + optimize->nstart_climbing
            + i * (optimize->nend_climbing - optimize->
nstart_climbing)
            / nthreads_climbing;
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: i=%u thread_climbing=%u\n",
                   i, optimize->thread_climbing[i]);
#endif
        }
      for (i = 0; i < nthreads_climbing; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
        }
      for (i = 0; i < nthreads_climbing; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: end\n");
#endif
}

void
optimize_climbing ()
{
  unsigned int i, j, k, b, s, adjust;
#if DEBUG_OPTIMIZE
```

```
  fprintf (stderr, "optimize_climbing: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    optimize->climbing[i] = 0.;
  b = optimize->simulation_best[0] * optimize->nvariables;
  s = optimize->nsimulations;
  adjust = 1;
  for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      optimize_step_climbing (s);
      k = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      if (k == b)
        {
          if (adjust)
            for (j = 0; j < optimize->nvariables; ++j)
              optimize->step[j] *= 0.5;
          for (j = 0; j < optimize->nvariables; ++j)
            optimize->climbing[j] = 0.;
          adjust = 1;
        }
      else
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
#if DEBUG_OPTIMIZE
              fprintf (stderr,
                       "optimize_climbing: best%u=%.14le old%u=%.14le\n",
                       j, optimize->value[k + j], j, optimize->value[b + j]);
#endif
              optimize->climbing[j]
                = (1. - optimize->relaxation) * optimize->climbing[j]
                + optimize->relaxation
                * (optimize->value[k + j] - optimize->value[b + j]);
#if DEBUG_OPTIMIZE
              fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
                       j, optimize->climbing[j]);
#endif
            }
          adjust = 0;
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing: end\n");
#endif
}

double
optimize_genetic_objective (Entity * entity)
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: start\n");
#endif
  for (j = 0; j < optimize->nvariables; ++j)
    {
      optimize->value[entity->id * optimize->nvariables + j]
        = genetic_get_variable (entity, optimize->genetic_variable + j);
    }
  objective = optimize_norm (entity->id);
  g_mutex_lock (mutex);
  for (j = 0; j < optimize->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
      fprintf (optimize->file_variables, buffer,
               genetic_get_variable (entity, optimize->genetic_variable + j));
    }
  fprintf (optimize->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: end\n");
#endif
  return objective;
}

void
optimize_genetic ()
{
  double *best_variable = NULL;
  char *best_genome = NULL;
  double best_objective = 0.;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: start\n");
  fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
           nthreads);
  fprintf (stderr,
           "optimize_genetic: nvariables=%u population=%u generations=%u\n",
           optimize->nvariables, optimize->nsimulations, optimize->
niterations);
  fprintf (stderr,
           "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
           optimize->mutation_ratio, optimize->reproduction_ratio,
           optimize->adaptation_ratio);
#endif
  genetic_algorithm_default (optimize->nvariables,
                             optimize->genetic_variable,
                             optimize->nsimulations,
                             optimize->niterations,
                             optimize->mutation_ratio,
                             optimize->reproduction_ratio,
                             optimize->adaptation_ratio,
                             optimize->seed,
                             optimize->threshold,
                             &optimize_genetic_objective,
                             &best_genome, &best_variable, &best_objective);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: the best\n");
#endif
  optimize->error_old = (double *) g_malloc (sizeof (double));
  optimize->value_old
    = (double *) g_malloc (optimize->nvariables * sizeof (double));
  optimize->error_old[0] = best_objective;
  memcpy (optimize->value_old, best_variable,
          optimize->nvariables * sizeof (double));
  g_free (best_genome);
  g_free (best_variable);
  optimize_print ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: end\n");
#endif
}

void
optimize_save_old ()
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_old: start\n");
  fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
#endif
  memcpy (optimize->error_old, optimize->error_best,
          optimize->nbest * sizeof (double));
  for (i = 0; i < optimize->nbest; ++i)
    {
      j = optimize->simulation_best[i];
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
#endif
      memcpy (optimize->value_old + i * optimize->nvariables,
              optimize->value + j * optimize->nvariables,
              optimize->nvariables * sizeof (double));
    }
#if DEBUG_OPTIMIZE
  for (i = 0; i < optimize->nvariables; ++i)
    fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
             i, optimize->value_old[i]);
  fprintf (stderr, "optimize_save_old: end\n");
#endif
}

void
optimize_merge_old ()
{
  unsigned int i, j, k;
  double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
    *enew, *eold;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: start\n");
#endif
  enew = optimize->error_best;
  eold = optimize->error_old;
  i = j = k = 0;
  do
    {
      if (*enew < *eold)
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value
                  + optimize->simulation_best[i] * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *enew;
          ++k;
          ++enew;
          ++i;
        }
      else
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value_old + j * optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *eold;
          ++k;
          ++eold;
          ++j;
        }
    }
  while (k < optimize->nbest);
  memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
  memcpy (optimize->error_old, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: end\n");
#endif
}

void
optimize_refine ()
{
  unsigned int i, j;
  double d;
#if HAVE_MPI
  MPI_Status mpi_stat;
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: start\n");
#endif
#if HAVE_MPI
  if (!optimize->mpi_rank)
    {
#endif
      for (j = 0; j < optimize->nvariables; ++j)
        {
          optimize->rangemin[j] = optimize->rangemax[j]
            = optimize->value_old[j];
        }
      for (i = 0; ++i < optimize->nbest;)
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
              optimize->rangemin[j]
                = fmin (optimize->rangemin[j],
                        optimize->value_old[i * optimize->nvariables + j]);
              optimize->rangemax[j]
                = fmax (optimize->rangemax[j],
                        optimize->value_old[i * optimize->nvariables + j]);
            }
        }
      for (j = 0; j < optimize->nvariables; ++j)
        {
          d = optimize->tolerance
            * (optimize->rangemax[j] - optimize->rangemin[j]);
          switch (optimize->algorithm)
            {
            case ALGORITHM_MONTE_CARLO:
              d *= 0.5;
              break;
            default:
              if (optimize->nsweeps[j] > 1)
                d /= optimize->nsweeps[j] - 1;
              else
                d = 0.;
            }
          optimize->rangemin[j] -= d;
          optimize->rangemin[j]
            = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
          optimize->rangemax[j] += d;
          optimize->rangemax[j]
            = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
          printf ("%s min=%lg max=%lg\n", optimize->label[j],
                  optimize->rangemin[j], optimize->rangemax[j]);
          fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
                    optimize->label[j], optimize->rangemin[j],
                    optimize->rangemax[j]);
         }
#if HAVE_MPI
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
          MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
        }
    }
  else
    {
      MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD, &mpi_stat);
      MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD, &mpi_stat);
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: end\n");
#endif
}

void
optimize_step ()
{
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: start\n");
#endif
  optimize_algorithm ();
  if (optimize->nsteps)
    optimize_climbing ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: end\n");
#endif
}

void
optimize_iterate ()
{
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: start\n");
#endif
  optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
  optimize->value_old =
    (double *) g_malloc (optimize->nbest * optimize->nvariables *
                         sizeof (double));
  optimize_step ();
  optimize_save_old ();
  optimize_refine ();
  optimize_print ();
  for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
    {
      optimize_step ();
      optimize_merge_old ();
      optimize_refine ();
      optimize_print ();
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: end\n");
#endif
}

void
optimize_free ()
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: start\n");
#endif
  for (j = 0; j < optimize->ninputs; ++j)
    {
      for (i = 0; i < optimize->nexperiments; ++i)
        g_mapped_file_unref (optimize->file[j][i]);
      g_free (optimize->file[j]);
    }
  g_free (optimize->error_old);
  g_free (optimize->value_old);
  g_free (optimize->value);
  g_free (optimize->genetic_variable);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: end\n");
#endif
}
```

```

void
optimize_open ()
{
  GTimeZone *tz;
  GDateTime *t0, *t;
  unsigned int i, j;

#if DEBUG_OPTIMIZE
  char *buffer;
  fprintf (stderr, "optimize_open: start\n");
#endif

  // Getting initial time
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial time\n");
#endif
  tz = g_time_zone_new_utc ();
  t0 = g_date_time_new_now (tz);

  // Obtaining and initing the pseudo-random numbers generator seed
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial seed\n");
#endif
  if (optimize->seed == DEFAULT_RANDOM_SEED)
    optimize->seed = input->seed;
  gsl_rng_set (optimize->rng, optimize->seed);

  // Replacing the working directory
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: replacing the working directory\n");
#endif
  g_chdir (input->directory);

  // Getting results file names
  optimize->result = input->result;
  optimize->variables = input->variables;

  // Obtaining the simulator file
  optimize->simulator = input->simulator;

  // Obtaining the evaluator file
  optimize->evaluator = input->evaluator;

  // Reading the algorithm
  optimize->algorithm = input->algorithm;
  switch (optimize->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      optimize_algorithm = optimize_MonteCarlo;
      break;
    case ALGORITHM_SWEEP:
      optimize_algorithm = optimize_sweep;
      break;
    case ALGORITHM_ORTHOGONAL:
      optimize_algorithm = optimize_orthogonal;
      break;
    default:
      optimize_algorithm = optimize_genetic;
      optimize->mutation_ratio = input->mutation_ratio;
      optimize->reproduction_ratio = input->
reproduction_ratio;
      optimize->adaptation_ratio = input->adaptation_ratio;
    }
  optimize->nvariables = input->nvariables;
  optimize->nsimulations = input->nsimulations;
  optimize->niterations = input->niterations;
  optimize->nbest = input->nbest;
  optimize->tolerance = input->tolerance;
  optimize->nsteps = input->nsteps;
  optimize->nestimates = 0;
  optimize->threshold = input->threshold;
  optimize->stop = 0;
  if (input->nsteps)
    {
      optimize->relaxation = input->relaxation;
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          optimize->nestimates = 2 * optimize->nvariables;
          optimize_estimate_climbing =
optimize_estimate_climbing_coordinates;
          break;
        default:
          optimize->nestimates = input->nestimates;
          optimize_estimate_climbing =
optimize_estimate_climbing_random;
```

```
        }
    }

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
#endif
  optimize->simulation_best
    = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
  optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));

  // Reading the experimental data
#if DEBUG_OPTIMIZE
  buffer = g_get_current_dir ();
  fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
  g_free (buffer);
#endif
  optimize->nexperiments = input->nexperiments;
  optimize->ninputs = input->experiment->ninputs;
  optimize->experiment
    = (char **) alloca (input->nexperiments * sizeof (char *));
  optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
  for (i = 0; i < input->experiment->ninputs; ++i)
    optimize->file[i] = (GMappedFile **)
      g_malloc (input->nexperiments * sizeof (GMappedFile *));
  for (i = 0; i < input->nexperiments; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u\n", i);
#endif
      optimize->experiment[i] = input->experiment[i].
name;
      optimize->weight[i] = input->experiment[i].weight;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
               optimize->experiment[i], optimize->weight[i]);
#endif
      for (j = 0; j < input->experiment->ninputs; ++j)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
#endif
          optimize->file[j][i]
            = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
        }
    }

  // Reading the variables data
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: reading variables\n");
#endif
  optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
  j = input->nvariables * sizeof (double);
  optimize->rangemin = (double *) alloca (j);
  optimize->rangeminabs = (double *) alloca (j);
  optimize->rangemax = (double *) alloca (j);
  optimize->rangemaxabs = (double *) alloca (j);
  optimize->step = (double *) alloca (j);
  j = input->nvariables * sizeof (unsigned int);
  optimize->precision = (unsigned int *) alloca (j);
  optimize->nsweeps = (unsigned int *) alloca (j);
  optimize->nbits = (unsigned int *) alloca (j);
  for (i = 0; i < input->nvariables; ++i)
    {
      optimize->label[i] = input->variable[i].name;
      optimize->rangemin[i] = input->variable[i].rangemin;
      optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
      optimize->rangemax[i] = input->variable[i].rangemax;
      optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
      optimize->precision[i] = input->variable[i].
precision;
      optimize->step[i] = input->variable[i].step;
      optimize->nsweeps[i] = input->variable[i].nsweeps;
      optimize->nbits[i] = input->variable[i].nbits;
    }
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {
      optimize->nsimulations = 1;
      for (i = 0; i < input->nvariables; ++i)
        {
          optimize->nsimulations *= optimize->nsweeps[i];
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
                   optimize->nsweeps[i], optimize->nsimulations);
#endif
```

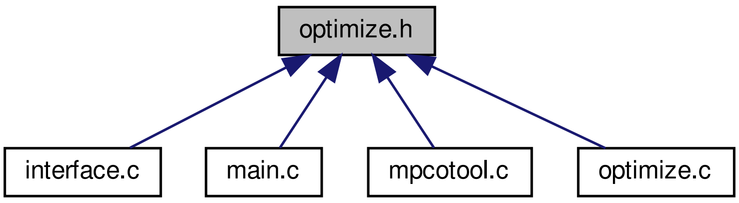
```
        }
    }
  if (optimize->nsteps)
    optimize->climbing
      = (double *) alloca (optimize->nvariables * sizeof (double));

  // Setting error norm
  switch (input->norm)
    {
    case ERROR_NORM_EUCLIDIAN:
      optimize_norm = optimize_norm_euclidian;
      break;
    case ERROR_NORM_MAXIMUM:
      optimize_norm = optimize_norm_maximum;
      break;
    case ERROR_NORM_P:
      optimize_norm = optimize_norm_p;
      optimize->p = input->p;
      break;
    default:
      optimize_norm = optimize_norm_taxicab;
    }

  // Allocating values
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: allocating variables\n");
  fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
           optimize->nvariables, optimize->algorithm);
#endif
  optimize->genetic_variable = NULL;
  if (optimize->algorithm == ALGORITHM_GENETIC)
    {
      optimize->genetic_variable = (GeneticVariable *)
        g_malloc (optimize->nvariables * sizeof (GeneticVariable));
      for (i = 0; i < optimize->nvariables; ++i)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
                   i, optimize->rangemin[i], optimize->rangemax[i],
                   optimize->nbits[i]);
#endif
          optimize->genetic_variable[i].minimum = optimize->
rangemin[i];
          optimize->genetic_variable[i].maximum = optimize->
rangemax[i];
          optimize->genetic_variable[i].nbits = optimize->nbits[i];
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
           optimize->nvariables, optimize->nsimulations);
#endif
  optimize->value = (double *)
    g_malloc ((optimize->nsimulations
               + optimize->nestimates * optimize->nsteps)
              * optimize->nvariables * sizeof (double));

  // Calculating simulations to perform for each task
#if HAVE_MPI
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
           optimize->mpi_rank, ntasks);
#endif
  optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
ntasks;
  optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
ntasks;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing
        = optimize->mpi_rank * optimize->nestimates / ntasks;
      optimize->nend_climbing
        = (1 + optimize->mpi_rank) * optimize->nestimates /
ntasks;
    }
#else
  optimize->nstart = 0;
  optimize->nend = optimize->nsimulations;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing = 0;
      optimize->nend_climbing = optimize->nestimates;
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
           optimize->nend);
```

```
#endif

  // Calculating simulations to perform for each thread
  optimize->thread
    = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
  for (i = 0; i <= nthreads; ++i)
    {
      optimize->thread[i] = optimize->nstart
        + i * (optimize->nend - optimize->nstart) / nthreads;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
               optimize->thread[i]);
#endif
    }
  if (optimize->nsteps)
    optimize->thread_climbing = (unsigned int *)
      alloca ((1 + nthreads_climbing) * sizeof (unsigned int));

  // Opening result files
  optimize->file_result = g_fopen (optimize->result, "w");
  optimize->file_variables = g_fopen (optimize->variables, "w");

  // Performing the algorithm
  switch (optimize->algorithm)
    {
      // Genetic algorithm
    case ALGORITHM_GENETIC:
      optimize_genetic ();
      break;

      // Iterative algorithm
    default:
      optimize_iterate ();
    }

  // Getting calculation time
  t = g_date_time_new_now (tz);
  optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
  g_date_time_unref (t);
  g_date_time_unref (t0);
  g_time_zone_unref (tz);
  printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
  fprintf (optimize->file_result, "%s = %.6lg s\n",
           _("Calculation time"), optimize->calculation_time);

  // Closing result files
  fclose (optimize->file_variables);
  fclose (optimize->file_result);

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: end\n");
#endif
}
```

4.21 optimize.h File Reference

Header file to define the optimization functions.

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



Data Structures

- struct Optimize

  Struct to define the optimization ation data.
- struct ParallelData

  Struct to pass to the GThreads parallelized function.

Functions

- void optimize_input (unsigned int simulation, char ∗input, GMappedFile ∗stencil)
- double optimize_parse (unsigned int simulation, unsigned int experiment)
- double optimize_norm_euclidian (unsigned int simulation)
- double optimize_norm_maximum (unsigned int simulation)
- double optimize_norm_p (unsigned int simulation)
- double optimize_norm_taxicab (unsigned int simulation)
- void optimize_print ()
- void optimize_save_variables (unsigned int simulation, double error)
- void optimize_best (unsigned int simulation, double value)
- void optimize_sequential ()
- void ∗ optimize_thread (ParallelData ∗data)
- void optimize_merge (unsigned int nsaveds, unsigned int ∗simulation_best, double ∗error_best)
- void optimize_synchronise ()
- void optimize_sweep ()
- void optimize_MonteCarlo ()
- void optimize_orthogonal ()
- void optimize_best_climbing (unsigned int simulation, double value)
- void optimize_climbing_sequential (unsigned int simulation)
- void ∗ optimize_climbing_thread (ParallelData ∗data)
- double optimize_estimate_climbing_random (unsigned int variable, unsigned int estimate)
- double optimize_estimate_climbing_coordinates (unsigned int variable, unsigned int estimate)
- void optimize_step_climbing (unsigned int simulation)
- void optimize_climbing ()
- double optimize_genetic_objective (**Entity** ∗entity)
- void optimize_genetic ()
- void optimize_save_old ()
- void optimize_merge_old ()
- void optimize_refine ()
- void optimize_step ()
- void optimize_iterate ()
- void optimize_free ()
- void optimize_open ()

Variables

- int **ntasks**
- unsigned int **nthreads**
- unsigned int nthreads_climbing

  Number of threads for the hill climbing method.
- GMutex **mutex** [1]
- void(∗ optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(∗ optimize_estimate_climbing)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the climbing.
- double(∗ optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Optimize optimize [1]

  Optimization data.

4.21.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file optimize.h.

4.21.2 Function Documentation

4.21.2.1 optimize_best()

```
void optimize_best (
            unsigned int simulation,
            double value )
```

Function to save the best simulations.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 444 of file optimize.c.

```
{
  unsigned int i, j;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: start\n");
  fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
           optimize->nsaveds, optimize->nbest);
#endif
  if (optimize->nsaveds < optimize->nbest
      || value < optimize->error_best[optimize->nsaveds - 1])
    {
      if (optimize->nsaveds < optimize->nbest)
        ++optimize->nsaveds;
      optimize->error_best[optimize->nsaveds - 1] = value;
      optimize->simulation_best[optimize->nsaveds - 1] = simulation;
      for (i = optimize->nsaveds; --i;)
        {
          if (optimize->error_best[i] < optimize->error_best[i - 1])
            {
              j = optimize->simulation_best[i];
              e = optimize->error_best[i];
              optimize->simulation_best[i] = optimize->simulation_best[i - 1];
              optimize->error_best[i] = optimize->error_best[i - 1];
              optimize->simulation_best[i - 1] = j;
              optimize->error_best[i - 1] = e;
            }
          else
            break;
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best: end\n");
#endif
}
```

4.21.2.2 optimize_best_climbing()

```
void optimize_best_climbing (
              unsigned int simulation,
              double value )
```

Function to save the best simulation in a hill climbing method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 806 of file optimize.c.

```
{
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best_climbing: start\n");
  fprintf (stderr,
           "optimize_best_climbing: simulation=%u value=%.14le best=%.14le\n",
           simulation, value, optimize->error_best[0]);
#endif
  if (value < optimize->error_best[0])
    {
      optimize->error_best[0] = value;
      optimize->simulation_best[0] = simulation;
#if DEBUG_OPTIMIZE
      fprintf (stderr,
               "optimize_best_climbing: BEST simulation=%u value=%.14le\n",
               simulation, value);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_best_climbing: end\n");
#endif
}
```

4.21.2.3 optimize_climbing()

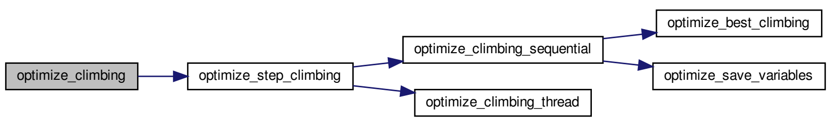
```
void optimize_climbing ( )
```

Function to optimize with a hill climbing method.

Definition at line 1034 of file optimize.c.

```
{
  unsigned int i, j, k, b, s, adjust;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    optimize->climbing[i] = 0.;
  b = optimize->simulation_best[0] * optimize->nvariables;
  s = optimize->nsimulations;
  adjust = 1;
  for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u old_best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      optimize_step_climbing (s);
      k = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing: step=%u best=%u\n",
               i, optimize->simulation_best[0]);
#endif
      if (k == b)
        {
          if (adjust)
            for (j = 0; j < optimize->nvariables; ++j)
              optimize->step[j] *= 0.5;
          for (j = 0; j < optimize->nvariables; ++j)
            optimize->climbing[j] = 0.;
          adjust = 1;
        }
      else
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
#if DEBUG_OPTIMIZE
              fprintf (stderr,
                       "optimize_climbing: best%u=%.14le old%u=%.14le\n",
                       j, optimize->value[k + j], j, optimize->value[b + j]);
#endif
              optimize->climbing[j]
                = (1. - optimize->relaxation) * optimize->climbing[j]
                + optimize->relaxation
                * (optimize->value[k + j] - optimize->value[b + j]);
#if DEBUG_OPTIMIZE
              fprintf (stderr, "optimize_climbing: climbing%u=%.14le\n",
                       j, optimize->climbing[j]);
#endif
            }
          adjust = 0;
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.4 optimize_climbing_sequential()

```
void optimize_climbing_sequential (
            unsigned int simulation )
```

Function to estimate the hill climbing sequentially.

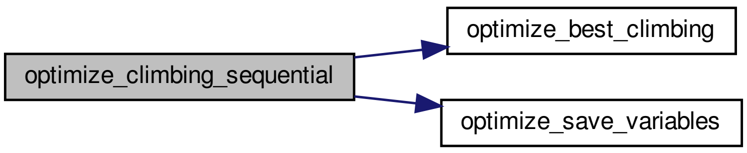
Parameters

simulation	Simulation number.

Definition at line 834 of file optimize.c.

```
{
  double e;
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: start\n");
  fprintf (stderr, "optimize_climbing_sequential: nstart_climbing=%u "
           "nend_climbing=%u\n",
           optimize->nstart_climbing, optimize->
nend_climbing);
#endif
  for (i = optimize->nstart_climbing; i < optimize->nend_climbing; ++i)
    {
      j = simulation + i;
      e = optimize_norm (j);
      optimize_best_climbing (j, e);
      optimize_save_variables (j, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_sequential: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.5 optimize_climbing_thread()

```
void* optimize_climbing_thread (
            ParallelData * data )
```

Function to estimate the hill climbing on a thread.

Returns

NULL

Parameters

data	Function data.

Definition at line 870 of file optimize.c.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: thread=%u start=%u end=%u\n",
           thread,
           optimize->thread_climbing[thread],
           optimize->thread_climbing[thread + 1]);
#endif
  for (i = optimize->thread_climbing[thread];
       i < optimize->thread_climbing[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best_climbing (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_climbing_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_climbing_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

4.21.2.6 optimize_estimate_climbing_coordinates()

```
double optimize_estimate_climbing_coordinates (
            unsigned int variable,
            unsigned int estimate )
```

Function to estimate a component of the hill climbing vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 935 of file optimize.c.

```
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_coordinates: start\n");
#endif
  x = optimize->climbing[variable];
  if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
    {
      if (estimate & 1)
        x += optimize->step[variable];
      else
        x -= optimize->step[variable];
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr,
           "optimize_estimate_climbing_coordinates: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_coordinates: end\n");
#endif
  return x;
}
```

4.21.2.7 optimize_estimate_climbing_random()

```
double optimize_estimate_climbing_random (
            unsigned int variable,
            unsigned int estimate )
```

Function to estimate a component of the hill climbing vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 911 of file optimize.c.

```
{
  double x;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: start\n");
#endif
  x = optimize->climbing[variable]
    + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
step[variable];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_estimate_climbing_random: climbing%u=%lg\n",
           variable, x);
  fprintf (stderr, "optimize_estimate_climbing_random: end\n");
#endif
  return x;
}
```

4.21.2.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the Optimize struct.

Definition at line 1391 of file optimize.c.

```
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: start\n");
#endif
  for (j = 0; j < optimize->ninputs; ++j)
    {
      for (i = 0; i < optimize->nexperiments; ++i)
        g_mapped_file_unref (optimize->file[j][i]);
      g_free (optimize->file[j]);
    }
  g_free (optimize->error_old);
  g_free (optimize->value_old);
  g_free (optimize->value);
  g_free (optimize->genetic_variable);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_free: end\n");
#endif
}
```

4.21.2.9 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1131 of file optimize.c.

```
{
  double *best_variable = NULL;
  char *best_genome = NULL;
  double best_objective = 0.;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: start\n");
  fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
           nthreads);
  fprintf (stderr,
           "optimize_genetic: nvariables=%u population=%u generations=%u\n",
           optimize->nvariables, optimize->
nsimulations, optimize->niterations);
  fprintf (stderr,
           "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
           optimize->mutation_ratio, optimize->
reproduction_ratio,
           optimize->adaptation_ratio);
#endif
  genetic_algorithm_default (optimize->nvariables,
                             optimize->genetic_variable,
                             optimize->nsimulations,
                             optimize->niterations,
                             optimize->mutation_ratio,
                             optimize->reproduction_ratio,
                             optimize->adaptation_ratio,
                             optimize->seed,
                             optimize->threshold,
                             &optimize_genetic_objective,
                             &best_genome, &best_variable, &best_objective);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: the best\n");
#endif
  optimize->error_old = (double *) g_malloc (sizeof (double));
  optimize->value_old
```

```
    = (double *) g_malloc (optimize->nvariables * sizeof (double));
  optimize->error_old[0] = best_objective;
  memcpy (optimize->value_old, best_variable,
          optimize->nvariables * sizeof (double));
  g_free (best_genome);
  g_free (best_variable);
  optimize_print ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic: end\n");
#endif
}
```

4.21.2.10 optimize_genetic_objective()

```
double optimize_genetic_objective (
            Entity * entity )
```

Function to calculate the objective function of an entity.

Returns

objective function value.

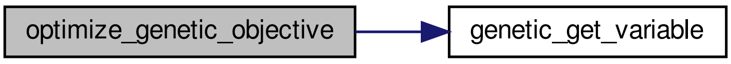
Parameters

entity	entity data.

Definition at line 1098 of file optimize.c.

```
{
  unsigned int j;
  double objective;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: start\n");
#endif
  for (j = 0; j < optimize->nvariables; ++j)
    {
      optimize->value[entity->id * optimize->nvariables + j]
        = genetic_get_variable (entity, optimize->genetic_variable + j);
    }
  objective = optimize_norm (entity->id);
  g_mutex_lock (mutex);
  for (j = 0; j < optimize->nvariables; ++j)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
      fprintf (optimize->file_variables, buffer,
               genetic_get_variable (entity, optimize->genetic_variable + j));
    }
  fprintf (optimize->file_variables, "%.14le\n", objective);
  g_mutex_unlock (mutex);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_genetic_objective: end\n");
#endif
  return objective;
}
```

Here is the call graph for this function:



4.21.2.11 optimize_input()

```
void optimize_input (
             unsigned int simulation,
             char * input,
             GMappedFile * stencil )
```

Function to write the simulation input file.

Parameters

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

Definition at line 93 of file optimize.c.

```
{
  char buffer[32], value[32];
  GRegex *regex;
  FILE *file;
  char *buffer2, *buffer3 = NULL, *content;
  gsize length;
  unsigned int i;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: start\n");
#endif

  // Checking the file
  if (!stencil)
    goto optimize_input_end;

  // Opening stencil
  content = g_mapped_file_get_contents (stencil);
  length = g_mapped_file_get_length (stencil);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
#endif
  file = g_fopen (input, "w");

  // Parsing stencil
  for (i = 0; i < optimize->nvariables; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: variable=%u\n", i);
#endif
      snprintf (buffer, 32, "@variable%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      if (i == 0)
        {
          buffer2 = g_regex_replace_literal (regex, content, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
#endif
        }
      else
        {
          length = strlen (buffer3);
          buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
                                             optimize->label[i],
                                             (GRegexMatchFlags) 0, NULL);
          g_free (buffer3);
        }
      g_regex_unref (regex);
      length = strlen (buffer2);
      snprintf (buffer, 32, "@value%u@", i + 1);
      regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
                           NULL);
      snprintf (value, 32, format[optimize->precision[i]],
                optimize->value[simulation * optimize->nvariables + i]);

#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_input: value=%s\n", value);
#endif
      buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
                                         (GRegexMatchFlags) 0, NULL);
      g_free (buffer2);
      g_regex_unref (regex);
    }

  // Saving input file
  fwrite (buffer3, strlen (buffer3), sizeof (char), file);
  g_free (buffer3);
  fclose (file);

optimize_input_end:
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_input: end\n");
#endif
  return;
}
```

4.21.2.12 optimize_iterate()

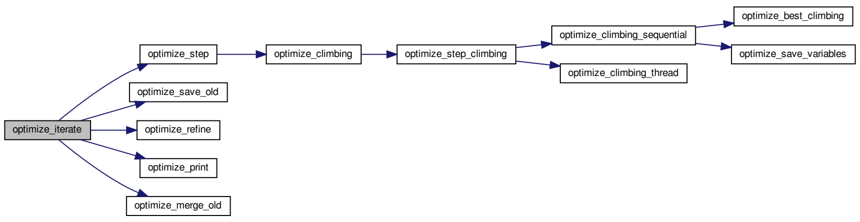
```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1361 of file optimize.c.

```
{
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: start\n");
#endif
  optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
  optimize->value_old =
    (double *) g_malloc (optimize->nbest * optimize->nvariables *
                         sizeof (double));
  optimize_step ();
  optimize_save_old ();
  optimize_refine ();
  optimize_print ();
  for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
    {
      optimize_step ();
      optimize_merge_old ();
      optimize_refine ();
      optimize_print ();
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_iterate: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.13 optimize_merge()

```
void optimize_merge (
            unsigned int nsaveds,
            unsigned int * simulation_best,
            double * error_best )
```

Function to merge the 2 optimization 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 557 of file optimize.c.

```
{
  unsigned int i, j, k, s[optimize->nbest];
  double e[optimize->nbest];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: start\n");
#endif
  i = j = k = 0;
  do
    {
      if (i == optimize->nsaveds)
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
          if (j == nsaveds)
            break;
        }
      else if (j == nsaveds)
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
          if (i == optimize->nsaveds)
            break;
        }
      else if (optimize->error_best[i] > error_best[j])
        {
          s[k] = simulation_best[j];
          e[k] = error_best[j];
          ++j;
          ++k;
        }
      else
        {
          s[k] = optimize->simulation_best[i];
          e[k] = optimize->error_best[i];
          ++i;
          ++k;
        }
    }
  while (k < optimize->nbest);
  optimize->nsaveds = k;
  memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
  memcpy (optimize->error_best, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge: end\n");
#endif
}
```

4.21.2.14 optimize_merge_old()

```
void optimize_merge_old ( )
```

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

Definition at line 1212 of file optimize.c.

```
{
  unsigned int i, j, k;
  double v[optimize->nbest * optimize->nvariables], e[
optimize->nbest],
    *enew, *eold;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: start\n");
#endif
  enew = optimize->error_best;
  eold = optimize->error_old;
  i = j = k = 0;
  do
    {
      if (*enew < *eold)
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value
                  + optimize->simulation_best[i] *
optimize->nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *enew;
          ++k;
          ++enew;
          ++i;
        }
      else
        {
          memcpy (v + k * optimize->nvariables,
                  optimize->value_old + j * optimize->
nvariables,
                  optimize->nvariables * sizeof (double));
          e[k] = *eold;
          ++k;
          ++eold;
          ++j;
        }
    }
  while (k < optimize->nbest);
  memcpy (optimize->value_old, v, k * optimize->
nvariables * sizeof (double));
  memcpy (optimize->error_old, e, k * sizeof (double));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_merge_old: end\n");
#endif
}
```

4.21.2.15 optimize_MonteCarlo()

void optimize_MonteCarlo ( )

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file optimize.c.

```
{
  unsigned int i, j;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    for (j = 0; j < optimize->nvariables; ++j)
      optimize->value[i * optimize->nvariables + j]
        = optimize->rangemin[j] + gsl_rng_uniform (optimize->
rng)
        * (optimize->rangemax[j] - optimize->rangemin[j]);
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_MonteCarlo: end\n");
#endif
}
```

4.21.2.16 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
              unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

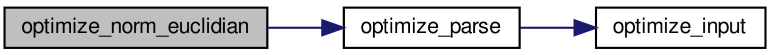
Parameters

simulation	simulation number.

Definition at line 292 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = optimize_parse (simulation, i);
      e += ei * ei;
    }
  e = sqrt (e);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_euclidian: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.21.2.17 optimize_norm_maximum()

```
double optimize_norm_maximum (
            unsigned int simulation )
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

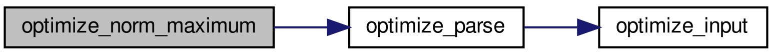
Parameters

simulation	simulation number.

Definition at line 319 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e = fmax (e, ei);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_maximum: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.21.2.18 optimize_norm_p()

```
double optimize_norm_p (
              unsigned int simulation )
```

Function to calculate the P error norm.

Returns

P error norm.

Parameters

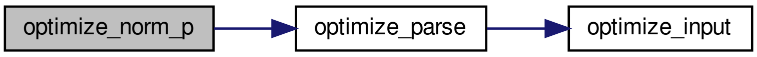
simulation	simulation number.

Definition at line 345 of file optimize.c.

```
{
  double e, ei;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_p: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    {
      ei = fabs (optimize_parse (simulation, i));
      e += pow (ei, optimize->p);
    }
  e = pow (e, 1. / optimize->p);
#if DEBUG_OPTIMIZE
```

```
  fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_p: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.21.2.19 optimize_norm_taxicab()

```
double optimize_norm_taxicab (
            unsigned int simulation )
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

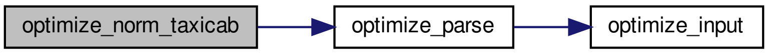
Parameters

simulation	simulation number.

Definition at line 372 of file optimize.c.

```
{
  double e;
  unsigned int i;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: start\n");
#endif
  e = 0.;
  for (i = 0; i < optimize->nexperiments; ++i)
    e += fabs (optimize_parse (simulation, i));
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
  fprintf (stderr, "optimize_norm_taxicab: end\n");
#endif
  return e;
}
```

Here is the call graph for this function:



4.21.2.20 optimize_open()

```
void optimize_open ( )
```

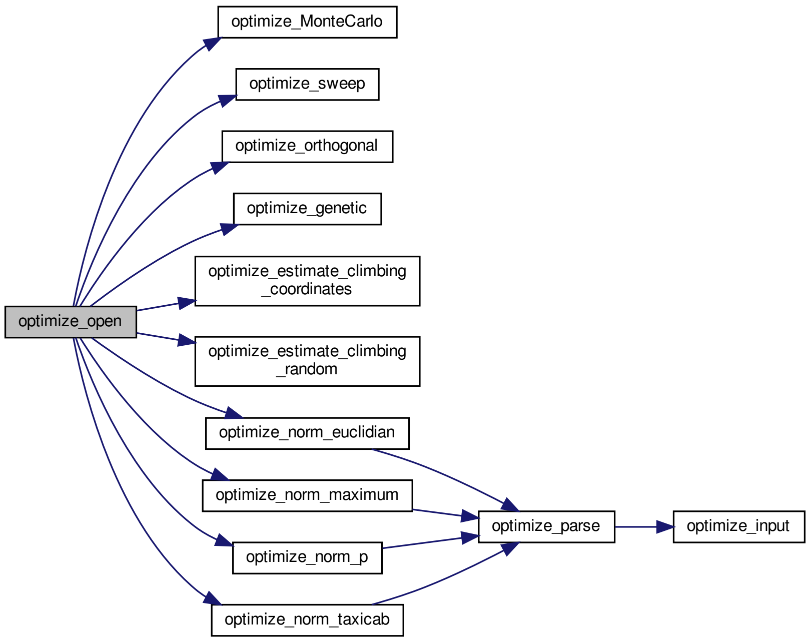
Function to open and perform a optimization.

Definition at line 1416 of file optimize.c.

```
{
  GTimeZone *tz;
  GDateTime *t0, *t;
  unsigned int i, j;

#if DEBUG_OPTIMIZE
  char *buffer;
  fprintf (stderr, "optimize_open: start\n");
#endif

  // Getting initial time
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial time\n");
#endif
  tz = g_time_zone_new_utc ();
  t0 = g_date_time_new_now (tz);

  // Obtaining and initing the pseudo-random numbers generator seed
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: getting initial seed\n");
#endif
  if (optimize->seed == DEFAULT_RANDOM_SEED)
    optimize->seed = input->seed;
  gsl_rng_set (optimize->rng, optimize->seed);

  // Replacing the working directory
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: replacing the working directory\n");
#endif
  g_chdir (input->directory);

  // Getting results file names
  optimize->result = input->result;
  optimize->variables = input->variables;

  // Obtaining the simulator file
  optimize->simulator = input->simulator;

  // Obtaining the evaluator file
  optimize->evaluator = input->evaluator;

  // Reading the algorithm
  optimize->algorithm = input->algorithm;
  switch (optimize->algorithm)
    {
    case ALGORITHM_MONTE_CARLO:
      optimize_algorithm = optimize_MonteCarlo;
      break;
    case ALGORITHM_SWEEP:
      optimize_algorithm = optimize_sweep;
      break;
    case ALGORITHM_ORTHOGONAL:
      optimize_algorithm = optimize_orthogonal;
      break;
    default:
      optimize_algorithm = optimize_genetic;
      optimize->mutation_ratio = input->mutation_ratio;
      optimize->reproduction_ratio = input->reproduction_ratio;
      optimize->adaptation_ratio = input->adaptation_ratio;
    }
  optimize->nvariables = input->nvariables;
  optimize->nsimulations = input->nsimulations;
  optimize->niterations = input->niterations;
  optimize->nbest = input->nbest;
  optimize->tolerance = input->tolerance;
  optimize->nsteps = input->nsteps;
  optimize->nestimates = 0;
  optimize->threshold = input->threshold;
  optimize->stop = 0;
  if (input->nsteps)
    {
      optimize->relaxation = input->relaxation;
      switch (input->climbing)
        {
        case CLIMBING_METHOD_COORDINATES:
          optimize->nestimates = 2 * optimize->nvariables;
          optimize_estimate_climbing = optimize_estimate_climbing_coordinates;
          break;
        default:
          optimize->nestimates = input->nestimates;
          optimize_estimate_climbing = optimize_estimate_climbing_random;
        }
    }

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
#endif
  optimize->simulation_best
    = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
  optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));

  // Reading the experimental data
#if DEBUG_OPTIMIZE
  buffer = g_get_current_dir ();
  fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
  g_free (buffer);
#endif
  optimize->nexperiments = input->nexperiments;
  optimize->ninputs = input->experiment->ninputs;
  optimize->experiment
    = (char **) alloca (input->nexperiments * sizeof (char *));
  optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
  for (i = 0; i < input->experiment->ninputs; ++i)
    optimize->file[i] = (GMappedFile **)
      g_malloc (input->nexperiments * sizeof (GMappedFile *));
  for (i = 0; i < input->nexperiments; ++i)
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u\n", i);
#endif
      optimize->experiment[i] = input->experiment[i].name;
      optimize->weight[i] = input->experiment[i].weight;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
               optimize->experiment[i], optimize->weight[i]);
#endif
      for (j = 0; j < input->experiment->ninputs; ++j)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
#endif
          optimize->file[j][i]
            = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
        }
    }

  // Reading the variables data
```

```
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: reading variables\n");
#endif
  optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
  j = input->nvariables * sizeof (double);
  optimize->rangemin = (double *) alloca (j);
  optimize->rangeminabs = (double *) alloca (j);
  optimize->rangemax = (double *) alloca (j);
  optimize->rangemaxabs = (double *) alloca (j);
  optimize->step = (double *) alloca (j);
  j = input->nvariables * sizeof (unsigned int);
  optimize->precision = (unsigned int *) alloca (j);
  optimize->nsweeps = (unsigned int *) alloca (j);
  optimize->nbits = (unsigned int *) alloca (j);
  for (i = 0; i < input->nvariables; ++i)
    {
      optimize->label[i] = input->variable[i].name;
      optimize->rangemin[i] = input->variable[i].rangemin;
      optimize->rangeminabs[i] = input->variable[i].rangeminabs;
      optimize->rangemax[i] = input->variable[i].rangemax;
      optimize->rangemaxabs[i] = input->variable[i].rangemaxabs;
      optimize->precision[i] = input->variable[i].precision;
      optimize->step[i] = input->variable[i].step;
      optimize->nsweeps[i] = input->variable[i].nsweeps;
      optimize->nbits[i] = input->variable[i].nbits;
    }
  if (input->algorithm == ALGORITHM_SWEEP
      || input->algorithm == ALGORITHM_ORTHOGONAL)
    {
      optimize->nsimulations = 1;
      for (i = 0; i < input->nvariables; ++i)
        {
          optimize->nsimulations *= optimize->nsweeps[i];
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
                   optimize->nsweeps[i], optimize->nsimulations);
#endif
        }
    }
  if (optimize->nsteps)
    optimize->climbing
      = (double *) alloca (optimize->nvariables * sizeof (double));

  // Setting error norm
  switch (input->norm)
    {
    case ERROR_NORM_EUCLIDIAN:
      optimize_norm = optimize_norm_euclidian;
      break;
    case ERROR_NORM_MAXIMUM:
      optimize_norm = optimize_norm_maximum;
      break;
    case ERROR_NORM_P:
      optimize_norm = optimize_norm_p;
      optimize->p = input->p;
      break;
    default:
      optimize_norm = optimize_norm_taxicab;
    }

  // Allocating values
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: allocating variables\n");
  fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
           optimize->nvariables, optimize->algorithm);
#endif
  optimize->genetic_variable = NULL;
  if (optimize->algorithm == ALGORITHM_GENETIC)
    {
      optimize->genetic_variable = (GeneticVariable *)
        g_malloc (optimize->nvariables * sizeof (GeneticVariable));
      for (i = 0; i < optimize->nvariables; ++i)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
                   i, optimize->rangemin[i], optimize->rangemax[i],
                   optimize->nbits[i]);
#endif
          optimize->genetic_variable[i].minimum = optimize->rangemin[i];
          optimize->genetic_variable[i].maximum = optimize->rangemax[i];
          optimize->genetic_variable[i].nbits = optimize->nbits[i];
        }
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
           optimize->nvariables, optimize->nsimulations);
#endif
  optimize->value = (double *)
    g_malloc ((optimize->nsimulations
               + optimize->nestimates * optimize->nsteps)
              * optimize->nvariables * sizeof (double));

  // Calculating simulations to perform for each task
#if HAVE_MPI
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
           optimize->mpi_rank, ntasks);
#endif
  optimize->nstart = optimize->mpi_rank * optimize->nsimulations / ntasks;
  optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations / ntasks;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing
        = optimize->mpi_rank * optimize->nestimates / ntasks;
      optimize->nend_climbing
        = (1 + optimize->mpi_rank) * optimize->nestimates / ntasks;
    }
#else
  optimize->nstart = 0;
  optimize->nend = optimize->nsimulations;
  if (optimize->nsteps)
    {
      optimize->nstart_climbing = 0;
      optimize->nend_climbing = optimize->nestimates;
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
           optimize->nend);
#endif

  // Calculating simulations to perform for each thread
  optimize->thread
    = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
  for (i = 0; i <= nthreads; ++i)
    {
      optimize->thread[i] = optimize->nstart
        + i * (optimize->nend - optimize->nstart) / nthreads;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
               optimize->thread[i]);
#endif
    }
  if (optimize->nsteps)
    optimize->thread_climbing = (unsigned int *)
      alloca ((1 + nthreads_climbing) * sizeof (unsigned int));

  // Opening result files
  optimize->file_result = g_fopen (optimize->result, "w");
  optimize->file_variables = g_fopen (optimize->variables, "w");

  // Performing the algorithm
  switch (optimize->algorithm)
    {
      // Genetic algorithm
    case ALGORITHM_GENETIC:
      optimize_genetic ();
      break;

      // Iterative algorithm
    default:
      optimize_iterate ();
    }

  // Getting calculation time
  t = g_date_time_new_now (tz);
  optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
  g_date_time_unref (t);
  g_date_time_unref (t0);
  g_time_zone_unref (tz);
  printf ("%s = %.6lg s\n", _("Calculation time"), optimize->
calculation_time);
  fprintf (optimize->file_result, "%s = %.6lg s\n",
           _("Calculation time"), optimize->calculation_time);

  // Closing result files
  fclose (optimize->file_variables);
  fclose (optimize->file_result);

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_open: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.21 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file optimize.c.

```
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += (l + gsl_rng_uniform (optimize->rng))
              * (optimize->rangemax[j] - optimize->
rangemin[j])
              / optimize->nsweeps[j];
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_orthogonal: end\n");
#endif
}
```

4.21.2.22 optimize_parse()

```
double optimize_parse (
            unsigned int simulation,
            unsigned int experiment )
```

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

Returns

Objective function value.

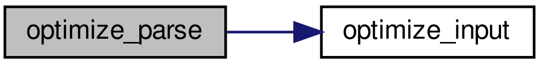
Parameters

simulation	Simulation number.
experiment	Experiment number.

Definition at line 182 of file optimize.c.

```
{
  unsigned int i;
  double e;
  char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
    *buffer3, *buffer4;
  FILE *file_result;

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: start\n");
  fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
           simulation, experiment);
#endif

  // Opening input files
  for (i = 0; i < optimize->ninputs; ++i)
    {
      snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
#endif
      optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
    }
  for (; i < MAX_NINPUTS; ++i)
    strcpy (&input[i][0], "");
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: parsing end\n");
#endif

  // Performing the simulation
  snprintf (output, 32, "output-%u-%u", simulation, experiment);
  buffer2 = g_path_get_dirname (optimize->simulator);
  buffer3 = g_path_get_basename (optimize->simulator);
  buffer4 = g_build_filename (buffer2, buffer3, NULL);
  snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
            buffer4, input[0], input[1], input[2], input[3], input[4],
            input[5], input[6], input[7], output);
  g_free (buffer4);
  g_free (buffer3);
  g_free (buffer2);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: %s\n", buffer);
#endif
  system (buffer);

  // Checking the objective value function
  if (optimize->evaluator)
    {
      snprintf (result, 32, "result-%u-%u", simulation, experiment);
      buffer2 = g_path_get_dirname (optimize->evaluator);
      buffer3 = g_path_get_basename (optimize->evaluator);
      buffer4 = g_build_filename (buffer2, buffer3, NULL);
      snprintf (buffer, 512, "\"%s\" %s %s %s",
                buffer4, output, optimize->experiment[experiment], result);
      g_free (buffer4);
      g_free (buffer3);
      g_free (buffer2);
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: %s\n", buffer);
      fprintf (stderr, "optimize_parse: result=%s\n", result);
#endif
      system (buffer);
      file_result = g_fopen (result, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }
  else
    {
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_parse: output=%s\n", output);
#endif
      strcpy (result, "");
      file_result = g_fopen (output, "r");
      e = atof (fgets (buffer, 512, file_result));
      fclose (file_result);
    }

  // Removing files
#if !DEBUG_OPTIMIZE
  for (i = 0; i < optimize->ninputs; ++i)
    {
      if (optimize->file[i][0])
        {
          snprintf (buffer, 512, RM " %s", &input[i][0]);
          system (buffer);
        }
    }
  snprintf (buffer, 512, RM " %s %s", output, result);
  system (buffer);
#endif

  // Processing pending events
  if (show_pending)
    show_pending ();

#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_parse: end\n");
#endif

  // Returning the objective function
  return e * optimize->weight[experiment];
}
```

Here is the call graph for this function:



4.21.2.23 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file optimize.c.

```
{
  unsigned int i;
  char buffer[512];
#if HAVE_MPI
  if (optimize->mpi_rank)
    return;
#endif
  printf ("%s\n", _("Best result"));
  fprintf (optimize->file_result, "%s\n", _("Best result"));
  printf ("error = %.15le\n", optimize->error_old[0]);
  fprintf (optimize->file_result, "error = %.15le\n",
optimize->error_old[0]);
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 512, "%s = %s\n",
                optimize->label[i], format[optimize->
precision[i]]);
      printf (buffer, optimize->value_old[i]);
      fprintf (optimize->file_result, buffer, optimize->
value_old[i]);
    }
  fflush (optimize->file_result);
}
```

4.21.2.24 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1260 of file optimize.c.

```
{
  unsigned int i, j;
  double d;
#if HAVE_MPI
  MPI_Status mpi_stat;
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: start\n");
#endif
#if HAVE_MPI
  if (!optimize->mpi_rank)
    {
#endif
      for (j = 0; j < optimize->nvariables; ++j)
        {
          optimize->rangemin[j] = optimize->rangemax[j]
            = optimize->value_old[j];
        }
      for (i = 0; ++i < optimize->nbest;)
        {
          for (j = 0; j < optimize->nvariables; ++j)
            {
              optimize->rangemin[j]
                = fmin (optimize->rangemin[j],
                        optimize->value_old[i * optimize->nvariables + j]);
              optimize->rangemax[j]
                = fmax (optimize->rangemax[j],
                        optimize->value_old[i * optimize->nvariables + j]);
            }
        }
      for (j = 0; j < optimize->nvariables; ++j)
        {
          d = optimize->tolerance
            * (optimize->rangemax[j] - optimize->rangemin[j]);
          switch (optimize->algorithm)
            {
            case ALGORITHM_MONTE_CARLO:
              d *= 0.5;
              break;
            default:
              if (optimize->nsweeps[j] > 1)
                d /= optimize->nsweeps[j] - 1;
              else
                d = 0.;
            }
          optimize->rangemin[j] -= d;
          optimize->rangemin[j]
            = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
          optimize->rangemax[j] += d;
          optimize->rangemax[j]
            = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
          printf ("%s min=%lg max=%lg\n", optimize->label[j],
                  optimize->rangemin[j], optimize->rangemax[j]);
          fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
                   optimize->label[j], optimize->rangemin[j],
                   optimize->rangemax[j]);
        }
#if HAVE_MPI
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
          MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
                    1, MPI_COMM_WORLD);
        }
    }
  else
    {
      MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
1,
                MPI_COMM_WORLD, &mpi_stat);
      MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
1,
                MPI_COMM_WORLD, &mpi_stat);
    }
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_refine: end\n");
#endif
}
```

4.21.2.25 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1180 of file optimize.c.

```
{
  unsigned int i, j;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_old: start\n");
  fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
#endif
  memcpy (optimize->error_old, optimize->error_best,
          optimize->nbest * sizeof (double));
  for (i = 0; i < optimize->nbest; ++i)
    {
      j = optimize->simulation_best[i];
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
#endif
      memcpy (optimize->value_old + i * optimize->
nvariables,
              optimize->value + j * optimize->nvariables,
              optimize->nvariables * sizeof (double));
    }
#if DEBUG_OPTIMIZE
  for (i = 0; i < optimize->nvariables; ++i)
    fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
             i, optimize->value_old[i]);
  fprintf (stderr, "optimize_save_old: end\n");
#endif
}
```

4.21.2.26 optimize_save_variables()

```
void optimize_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 419 of file optimize.c.

```
{
  unsigned int i;
  char buffer[64];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: start\n");
#endif
  for (i = 0; i < optimize->nvariables; ++i)
    {
      snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
      fprintf (optimize->file_variables, buffer,
               optimize->value[simulation * optimize->
nvariables + i]);
    }
  fprintf (optimize->file_variables, "%.14le\n", error);
  fflush (optimize->file_variables);
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_save_variables: end\n");
#endif
}
```

4.21.2.27 optimize_sequential()

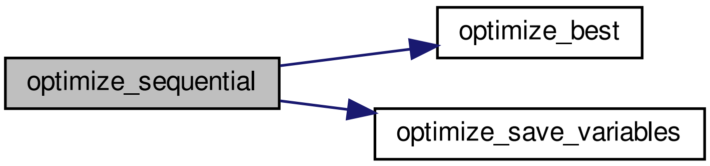
```
void optimize_sequential ( )
```

Function to optimize sequentially.

Definition at line 485 of file optimize.c.

```
{
  unsigned int i;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: start\n");
  fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
           optimize->nstart, optimize->nend);
#endif
  for (i = optimize->nstart; i < optimize->nend; ++i)
    {
      e = optimize_norm (i);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        {
          optimize->stop = 1;
          break;
        }
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sequential: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.28 optimize_step()

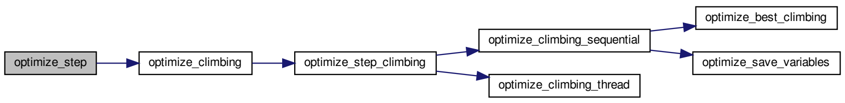
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1344 of file optimize.c.

```
{
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: start\n");
#endif
  optimize_algorithm ();
  if (optimize->nsteps)
    optimize_climbing ();
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.29 optimize_step_climbing()

```
void optimize_step_climbing (
            unsigned int simulation )
```

Function to do a step of the hill climbing method.

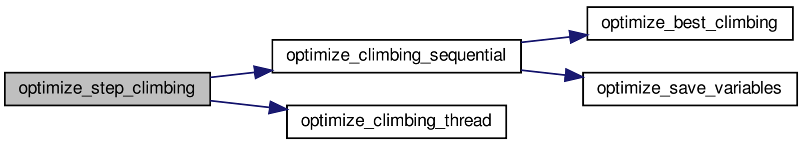
Parameters

simulation	Simulation number.

Definition at line 965 of file optimize.c.

```
{
  GThread *thread[nthreads_climbing];
  ParallelData data[nthreads_climbing];
  unsigned int i, j, k, b;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: start\n");
#endif
  for (i = 0; i < optimize->nestimates; ++i)
    {
      k = (simulation + i) * optimize->nvariables;
      b = optimize->simulation_best[0] * optimize->nvariables;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_step_climbing: simulation=%u best=%u\n",
               simulation + i, optimize->simulation_best[0]);
#endif
      for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
        {
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u best%u=%.14le\n",
                   i, j, optimize->value[b]);
#endif
          optimize->value[k]
            = optimize->value[b] + optimize_estimate_climbing (j, i);
          optimize->value[k] = fmin (fmax (optimize->value[k],
                                           optimize->rangeminabs[j]),
                                     optimize->rangemaxabs[j]);
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: estimate=%u variable%u=%.14le\n",
                   i, j, optimize->value[k]);
#endif
        }
    }
  if (nthreads_climbing == 1)
    optimize_climbing_sequential (simulation);
  else
    {
      for (i = 0; i <= nthreads_climbing; ++i)
        {
          optimize->thread_climbing[i]
            = simulation + optimize->nstart_climbing
            + i * (optimize->nend_climbing - optimize->nstart_climbing)
            / nthreads_climbing;
#if DEBUG_OPTIMIZE
          fprintf (stderr,
                   "optimize_step_climbing: i=%u thread_climbing=%u\n",
                   i, optimize->thread_climbing[i]);
#endif
        }
      for (i = 0; i < nthreads_climbing; ++i)
        {
          data[i].thread = i;
          thread[i] = g_thread_new
            (NULL, (GThreadFunc) optimize_climbing_thread, &data[i]);
        }
      for (i = 0; i < nthreads_climbing; ++i)
        g_thread_join (thread[i]);
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_step_climbing: end\n");
#endif
}
```

Here is the call graph for this function:



4.21.2.30 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file optimize.c.

```
{
  unsigned int i, j, k, l;
  double e;
  GThread *thread[nthreads];
  ParallelData data[nthreads];
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: start\n");
#endif
  for (i = 0; i < optimize->nsimulations; ++i)
    {
      k = i;
      for (j = 0; j < optimize->nvariables; ++j)
        {
          l = k % optimize->nsweeps[j];
          k /= optimize->nsweeps[j];
          e = optimize->rangemin[j];
          if (optimize->nsweeps[j] > 1)
            e += l * (optimize->rangemax[j] - optimize->
rangemin[j])
              / (optimize->nsweeps[j] - 1);
          optimize->value[i * optimize->nvariables + j] = e;
        }
    }
  optimize->nsaveds = 0;
  if (nthreads <= 1)
    optimize_sequential ();
  else
    {
      for (i = 0; i < nthreads; ++i)
        {
          data[i].thread = i;
          thread[i]
            = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
        }
      for (i = 0; i < nthreads; ++i)
        g_thread_join (thread[i]);
    }
#if HAVE_MPI
  // Communicating tasks results
  optimize_synchronise ();
#endif
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_sweep: end\n");
#endif
}
```

4.21.2.31 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file optimize.c.

```
{
  unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
  double error_best[optimize->nbest];
  MPI_Status mpi_stat;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: start\n");
#endif
  if (optimize->mpi_rank == 0)
    {
      for (i = 1; i < ntasks; ++i)
        {
          MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
                    MPI_COMM_WORLD, &mpi_stat);
          optimize_merge (nsaveds, simulation_best, error_best);
          MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
          if (stop)
            optimize->stop = 1;
        }
      for (i = 1; i < ntasks; ++i)
        MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
    }
  else
    {
      MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
      MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
                MPI_COMM_WORLD);
      MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
      MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
      if (stop)
        optimize->stop = 1;
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_synchronise: end\n");
#endif
}
```

4.21.2.32 optimize_thread()

```
void* optimize_thread (
            ParallelData * data )
```

Function to optimize on a thread.

Returns

NULL.

Parameters

data	Function data.

Definition at line 519 of file optimize.c.

```
{
  unsigned int i, thread;
  double e;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: start\n");
#endif
  thread = data->thread;
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
           optimize->thread[thread], optimize->thread[thread + 1]);
#endif
  for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
    {
      e = optimize_norm (i);
      g_mutex_lock (mutex);
      optimize_best (i, e);
      optimize_save_variables (i, e);
      if (e < optimize->threshold)
        optimize->stop = 1;
      g_mutex_unlock (mutex);
      if (optimize->stop)
        break;
#if DEBUG_OPTIMIZE
      fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
#endif
    }
#if DEBUG_OPTIMIZE
  fprintf (stderr, "optimize_thread: end\n");
#endif
  g_thread_exit (NULL);
  return NULL;
}
```

4.22 optimize.h

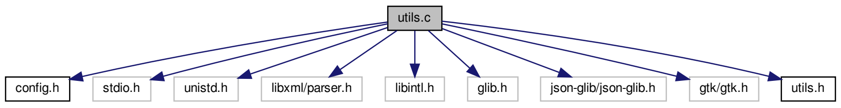
```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef OPTIMIZE__H
#define OPTIMIZE__H 1

typedef struct
{
  GMappedFile **file[MAX_NINPUTS];
  char **experiment;
  char **label;
  gsl_rng *rng;
  GeneticVariable *genetic_variable;
  FILE *file_result;
  FILE *file_variables;
  char *result;
  char *variables;
  char *simulator;
  char *evaluator;
  double *value;
  double *rangemin;
  double *rangemax;
  double *rangeminabs;
  double *rangemaxabs;
  double *error_best;
  double *weight;
  double *step;
  double *climbing;
  double *value_old;
  double *error_old;
  unsigned int *precision;
  unsigned int *nsweeps;
  unsigned int *nbits;
  unsigned int *thread;
  unsigned int *thread_climbing;
  unsigned int *simulation_best;
  double tolerance;
  double mutation_ratio;
  double reproduction_ratio;
  double adaptation_ratio;
  double relaxation;
  double calculation_time;
  double p;
  double threshold;
  unsigned long int seed;
  unsigned int nvariables;
  unsigned int nexperiments;
  unsigned int ninputs;
  unsigned int nsimulations;
  unsigned int nsteps;
  unsigned int nestimates;
  unsigned int algorithm;
  unsigned int nstart;
  unsigned int nend;
  unsigned int nstart_climbing;
  unsigned int nend_climbing;
  unsigned int niterations;
  unsigned int nbest;
  unsigned int nsaveds;
  unsigned int stop;
#if HAVE_MPI
  int mpi_rank;
#endif
} Optimize;

typedef struct
{
  unsigned int thread;
} ParallelData;

// Global variables
extern int ntasks;
extern unsigned int nthreads;
extern unsigned int nthreads_climbing;
extern GMutex mutex[1];
extern void (*optimize_algorithm) ();
extern double (*optimize_estimate_climbing) (unsigned int variable,
                                             unsigned int estimate);
extern double (*optimize_norm) (unsigned int simulation);
extern Optimize optimize[1];

// Public functions
void optimize_input (unsigned int simulation, char *input,
                     GMappedFile * stencil);
double optimize_parse (unsigned int simulation, unsigned int experiment);
double optimize_norm_euclidian (unsigned int simulation);
double optimize_norm_maximum (unsigned int simulation);
double optimize_norm_p (unsigned int simulation);
double optimize_norm_taxicab (unsigned int simulation);
void optimize_print ();
void optimize_save_variables (unsigned int simulation, double error);
void optimize_best (unsigned int simulation, double value);
void optimize_sequential ();
void *optimize_thread (ParallelData * data);
void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
                     double *error_best);
#if HAVE_MPI
void optimize_synchronise ();
#endif
void optimize_sweep ();
void optimize_MonteCarlo ();
void optimize_orthogonal ();
void optimize_best_climbing (unsigned int simulation, double value);
void optimize_climbing_sequential (unsigned int simulation);
void *optimize_climbing_thread (ParallelData * data);
double optimize_estimate_climbing_random (unsigned int variable,
                                          unsigned int estimate);
double optimize_estimate_climbing_coordinates (unsigned int variable,
                                               unsigned int estimate);
void optimize_step_climbing (unsigned int simulation);
void optimize_climbing ();
double optimize_genetic_objective (Entity * entity);
void optimize_genetic ();
void optimize_save_old ();
void optimize_merge_old ();
void optimize_refine ();
void optimize_step ();
void optimize_iterate ();
void optimize_free ();
void optimize_open ();

#endif
```

4.23 utils.c File Reference

Source file to define some useful functions.

```
#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include <gtk/gtk.h>
#include "utils.h"
```

Include dependency graph for utils.c:



Functions

- void show_message (char ∗title, char ∗msg, int type)
- void show_error (char ∗msg)
- int xml_node_get_int (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- unsigned int xml_node_get_uint (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- unsigned int xml_node_get_uint_with_default (xmlNode ∗node, const xmlChar ∗prop, unsigned int default←_value, int ∗error_code)
- double xml_node_get_float (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- double xml_node_get_float_with_default (xmlNode ∗node, const xmlChar ∗prop, double default_value, int ∗error_code)
- void xml_node_set_int (xmlNode ∗node, const xmlChar ∗prop, int value)
- void xml_node_set_uint (xmlNode ∗node, const xmlChar ∗prop, unsigned int value)
- void xml_node_set_float (xmlNode ∗node, const xmlChar ∗prop, double value)
- int json_object_get_int (JsonObject ∗object, const char ∗prop, int ∗error_code)
- unsigned int json_object_get_uint (JsonObject ∗object, const char ∗prop, int ∗error_code)
- unsigned int json_object_get_uint_with_default (JsonObject ∗object, const char ∗prop, unsigned int default←_value, int ∗error_code)
- double json_object_get_float (JsonObject ∗object, const char ∗prop, int ∗error_code)
- double json_object_get_float_with_default (JsonObject ∗object, const char ∗prop, double default_value, int ∗error_code)
- void json_object_set_int (JsonObject ∗object, const char ∗prop, int value)
- void json_object_set_uint (JsonObject ∗object, const char ∗prop, unsigned int value)
- void json_object_set_float (JsonObject ∗object, const char ∗prop, double value)
- int cores_number ()
- void process_pending ()
- unsigned int gtk_array_get_active (GtkRadioButton ∗array[ ], unsigned int n)

Variables

- GtkWindow ∗ main_window

  Main GtkWindow.
- char ∗ error_message

  Error message.
- void(∗ show_pending)() = NULL

  Pointer to the function to show pending events.

4.23.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file utils.c.

4.23.2 Function Documentation

4.23.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line 440 of file utils.c.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

4.23.2.2 gtk_array_get_active()

```
unsigned int gtk_array_get_active (
              GtkRadioButton * array[ ],
              unsigned int n )
```

Function to get the active GtkRadioButton.

Returns

Active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Definition at line 469 of file utils.c.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

4.23.2.3 json_object_get_float()

```
double json_object_get_float (
              JsonObject * object,
              const char * prop,
              int * error_code )
```

Function to get a floating point number of a JSON object property.

Returns

Floating point number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 350 of file utils.c.

```
{
  const char *buffer;
  double x = 0.;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return x;
}
```

4.23.2.4 json_object_get_float_with_default()

```
double json_object_get_float_with_default (
             JsonObject * object,
             const char * prop,
             double default_value,
             int * error_code )
```

Function to get a floating point number of a JSON object property with a default value.

Returns

Floating point number value.

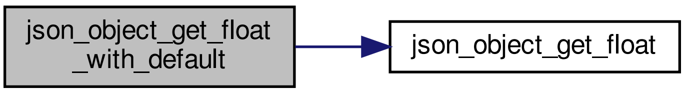
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 376 of file utils.c.

```
{
  double x;
  if (json_object_get_member (object, prop))
    x = json_object_get_float (object, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



4.23.2.5 json_object_get_int()

```
int json_object_get_int (
             JsonObject * object,
             const char * prop,
             int * error_code )
```

Function to get an integer number of a JSON object property.

Returns

Integer number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 276 of file utils.c.

```
{
  const char *buffer;
  int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return i;
}
```

4.23.2.6 json_object_get_uint()

```
unsigned int json_object_get_uint (
              JsonObject * object,
              const char * prop,
              int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Returns

Unsigned integer number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 301 of file utils.c.

```
{
  const char *buffer;
  unsigned int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
```

```
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return i;
}
```

4.23.2.7 json_object_get_uint_with_default()

```
unsigned int json_object_get_uint_with_default (
              JsonObject * object,
              const char * prop,
              unsigned int default_value,
              int * error_code )
```

Function to get an unsigned integer number of a JSON object property with a default value.

Returns

Unsigned integer number value.

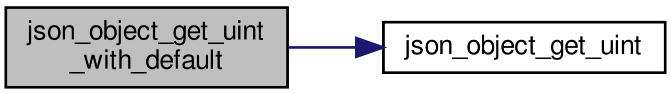
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 327 of file utils.c.

```
{
  unsigned int i;
  if (json_object_get_member (object, prop))
    i = json_object_get_uint (object, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

Here is the call graph for this function:



4.23.2.8 json_object_set_float()

```
void json_object_set_float (
            JsonObject * object,
            const char * prop,
            double value )
```

Function to set a floating point number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Floating point number value.

Definition at line 425 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%.14lg", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.23.2.9 json_object_set_int()

```
void json_object_set_int (
            JsonObject * object,
            const char * prop,
            int value )
```

Function to set an integer number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Integer number value.

Definition at line 398 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%d", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.23.2.10 json_object_set_uint()

```
void json_object_set_uint (
            JsonObject * object,
            const char * prop,
            unsigned int value )
```

Function to set an unsigned integer number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Unsigned integer number value.

Definition at line 411 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%u", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.23.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 457 of file utils.c.

```
{
  while (gtk_events_pending ())
    gtk_main_iteration ();
}
```

4.23.2.12 show_error()

```
void show_error (
            char * msg )
```

Function to show a dialog with an error message.

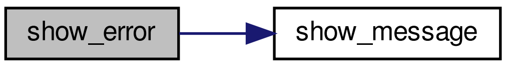
Parameters

msg	Error message.

Definition at line 101 of file utils.c.

```
{
  show_message (_("ERROR!"), msg, ERROR_TYPE);
}
```

Here is the call graph for this function:



4.23.2.13 show_message()

```
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 66 of file utils.c.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *)
    gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
                            (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}
```

4.23.2.14 xml_node_get_float()

```
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.

Returns

Floating point number value.

Parameters

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

Definition at line 188 of file utils.c.

```
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}
```

4.23.2.15 xml_node_get_float_with_default()

```
double xml_node_get_float_with_default (
             xmlNode * node,
             const xmlChar * prop,
             double default_value,
             int * error_code )
```

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

Returns

Floating point number value.

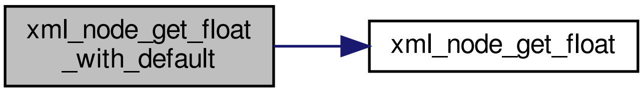
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 215 of file utils.c.

```
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



4.23.2.16 xml_node_get_int()

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

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

Returns

Integer number value.

Parameters

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

Definition at line 112 of file utils.c.

```
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

4.23.2.17 xml_node_get_uint()

```
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.

Returns

Unsigned integer number value.

Parameters

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

Definition at line 138 of file utils.c.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

4.23.2.18 xml_node_get_uint_with_default()

```
unsigned int xml_node_get_uint_with_default (
              xmlNode * node,
              const xmlChar * prop,
              unsigned int default_value,
              int * error_code )
```

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

Returns

Unsigned integer number value.

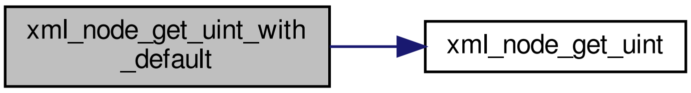
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 165 of file utils.c.

```
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

Here is the call graph for this function:



4.23.2.19 xml_node_set_float()

```
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 261 of file utils.c.

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

4.23.2.20 xml_node_set_int()

```
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 235 of file utils.c.

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

4.23.2.21 xml_node_set_uint()

```
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 248 of file utils.c.

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

4.24 utils.c

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <unistd.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#ifdef G_OS_WIN32
#include <windows.h>
#endif
#if HAVE_GTK
#include <gtk/gtk.h>
#endif
#include "utils.h"

#if HAVE_GTK
GtkWindow *main_window;
#endif

char *error_message;
void (*show_pending) () = NULL;

void
show_message (char *title,
              char *msg,
              int type
#if !HAVE_GTK
```

```
               __attribute__ ((unused))
#endif
  )
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *)
    gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
                            (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}

void
show_error (char *msg)
{
  show_message (_("ERROR!"), msg, ERROR_TYPE);
}

int
xml_node_get_int (xmlNode * node,
                  const xmlChar * prop,
                  int *error_code)
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint (xmlNode * node,
                   const xmlChar * prop,
                   int *error_code)
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}

unsigned int
xml_node_get_uint_with_default (xmlNode * node,
                                const xmlChar * prop,
                                unsigned int default_value,
                                int *error_code)
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}

double
xml_node_get_float (xmlNode * node,
                    const xmlChar * prop,
                    int *error_code)
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}

double
xml_node_get_float_with_default (xmlNode * node,
                                 const xmlChar * prop,
                                 double default_value,
                                 int *error_code)
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}

void
xml_node_set_int (xmlNode * node,
                  const xmlChar * prop,
                  int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%d", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_uint (xmlNode * node,
                   const xmlChar * prop,
                   unsigned int value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%u", value);
  xmlSetProp (node, prop, buffer);
}

void
xml_node_set_float (xmlNode * node,
                    const xmlChar * prop,
                    double value)
{
  xmlChar buffer[64];
  snprintf ((char *) buffer, 64, "%.14lg", value);
  xmlSetProp (node, prop, buffer);
}

int
json_object_get_int (JsonObject * object,
                     const char *prop,
                     int *error_code)
{
  const char *buffer;
  int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%d", &i) != 1)
        *error_code = 2;
      else
```

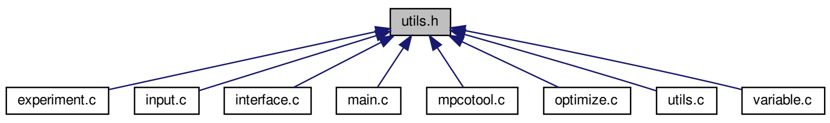
```
        *error_code = 0;
    }
  return i;
}

unsigned int
json_object_get_uint (JsonObject * object,
                      const char *prop,
                      int *error_code)
{
  const char *buffer;
  unsigned int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return i;
}

unsigned int
json_object_get_uint_with_default (JsonObject * object,
                                   const char *prop,
                                   unsigned int default_value,
                                   int *error_code)
{
  unsigned int i;
  if (json_object_get_member (object, prop))
    i = json_object_get_uint (object, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}

double
json_object_get_float (JsonObject * object,
                       const char *prop,
                       int *error_code)
{
  const char *buffer;
  double x = 0.;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return x;
}

double
json_object_get_float_with_default (JsonObject * object,
                                    const char *prop,
                                    double default_value,
                                    int *error_code)
{
  double x;
  if (json_object_get_member (object, prop))
    x = json_object_get_float (object, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}

void
json_object_set_int (JsonObject * object,
                     const char *prop,
                     int value)
{
  char buffer[64];
  snprintf (buffer, 64, "%d", value);
  json_object_set_string_member (object, prop, buffer);
```

```
}

void
json_object_set_uint (JsonObject * object,
                      const char *prop,
                      unsigned int value)
{
  char buffer[64];
  snprintf (buffer, 64, "%u", value);
  json_object_set_string_member (object, prop, buffer);
}

void
json_object_set_float (JsonObject * object,
                       const char *prop,
                       double value)
{
  char buffer[64];
  snprintf (buffer, 64, "%.14lg", value);
  json_object_set_string_member (object, prop, buffer);
}

int
cores_number ()
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}

#if HAVE_GTK

void
process_pending ()
{
  while (gtk_events_pending ())
    gtk_main_iteration ();
}

unsigned int
gtk_array_get_active (GtkRadioButton * array[],
                      unsigned int n)
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}

#endif
```

4.25 utils.h File Reference

Header file to define some useful functions.

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



Macros

- #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.

Functions

- void show_message (char ∗title, char ∗msg, int type)
- void show_error (char ∗msg)
- int xml_node_get_int (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- unsigned int xml_node_get_uint (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- unsigned int xml_node_get_uint_with_default (xmlNode ∗node, const xmlChar ∗prop, unsigned int default←_value, int ∗error_code)
- double xml_node_get_float (xmlNode ∗node, const xmlChar ∗prop, int ∗error_code)
- double xml_node_get_float_with_default (xmlNode ∗node, const xmlChar ∗prop, double default_value, int ∗error_code)
- void xml_node_set_int (xmlNode ∗node, const xmlChar ∗prop, int value)
- void xml_node_set_uint (xmlNode ∗node, const xmlChar ∗prop, unsigned int value)
- void xml_node_set_float (xmlNode ∗node, const xmlChar ∗prop, double value)
- int json_object_get_int (JsonObject ∗object, const char ∗prop, int ∗error_code)
- unsigned int json_object_get_uint (JsonObject ∗object, const char ∗prop, int ∗error_code)
- unsigned int json_object_get_uint_with_default (JsonObject ∗object, const char ∗prop, unsigned int default←_value, int ∗error_code)
- double json_object_get_float (JsonObject ∗object, const char ∗prop, int ∗error_code)
- double json_object_get_float_with_default (JsonObject ∗object, const char ∗prop, double default_value, int ∗error_code)
- void json_object_set_int (JsonObject ∗object, const char ∗prop, int value)
- void json_object_set_uint (JsonObject ∗object, const char ∗prop, unsigned int value)
- void json_object_set_float (JsonObject ∗object, const char ∗prop, double value)
- int cores_number ()
- void process_pending ()
- unsigned int gtk_array_get_active (GtkRadioButton ∗array[ ], unsigned int n)

Variables

- GtkWindow ∗ main_window

  Main GtkWindow.
- char ∗ error_message

  Error message.
- void(∗ show_pending)()

  Pointer to the function to show pending events.

4.25.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file utils.h.

4.25.2 Function Documentation

4.25.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line 440 of file utils.c.

```
{
#ifdef G_OS_WIN32
  SYSTEM_INFO sysinfo;
  GetSystemInfo (&sysinfo);
  return sysinfo.dwNumberOfProcessors;
#else
  return (int) sysconf (_SC_NPROCESSORS_ONLN);
#endif
}
```

4.25.2.2 gtk_array_get_active()

```
unsigned int gtk_array_get_active (
            GtkRadioButton * array[],
            unsigned int n )
```

Function to get the active GtkRadioButton.

Returns

Active GtkRadioButton.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Definition at line 469 of file utils.c.

```
{
  unsigned int i;
  for (i = 0; i < n; ++i)
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
      break;
  return i;
}
```

4.25.2.3 json_object_get_float()

```
double json_object_get_float (
            JsonObject * object,
            const char * prop,
            int * error_code )
```

Function to get a floating point number of a JSON object property.

Returns

Floating point number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 350 of file utils.c.

```
{
  const char *buffer;
  double x = 0.;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return x;
}
```

4.25.2.4 json_object_get_float_with_default()

```
double json_object_get_float_with_default (
            JsonObject * object,
            const char * prop,
            double default_value,
            int * error_code )
```

Function to get a floating point number of a JSON object property with a default value.

Returns

Floating point number value.

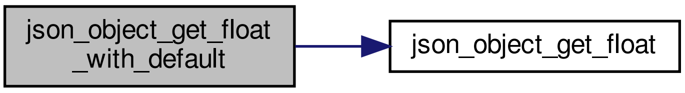
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 376 of file utils.c.

```
{
  double x;
  if (json_object_get_member (object, prop))
    x = json_object_get_float (object, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



4.25.2.5 json_object_get_int()

```
int json_object_get_int (
              JsonObject * object,
              const char * prop,
              int * error_code )
```

Function to get an integer number of a JSON object property.

Returns

Integer number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 276 of file utils.c.

```
{
  const char *buffer;
  int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return i;
}
```

4.25.2.6 json_object_get_uint()

```
unsigned int json_object_get_uint (
               JsonObject * object,
               const char * prop,
               int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Returns

Unsigned integer number value.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Definition at line 301 of file utils.c.

```
{
  const char *buffer;
  unsigned int i = 0;
  buffer = json_object_get_string_member (object, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf (buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
    }
  return i;
}
```

4.25.2.7 json_object_get_uint_with_default()

```
unsigned int json_object_get_uint_with_default (
            JsonObject * object,
            const char * prop,
            unsigned int default_value,
            int * error_code )
```

Function to get an unsigned integer number of a JSON object property with a default value.

Returns

Unsigned integer number value.

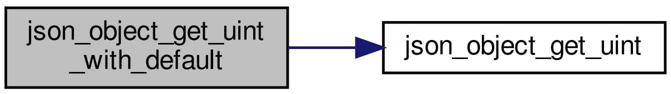
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 327 of file utils.c.

```
{
  unsigned int i;
  if (json_object_get_member (object, prop))
    i = json_object_get_uint (object, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

Here is the call graph for this function:



4.25.2.8 json_object_set_float()

```
void json_object_set_float (
            JsonObject * object,
            const char * prop,
            double value )
```

Function to set a floating point number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Floating point number value.

Definition at line 425 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%.14lg", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.25.2.9 json_object_set_int()

```
void json_object_set_int (
            JsonObject * object,
            const char * prop,
            int value )
```

Function to set an integer number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Integer number value.

Definition at line 398 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%d", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.25.2.10 json_object_set_uint()

```
void json_object_set_uint (
            JsonObject * object,
            const char * prop,
            unsigned int value )
```

Function to set an unsigned integer number in a JSON object property.

Parameters

object	JSON object.
prop	JSON property.
value	Unsigned integer number value.

Definition at line 411 of file utils.c.

```
{
  char buffer[64];
  snprintf (buffer, 64, "%u", value);
  json_object_set_string_member (object, prop, buffer);
}
```

4.25.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 457 of file utils.c.

```
{
  while (gtk_events_pending ())
    gtk_main_iteration ();
}
```

4.25.2.12 show_error()

```
void show_error (
            char * msg )
```

Function to show a dialog with an error message.

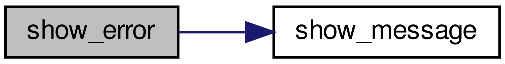
Parameters

msg	Error message.

Definition at line 101 of file utils.c.

```
{
  show_message (_("ERROR!"), msg, ERROR_TYPE);
}
```

Here is the call graph for this function:



4.25.2.13 show_message()

```
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 66 of file utils.c.

```
{
#if HAVE_GTK
  GtkMessageDialog *dlg;

  // Creating the dialog
  dlg = (GtkMessageDialog *)
    gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
                            (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);

  // Setting the dialog title
  gtk_window_set_title (GTK_WINDOW (dlg), title);

  // Showing the dialog and waiting response
  gtk_dialog_run (GTK_DIALOG (dlg));

  // Closing and freeing memory
  gtk_widget_destroy (GTK_WIDGET (dlg));

#else
  printf ("%s: %s\n", title, msg);
#endif
}
```

4.25.2.14 xml_node_get_float()

```
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.

Returns

Floating point number value.

Parameters

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

Definition at line 188 of file utils.c.

```
{
  double x = 0.;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%lf", &x) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return x;
}
```

4.25.2.15 xml_node_get_float_with_default()

```
double xml_node_get_float_with_default (
            xmlNode * node,
            const xmlChar * prop,
            double default_value,
            int * error_code )
```

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

Returns

Floating point number value.

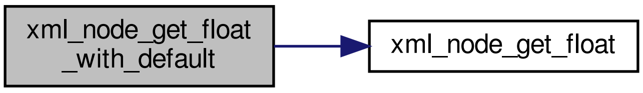
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 215 of file utils.c.

```
{
  double x;
  if (xmlHasProp (node, prop))
    x = xml_node_get_float (node, prop, error_code);
  else
    {
      x = default_value;
      *error_code = 0;
    }
  return x;
}
```

Here is the call graph for this function:



4.25.2.16 xml_node_get_int()

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

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

Returns

Integer number value.

Parameters

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

Definition at line 112 of file utils.c.

```
{
  int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%d", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

4.25.2.17 xml_node_get_uint()

```
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.

Returns

Unsigned integer number value.

Parameters

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

Definition at line 138 of file utils.c.

```
{
  unsigned int i = 0;
  xmlChar *buffer;
  buffer = xmlGetProp (node, prop);
  if (!buffer)
    *error_code = 1;
  else
    {
      if (sscanf ((char *) buffer, "%u", &i) != 1)
        *error_code = 2;
      else
        *error_code = 0;
      xmlFree (buffer);
    }
  return i;
}
```

4.25.2.18 xml_node_get_uint_with_default()

```
unsigned int xml_node_get_uint_with_default (
                xmlNode * node,
                const xmlChar * prop,
                unsigned int default_value,
                int * error_code )
```

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

Returns

Unsigned integer number value.

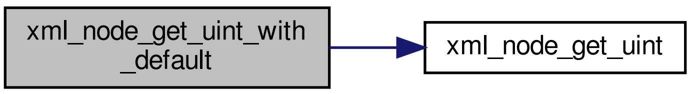
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 165 of file utils.c.

```
{
  unsigned int i;
  if (xmlHasProp (node, prop))
    i = xml_node_get_uint (node, prop, error_code);
  else
    {
      i = default_value;
      *error_code = 0;
    }
  return i;
}
```

Here is the call graph for this function:



4.25.2.19 xml_node_set_float()

```
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 261 of file utils.c.

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

4.25.2.20 xml_node_set_int()

```
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 235 of file utils.c.

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

4.25.2.21 xml_node_set_uint()

```
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 248 of file utils.c.

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

4.26 utils.h

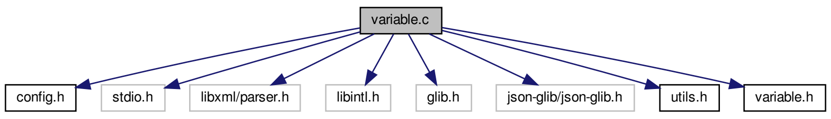
```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef UTILS__H
#define UTILS__H 1

#if HAVE_GTK
#define ERROR_TYPE GTK_MESSAGE_ERROR
#define INFO_TYPE GTK_MESSAGE_INFO
extern GtkWindow *main_window;
#else
#define ERROR_TYPE 0
#define INFO_TYPE 0
#endif

extern char *error_message;
extern void (*show_pending) ();

// Public functions
void show_message (char *title, char *msg, int type);
void show_error (char *msg);
int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
                                int *error_code);
unsigned int xml_node_get_uint_with_default (xmlNode * node,
                                             const xmlChar * prop,
                                             unsigned int default_value,
                                             int *error_code);
double xml_node_get_float (xmlNode * node, const xmlChar * prop,
                           int *error_code);
double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
,
                                        double default_value, int *error_code);
void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
                        unsigned int value);
void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
int json_object_get_int (JsonObject * object, const char *prop,
                         int *error_code);
unsigned int json_object_get_uint (JsonObject * object, const char *prop,
                                   int *error_code);
unsigned int json_object_get_uint_with_default (JsonObject * object,
                                                const char *prop,
                                                unsigned int default_value,
                                                int *error_code);
double json_object_get_float (JsonObject * object, const char *prop,
                              int *error_code);
double json_object_get_float_with_default (JsonObject * object,
                                           const char *prop,
                                           double default_value,
                                           int *error_code);
void json_object_set_int (JsonObject * object, const char *prop, int value);
void json_object_set_uint (JsonObject * object, const char *prop,
                           unsigned int value);
void json_object_set_float (JsonObject * object, const char *prop,
                            double value);
int cores_number ();
#if HAVE_GTK
void process_pending ();
unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
#endif

#endif
```

4.27 variable.c File Reference

Source file to define the variable data.

```
#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "variable.h"
```

Include dependency graph for variable.c:



Macros

- #define DEBUG_VARIABLE 0

  Macro to debug variable functions.

Functions

- void variable_new (Variable ∗variable)
- void variable_free (Variable ∗variable, unsigned int type)
- void variable_error (Variable ∗variable, char ∗message)
- int variable_open_xml (Variable ∗variable, xmlNode ∗node, unsigned int algorithm, unsigned int nsteps)
- int variable_open_json (Variable ∗variable, JsonNode ∗node, unsigned int algorithm, unsigned int nsteps)

Variables

- const char ∗ format [NPRECISIONS]

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

    Array of variable precisions.

4.27.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file variable.c.

4.27.2 Function Documentation

4.27.2.1 variable_error()

```
void variable_error (
            Variable ∗ variable,
            char ∗ message )
```

Function to print a message error opening an Variable struct.

Parameters

variable	Variable struct.
message	Error message.

Definition at line 100 of file variable.c.

```
{
  char buffer[64];
  if (!variable->name)
    snprintf (buffer, 64, "%s: %s", _("Variable"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
  error_message = g_strdup (buffer);
}
```

4.27.2.2 variable_free()

```
void variable_free (
            Variable * variable,
            unsigned int type )
```

Function to free the memory of a Variable struct.

Parameters

variable	Variable struct.
type	Type of input file.

Definition at line 79 of file variable.c.

```
{
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    xmlFree (variable->name);
  else
    g_free (variable->name);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: end\n");
#endif
}
```

4.27.2.3 variable_new()

```
void variable_new (
            Variable * variable )
```

Function to create a new Variable struct.

Parameters

variable	Variable struct.

Definition at line 64 of file variable.c.

```
{
```

```
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: start\n");
#endif
  variable->name = NULL;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: end\n");
#endif
}
```

4.27.2.4 variable_open_json()

```
int variable_open_json (
            Variable * variable,
            JsonNode * node,
            unsigned int algorithm,
            unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

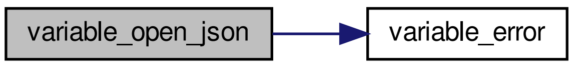
Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

Definition at line 279 of file variable.c.

```
{
  JsonObject *object;
  const char *label;
  int error_code;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: start\n");
#endif
  object = json_node_get_object (node);
  label = json_object_get_string_member (object, LABEL_NAME);
  if (!label)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  variable->name = g_strdup (label);
  if (json_object_get_member (object, LABEL_MINIMUM))
    {
      variable->rangemin
        = json_object_get_float (object, LABEL_MINIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs
        = json_object_get_float_with_default (object,
LABEL_ABSOLUTE_MINIMUM,
                                              -G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (json_object_get_member (object, LABEL_MAXIMUM))
    {
      variable->rangemax
        = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs
        = json_object_get_float_with_default (object,
LABEL_ABSOLUTE_MAXIMUM,
                                              G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = json_object_get_uint_with_default (object,
LABEL_PRECISION,
                                         DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
    {
      if (json_object_get_member (object, LABEL_NSWEEPS))
        {
          variable->nsweeps
            = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
    {
      // Obtaining bits representing each variable
      if (json_object_get_member (object, LABEL_NBITS))
        {
          variable->nbits
            = json_object_get_uint (object, LABEL_NBITS, &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_JSON);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.27.2.5 variable_open_xml()

```
int variable_open_xml (
            Variable * variable,
            xmlNode * node,
            unsigned int algorithm,
            unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

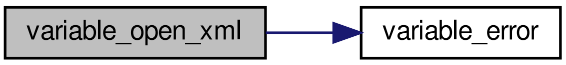
Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

Definition at line 119 of file variable.c.

```
{
  int error_code;

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: start\n");
#endif

  variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!variable->name)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
    {
      variable->rangemin
        = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
    {
      variable->rangemax
        = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                      DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
    {
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
        {
          variable->nsweeps
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                 &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
    {
      // Obtaining bits representing each variable
      if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
        {
          variable->nbits
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                 &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step
        = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_XML);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.27.3 Variable Documentation

4.27.3.1 format

```
const char* format[NPRECISIONS]
```

Initial value:

```
= {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
}
```

Array of C-strings with variable formats.

Definition at line 50 of file variable.c.

4.27.3.2 precision

```
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 55 of file variable.c.

4.28 variable.c

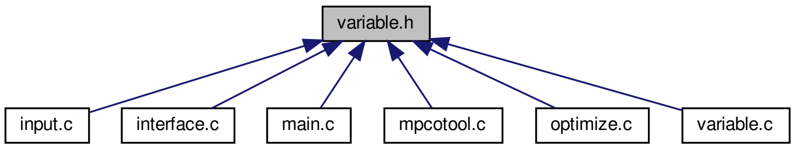
```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    1. Redistributions of source code must retain the above copyright notice,
        this list of conditions and the following disclaimer.

    2. Redistributions in binary form must reproduce the above copyright notice,
        this list of conditions and the following disclaimer in the
        documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```c
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#define _GNU_SOURCE
#include "config.h"
#include <stdio.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "variable.h"

#define DEBUG_VARIABLE 0

const char *format[NPRECISIONS] = {
  "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
  "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
};

const double precision[NPRECISIONS] = {
  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
};

void
variable_new (Variable * variable)
{
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: start\n");
#endif
  variable->name = NULL;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: end\n");
#endif
}

void
variable_free (Variable * variable,
               unsigned int type)
{
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    xmlFree (variable->name);
  else
    g_free (variable->name);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: end\n");
#endif
}

void
variable_error (Variable * variable,
                char *message)
{
  char buffer[64];
  if (!variable->name)
    snprintf (buffer, 64, "%s: %s", _("Variable"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
  error_message = g_strdup (buffer);
}

int
variable_open_xml (Variable * variable,
                   xmlNode * node,
                   unsigned int algorithm,
                   unsigned int nsteps)
{
  int error_code;

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: start\n");
#endif

  variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!variable->name)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
    {
      variable->rangemin
        = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
    {
      variable->rangemax
        = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
                                      DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
    {
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
        {
          variable->nsweeps
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                 &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
    {
      // Obtaining bits representing each variable
      if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
        {
          variable->nbits
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                 &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step
        = xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_XML);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 0;
}

int
variable_open_json (Variable * variable,
                    JsonNode * node,
                    unsigned int algorithm,
                    unsigned int nsteps)
{
  JsonObject *object;
  const char *label;
  int error_code;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: start\n");
#endif
  object = json_node_get_object (node);
  label = json_object_get_string_member (object, LABEL_NAME);
  if (!label)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  variable->name = g_strdup (label);
  if (json_object_get_member (object, LABEL_MINIMUM))
    {
      variable->rangemin
        = json_object_get_float (object, LABEL_MINIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs
        = json_object_get_float_with_default (object, LABEL_ABSOLUTE_MINIMUM,
                                              -G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (json_object_get_member (object, LABEL_MAXIMUM))
    {
      variable->rangemax
        = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs
        = json_object_get_float_with_default (object,
LABEL_ABSOLUTE_MAXIMUM,
                                              G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = json_object_get_uint_with_default (object,
LABEL_PRECISION,
                                         DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
    {
      if (json_object_get_member (object, LABEL_NSWEEPS))
        {
          variable->nsweeps
            = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
    {
      // Obtaining bits representing each variable
      if (json_object_get_member (object, LABEL_NBITS))
        {
          variable->nbits
            = json_object_get_uint (object, LABEL_NBITS, &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_JSON);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 0;
}
```

4.29 variable.h File Reference

Header file to define the variable data.

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



Data Structures

- struct Variable

  Struct to define the variable data.

Enumerations

- enum Algorithm { ALGORITHM_MONTE_CARLO = 0, ALGORITHM_SWEEP = 1, ALGORITHM_GENETIC = 2, ALGORITHM_ORTHOGONAL = 3 }

  Enum to define the algorithms.

Functions

- void variable_new (Variable ∗variable)
- void variable_free (Variable ∗variable, unsigned int type)
- void variable_error (Variable ∗variable, char ∗message)
- int variable_open_xml (Variable ∗variable, xmlNode ∗node, unsigned int algorithm, unsigned int nsteps)
- int variable_open_json (Variable ∗variable, JsonNode ∗node, unsigned int algorithm, unsigned int nsteps)

Variables

- const char ∗ format [NPRECISIONS]

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

  Array of variable precisions.

4.29.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file variable.h.

4.29.2 Enumeration Type Documentation

4.29.2.1 Algorithm

```
enum Algorithm
```

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO	Monte-Carlo algorithm.
ALGORITHM_SWEEP	Sweep algorithm.
ALGORITHM_GENETIC	Genetic algorithm.
ALGORITHM_ORTHOGONAL	Orthogonal sampling algorithm.

Definition at line 42 of file variable.h.

```
{
  ALGORITHM_MONTE_CARLO = 0,
  ALGORITHM_SWEEP = 1,
  ALGORITHM_GENETIC = 2,
  ALGORITHM_ORTHOGONAL = 3
};
```

4.29.3 Function Documentation

4.29.3.1 variable_error()

```
void variable_error (
            Variable * variable,
            char * message )
```

Function to print a message error opening an Variable struct.

Parameters

variable	Variable struct.
message	Error message.

Definition at line 100 of file variable.c.

```
{
  char buffer[64];
  if (!variable->name)
    snprintf (buffer, 64, "%s: %s", _("Variable"), message);
  else
    snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
  error_message = g_strdup (buffer);
}
```

4.29.3.2 variable_free()

```
void variable_free (
            Variable * variable,
            unsigned int type )
```

Function to free the memory of a Variable struct.

Parameters

variable	Variable struct.
type	Type of input file.

Definition at line 79 of file variable.c.

```
{
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: start\n");
#endif
  if (type == INPUT_TYPE_XML)
    xmlFree (variable->name);
  else
    g_free (variable->name);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_free: end\n");
#endif
}
```

4.29.3.3 variable_new()

```
void variable_new (
            Variable * variable )
```

Function to create a new Variable struct.

Parameters

variable	Variable struct.

Definition at line 64 of file variable.c.

```
{
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: start\n");
#endif
  variable->name = NULL;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_new: end\n");
#endif
}
```

4.29.3.4 variable_open_json()

```
int variable_open_json (
            Variable * variable,
            JsonNode * node,
            unsigned int algorithm,
            unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

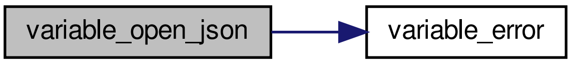
Parameters

variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

Definition at line 279 of file variable.c.

```
{
  JsonObject *object;
  const char *label;
  int error_code;
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: start\n");
#endif
  object = json_node_get_object (node);
  label = json_object_get_string_member (object, LABEL_NAME);
  if (!label)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  variable->name = g_strdup (label);
  if (json_object_get_member (object, LABEL_MINIMUM))
    {
      variable->rangemin
        = json_object_get_float (object, LABEL_MINIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs
        = json_object_get_float_with_default (object,
LABEL_ABSOLUTE_MINIMUM,
                                              -G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (json_object_get_member (object, LABEL_MAXIMUM))
    {
      variable->rangemax
        = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs
        = json_object_get_float_with_default (object,
LABEL_ABSOLUTE_MAXIMUM,
                                              G_MAXDOUBLE, &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = json_object_get_uint_with_default (object,
LABEL_PRECISION,
                                          DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm ==
ALGORITHM_ORTHOGONAL)
    {
      if (json_object_get_member (object, LABEL_NSWEEPS))
        {
          variable->nsweeps
            = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
    {
      // Obtaining bits representing each variable
      if (json_object_get_member (object, LABEL_NBITS))
        {
          variable->nbits
            = json_object_get_uint (object, LABEL_NBITS, &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step = json_object_get_float (object,
LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_JSON);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_json: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.29.3.5 variable_open_xml()

```
int variable_open_xml (
            Variable * variable,
            xmlNode * node,
            unsigned int algorithm,
            unsigned int nsteps )
```

Function to open the variable file.

Returns

1 on success, 0 on error.

Parameters

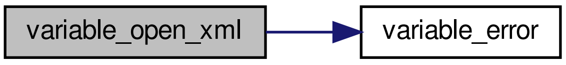
variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the hill climbing method.

Definition at line 119 of file variable.c.

```
{
  int error_code;

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: start\n");
#endif

  variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
  if (!variable->name)
    {
      variable_error (variable, _("no name"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
    {
      variable->rangemin
        = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad minimum"));
          goto exit_on_error;
        }
      variable->rangeminabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute minimum"));
          goto exit_on_error;
        }
      if (variable->rangemin < variable->rangeminabs)
        {
          variable_error (variable, _("minimum range not allowed"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no minimum range"));
      goto exit_on_error;
    }
  if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
    {
      variable->rangemax
        = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
                              &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad maximum"));
          goto exit_on_error;
        }
      variable->rangemaxabs = xml_node_get_float_with_default
        (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
         &error_code);
      if (error_code)
        {
          variable_error (variable, _("bad absolute maximum"));
          goto exit_on_error;
        }
      if (variable->rangemax > variable->rangemaxabs)
        {
          variable_error (variable, _("maximum range not allowed"));
          goto exit_on_error;
        }
      if (variable->rangemax < variable->rangemin)
        {
          variable_error (variable, _("bad range"));
          goto exit_on_error;
        }
    }
  else
    {
      variable_error (variable, _("no maximum range"));
      goto exit_on_error;
    }
  variable->precision
    = xml_node_get_uint_with_default (node, (const xmlChar *) LABEL_PRECISION,
                                      DEFAULT_PRECISION, &error_code);
  if (error_code || variable->precision >= NPRECISIONS)
    {
      variable_error (variable, _("bad precision"));
      goto exit_on_error;
    }
  if (algorithm == ALGORITHM_SWEEP || algorithm == ALGORITHM_ORTHOGONAL)
    {
      if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
        {
          variable->nsweeps
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
                                 &error_code);
          if (error_code || !variable->nsweeps)
            {
              variable_error (variable, _("bad sweeps"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no sweeps number"));
          goto exit_on_error;
        }
#if DEBUG_VARIABLE
      fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
#endif
    }
  if (algorithm == ALGORITHM_GENETIC)
```

```
    {
      // Obtaining bits representing each variable
      if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
        {
          variable->nbits
            = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
                                 &error_code);
          if (error_code || !variable->nbits)
            {
              variable_error (variable, _("invalid bits number"));
              goto exit_on_error;
            }
        }
      else
        {
          variable_error (variable, _("no bits number"));
          goto exit_on_error;
        }
    }
  else if (nsteps)
    {
      variable->step
        = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
      if (error_code || variable->step < 0.)
        {
          variable_error (variable, _("bad step size"));
          goto exit_on_error;
        }
    }

#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 1;
exit_on_error:
  variable_free (variable, INPUT_TYPE_XML);
#if DEBUG_VARIABLE
  fprintf (stderr, "variable_open_xml: end\n");
#endif
  return 0;
}
```

Here is the call graph for this function:



4.30 variable.h

```
/*
MPCOTool:
The Multi-Purposes Calibration and Optimization Tool. A software to perform
calibrations or optimizations of empirical parameters.

AUTHORS: Javier Burguete and Borja Latorre.

Copyright 2012-2018, AUTHORS.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
*/

#ifndef VARIABLE__H
#define VARIABLE__H 1

enum Algorithm
{
  ALGORITHM_MONTE_CARLO = 0,
  ALGORITHM_SWEEP = 1,
  ALGORITHM_GENETIC = 2,
  ALGORITHM_ORTHOGONAL = 3
};

typedef struct
{
  char *name;
  double rangemin;
  double rangemax;
  double rangeminabs;
  double rangemaxabs;
  double step;
  unsigned int precision;
  unsigned int nsweeps;
  unsigned int nbits;
} Variable;

extern const char *format[NPRECISIONS];
extern const double precision[NPRECISIONS];

// Public functions
void variable_new (Variable * variable);
void variable_free (Variable * variable, unsigned int type);
void variable_error (Variable * variable, char *message);
int variable_open_xml (Variable * variable, xmlNode * node,
                       unsigned int algorithm, unsigned int nsteps);
int variable_open_json (Variable * variable, JsonNode * node,
                        unsigned int algorithm, unsigned int nsteps);

#endif
```

Index