MPCOTool

3.4.5

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:

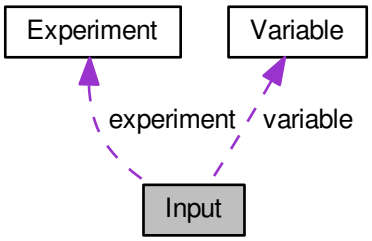
- experiment.h

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 direction search method.
- unsigned int direction

  Method to estimate the direction search.
- unsigned int nestimates

  Number of simulations to estimate the direction search.
- 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 71 of file input.h.

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

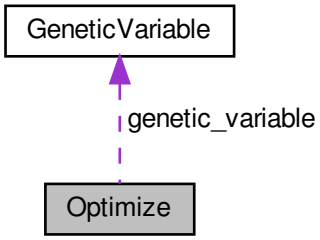
- input.h

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 direction search method step sizes.
- double * direction
  Vector of direction search 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_direction
- 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 direction search method.

- unsigned int nestimates

  Number of simulations to estimate the direction.
- 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_direction

  Beginning simulation number of the task for the direction search method.
- unsigned int nend_direction

  Ending simulation number of the task for the direction search 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_direction

```
unsigned int* Optimize::thread_direction
```

Array of simulation numbers to calculate on the thread for the direction search method.

Definition at line 80 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_direction
  Direction threads number GtkLabel.
- GtkSpinButton * spin_direction
  Direction 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 122 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 67 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
  Direction search 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:

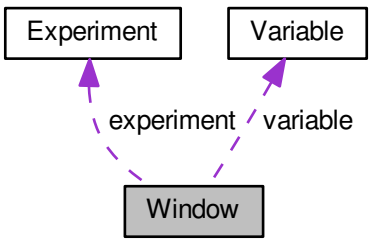
- variable.h

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_direction

  GtkCheckButton to check running the direction search method.
- GtkGrid * grid_direction

  GtkGrid to pack the direction search method widgets.
- GtkRadioButton * button_direction [NDIRECTIONS]

  GtkRadioButtons array to set the direction estimate 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 79 of file interface.h.

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

- interface.h

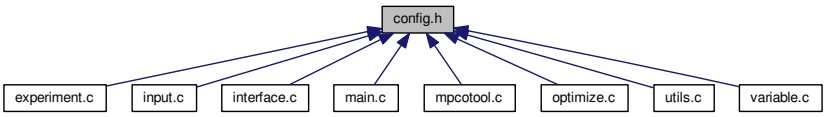
Chapter 4

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 NDIRECTIONS 2

  Number of direction 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_OPTIMIZE "optimize"

  optimize label.
- #define LABEL_COORDINATES "coordinates"

  coordinates label.
- #define LABEL_DIRECTION "direction"

  direction 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_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 2012-2017, 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.

```
00127 {
00128   INPUT_TYPE_XML = 0,
00129   INPUT_TYPE_JSON = 1
00130 };
```

4.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2017, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00039 #ifndef CONFIG__H
00040 #define CONFIG__H 1
00041
00042 // Gettext simplification
00043 #define _(string) (gettext(string))
00044
00045 // Array sizes
00046
00047 #define MAX_NINPUTS 8
00048 #define NALGORITHMS 4
00050 #define NDIRECTIONS 2
00051 #define NNORMS 4
00052 #define NPRECISIONS 15
00053
00054 // Default choices
00055
00056 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00057 #define DEFAULT_RANDOM_SEED 7007
00058 #define DEFAULT_RELAXATION 1.
00059
00060 // Interface labels
00061
00062 #define LOCALE_DIR "locales"
00063 #define PROGRAM_INTERFACE "mpcotool"
00064
00065 // Labels
00066
00067 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00068 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00070 #define LABEL_ADAPTATION "adaptation"
00072 #define LABEL_ALGORITHM "algorithm"
00073 #define LABEL_OPTIMIZE "optimize"
00074 #define LABEL_COORDINATES "coordinates"
00075 #define LABEL_DIRECTION "direction"
00076 #define LABEL_EUCLIDIAN "euclidian"
00077 #define LABEL_EVALUATOR "evaluator"
00078 #define LABEL_EXPERIMENT "experiment"
00079 #define LABEL_EXPERIMENTS "experiments"
00080 #define LABEL_GENETIC "genetic"
00081 #define LABEL_MINIMUM "minimum"
00082 #define LABEL_MAXIMUM "maximum"
00083 #define LABEL_MONTE_CARLO "Monte-Carlo"
00084 #define LABEL_MUTATION "mutation"
```

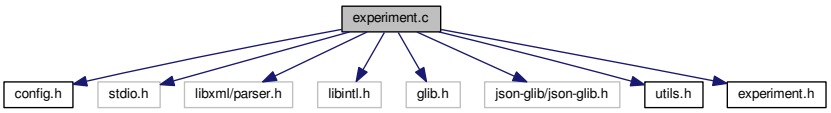
```
00085 #define LABEL_NAME "name"
00086 #define LABEL_NBEST "nbest"
00087 #define LABEL_NBITS "nbits"
00088 #define LABEL_NESTIMATES "nestimates"
00089 #define LABEL_NGENERATIONS "ngenerations"
00090 #define LABEL_NITERATIONS "niterations"
00091 #define LABEL_NORM "norm"
00092 #define LABEL_NPOPULATION "npopulation"
00093 #define LABEL_NSIMULATIONS "nsimulations"
00094 #define LABEL_NSTEPS "nsteps"
00095 #define LABEL_NSWEEPS "nsweeps"
00096 #define LABEL_ORTHOGONAL "orthogonal"
00097 #define LABEL_P "p"
00098 #define LABEL_PRECISION "precision"
00099 #define LABEL_RANDOM "random"
00100 #define LABEL_RELAXATION "relaxation"
00101 #define LABEL_REPRODUCTION "reproduction"
00102 #define LABEL_RESULT_FILE "result_file"
00103 #define LABEL_SIMULATOR "simulator"
00104 #define LABEL_SEED "seed"
00105 #define LABEL_STEP "step"
00106 #define LABEL_SWEEP "sweep"
00107 #define LABEL_TAXICAB "taxicab"
00108 #define LABEL_TEMPLATE1 "template1"
00109 #define LABEL_TEMPLATE2 "template2"
00110 #define LABEL_TEMPLATE3 "template3"
00111 #define LABEL_TEMPLATE4 "template4"
00112 #define LABEL_TEMPLATE5 "template5"
00113 #define LABEL_TEMPLATE6 "template6"
00114 #define LABEL_TEMPLATE7 "template7"
00115 #define LABEL_TEMPLATE8 "template8"
00116 #define LABEL_THRESHOLD "threshold"
00117 #define LABEL_TOLERANCE "tolerance"
00118 #define LABEL_VARIABLE "variable"
00119 #define LABEL_VARIABLES "variables"
00120 #define LABEL_VARIABLES_FILE "variables_file"
00121 #define LABEL_WEIGHT "weight"
00122
00123 // Enumerations
00124
00126 enum INPUT_TYPE
00127 {
00128   INPUT_TYPE_XML = 0,
00129   INPUT_TYPE_JSON = 1
00130 };
00131
00132 #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

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.

```
00111 {
00112   char buffer[64];
00113   if (!experiment->name)
00114     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00117               experiment->name, message);
00118   error_message = g_strdup (buffer);
00119 }
```

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.

```
00082 {
00083   unsigned int i;
00084 #if DEBUG_EXPERIMENT
00085   fprintf (stderr, "experiment_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     {
00089       for (i = 0; i < experiment->ninputs; ++i)
00090         xmlFree (experiment->stencil[i]);
00091       xmlFree (experiment->name);
00092     }
00093   else
00094     {
00095       for (i = 0; i < experiment->ninputs; ++i)
00096         g_free (experiment->stencil[i]);
00097       g_free (experiment->name);
00098     }
00099   experiment->ninputs = 0;
00100 #if DEBUG_EXPERIMENT
00101   fprintf (stderr, "experiment_free: end\n");
00102 #endif
00103 }
```

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.

```
00062 {
00063   unsigned int i;
00064 #if DEBUG_EXPERIMENT
00065   fprintf (stderr, "experiment_new: start\n");
00066 #endif
00067   experiment->name = NULL;
00068   experiment->ninputs = 0;
00069   for (i = 0; i < MAX_NINPUTS; ++i)
00070     experiment->stencil[i] = NULL;
00071 #if DEBUG_EXPERIMENT
00072   fprintf (stderr, "input_new: end\n");
00073 #endif
00074 }
```

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.

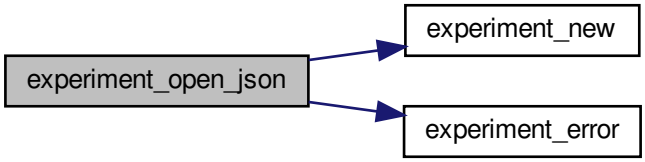
Definition at line 231 of file experiment.c.

```
00235 {
00236   char buffer[64];
00237   JsonObject *object;
00238   const char *name;
00239   int error_code;
```

```
00240   unsigned int i;
00241
00242 #if DEBUG_EXPERIMENT
00243   fprintf (stderr, "experiment_open_json: start\n");
00244 #endif
00245
00246   // Resetting experiment data
00247   experiment_new (experiment);
00248
00249   // Getting JSON object
00250   object = json_node_get_object (node);
00251
00252   // Reading the experimental data
00253   name = json_object_get_string_member (object, LABEL_NAME);
00254   if (!name)
00255     {
00256       experiment_error (experiment, _("no data file name"));
00257       goto exit_on_error;
00258     }
00259   experiment->name = g_strdup (name);
00260 #if DEBUG_EXPERIMENT
00261   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00262 #endif
00263   experiment->weight
00264     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00265                                           &error_code);
00266   if (error_code)
00267     {
00268       experiment_error (experiment, _("bad weight"));
00269       goto exit_on_error;
00270     }
00271 #if DEBUG_EXPERIMENT
00272   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00273 #endif
00274   name = json_object_get_string_member (object, stencil[0]);
00275   if (name)
00276     {
00277 #if DEBUG_EXPERIMENT
00278       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00279                name, stencil[0]);
00280 #endif
00281       ++experiment->ninputs;
00282     }
00283   else
00284     {
00285       experiment_error (experiment, _("no template"));
00286       goto exit_on_error;
00287     }
00288   experiment->stencil[0] = g_strdup (name);
00289   for (i = 1; i < MAX_NINPUTS; ++i)
00290     {
00291 #if DEBUG_EXPERIMENT
00292       fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00293 #endif
00294       if (json_object_get_member (object, stencil[i]))
00295         {
00296           if (ninputs && ninputs <= i)
00297             {
00298               experiment_error (experiment, _("bad templates number"));
00299               goto exit_on_error;
00300             }
00301           name = json_object_get_string_member (object, stencil[i]);
00302 #if DEBUG_EXPERIMENT
00303           fprintf (stderr,
00304                    "experiment_open_json: experiment=%s stencil%u=%s\n",
00305                    experiment->nexperiments, name, stencil[i]);
00306 #endif
00307           experiment->stencil[i] = g_strdup (name);
00308           ++experiment->ninputs;
00309         }
00310       else if (ninputs && ninputs > i)
00311         {
00312           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00313           experiment_error (experiment, buffer);
00314           goto exit_on_error;
00315         }
00316       else
00317         break;
00318     }
00319
00320 #if DEBUG_EXPERIMENT
00321   fprintf (stderr, "experiment_open_json: end\n");
00322 #endif
00323   return 1;
00324
00325 exit_on_error:
```

```
00326   experiment_free (experiment, INPUT_TYPE_JSON);
00327 #if DEBUG_EXPERIMENT
00328   fprintf (stderr, "experiment_open_json: end\n");
00329 #endif
00330   return 0;
00331 }
```

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

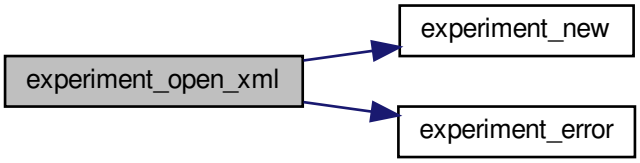
experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 127 of file experiment.c.

```
00131 {
00132   char buffer[64];
00133   int error_code;
00134   unsigned int i;
00135 
00136 #if DEBUG_EXPERIMENT
00137   fprintf (stderr, "experiment_open_xml: start\n");
00138 #endif
00139 
00140   // Resetting experiment data
00141   experiment_new (experiment);
```

```
00142
00143   // Reading the experimental data
00144   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145   if (!experiment->name)
00146     {
00147       experiment_error (experiment, _("no data file name"));
00148       goto exit_on_error;
00149     }
00150 #if DEBUG_EXPERIMENT
00151   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152 #endif
00153   experiment->weight
00154     =
00155     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00156                                      &error_code);
00157   if (error_code)
00158     {
00159       experiment_error (experiment, _("bad weight"));
00160       goto exit_on_error;
00161     }
00162 #if DEBUG_EXPERIMENT
00163   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00164 #endif
00165   experiment->stencil[0]
00166     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00167   if (experiment->stencil[0])
00168     {
00169 #if DEBUG_EXPERIMENT
00170       fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00171                experiment->name, stencil[0]);
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, _("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG_EXPERIMENT
00183       fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00184 #endif
00185       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00186         {
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, _("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->stencil[i]
00193             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00194 #if DEBUG_EXPERIMENT
00195           fprintf (stderr,
00196                    "experiment_open_xml: experiment=%s stencil%u=%s\n",
00197                    experiment->nexperiments, experiment->name,
00198                    experiment->stencil[i]);
00199 #endif
00200           ++experiment->ninputs;
00201         }
00202       else if (ninputs && ninputs > i)
00203         {
00204           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00205           experiment_error (experiment, buffer);
00206           goto exit_on_error;
00207         }
00208       else
00209         break;
00210     }
00211
00212 #if DEBUG_EXPERIMENT
00213   fprintf (stderr, "experiment_open_xml: end\n");
00214 #endif
00215   return 1;
00216
00217 exit_on_error:
00218   experiment_free (experiment, INPUT_TYPE_XML);
00219 #if DEBUG_EXPERIMENT
00220   fprintf (stderr, "experiment_open_xml: end\n");
00221 #endif
00222   return 0;
00223 }
```

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

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "utils.h"
00046 #include "experiment.h"
00047
00048 #define DEBUG_EXPERIMENT 0
00049
00050 const char *stencil[MAX_NINPUTS] = {
00051   LABEL_TEMPLATE1, LABEL_TEMPLATE2,
      LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00052   LABEL_TEMPLATE5, LABEL_TEMPLATE6,
      LABEL_TEMPLATE7, LABEL_TEMPLATE8
00053 };
00054
00056
00060 void
00061 experiment_new (Experiment * experiment)
00062 {
00063   unsigned int i;
00064 #if DEBUG_EXPERIMENT
00065   fprintf (stderr, "experiment_new: start\n");
00066 #endif
00067   experiment->name = NULL;
00068   experiment->ninputs = 0;
00069   for (i = 0; i < MAX_NINPUTS; ++i)
00070     experiment->stencil[i] = NULL;
00071 #if DEBUG_EXPERIMENT
00072   fprintf (stderr, "input_new: end\n");
00073 #endif
00074 }
00075
00079 void
00080 experiment_free (Experiment * experiment,
00081                  unsigned int type)
00082 {
00083   unsigned int i;
00084 #if DEBUG_EXPERIMENT
00085   fprintf (stderr, "experiment_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     {
00089       for (i = 0; i < experiment->ninputs; ++i)
00090         xmlFree (experiment->stencil[i]);
00091       xmlFree (experiment->name);
00092     }
00093   else
00094     {
00095       for (i = 0; i < experiment->ninputs; ++i)
00096         g_free (experiment->stencil[i]);
00097       g_free (experiment->name);
00098     }
00099   experiment->ninputs = 0;
00100 #if DEBUG_EXPERIMENT
00101   fprintf (stderr, "experiment_free: end\n");
00102 #endif
00103 }
00104
00108 void
00109 experiment_error (Experiment * experiment,
00110                   char *message)
00111 {
00112   char buffer[64];
00113   if (!experiment->name)
00114     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00117               experiment->name, message);
00118   error_message = g_strdup (buffer);
00119 }
00120
00126 int
00127 experiment_open_xml (Experiment * experiment,
```

```
00128                     xmlNode * node,
00129                     unsigned int ninputs)
00131 {
00132   char buffer[64];
00133   int error_code;
00134   unsigned int i;
00135
00136 #if DEBUG_EXPERIMENT
00137   fprintf (stderr, "experiment_open_xml: start\n");
00138 #endif
00139
00140   // Resetting experiment data
00141   experiment_new (experiment);
00142
00143   // Reading the experimental data
00144   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145   if (!experiment->name)
00146     {
00147       experiment_error (experiment, _("no data file name"));
00148       goto exit_on_error;
00149     }
00150 #if DEBUG_EXPERIMENT
00151   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152 #endif
00153   experiment->weight
00154     =
00155     xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_WEIGHT, 1.,
00156                                      &error_code);
00157   if (error_code)
00158     {
00159       experiment_error (experiment, _("bad weight"));
00160       goto exit_on_error;
00161     }
00162 #if DEBUG_EXPERIMENT
00163   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00164 #endif
00165   experiment->stencil[0]
00166     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00167   if (experiment->stencil[0])
00168     {
00169 #if DEBUG_EXPERIMENT
00170       fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00171                experiment->name, stencil[0]);
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, _("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG_EXPERIMENT
00183       fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00184 #endif
00185       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00186         {
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, _("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->stencil[i]
00193             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00194 #if DEBUG_EXPERIMENT
00195           fprintf (stderr,
00196                    "experiment_open_xml: experiment=%s stencil%u=%s\n",
00197                    experiment->nexperiments, experiment->name,
00198                    experiment->stencil[i]);
00199 #endif
00200           ++experiment->ninputs;
00201         }
00202       else if (ninputs && ninputs > i)
00203         {
00204           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00205           experiment_error (experiment, buffer);
00206           goto exit_on_error;
00207         }
00208       else
00209         break;
00210     }
00211
00212 #if DEBUG_EXPERIMENT
00213   fprintf (stderr, "experiment_open_xml: end\n");
00214 #endif
```

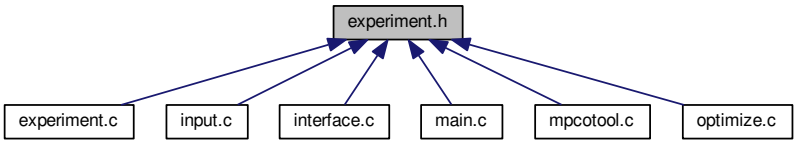
```
00215   return 1;
00216
00217 exit_on_error:
00218   experiment_free (experiment, INPUT_TYPE_XML);
00219 #if DEBUG_EXPERIMENT
00220   fprintf (stderr, "experiment_open_xml: end\n");
00221 #endif
00222   return 0;
00223 }
00224
00230 int
00231 experiment_open_json (Experiment * experiment,
00232                       JsonNode * node,
00233                       unsigned int ninputs)
00235 {
00236   char buffer[64];
00237   JsonObject *object;
00238   const char *name;
00239   int error_code;
00240   unsigned int i;
00241
00242 #if DEBUG_EXPERIMENT
00243   fprintf (stderr, "experiment_open_json: start\n");
00244 #endif
00245
00246   // Resetting experiment data
00247   experiment_new (experiment);
00248
00249   // Getting JSON object
00250   object = json_node_get_object (node);
00251
00252   // Reading the experimental data
00253   name = json_object_get_string_member (object, LABEL_NAME);
00254   if (!name)
00255     {
00256       experiment_error (experiment, _("no data file name"));
00257       goto exit_on_error;
00258     }
00259   experiment->name = g_strdup (name);
00260 #if DEBUG_EXPERIMENT
00261   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00262 #endif
00263   experiment->weight
00264     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00265                                           &error_code);
00266   if (error_code)
00267     {
00268       experiment_error (experiment, _("bad weight"));
00269       goto exit_on_error;
00270     }
00271 #if DEBUG_EXPERIMENT
00272   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00273 #endif
00274   name = json_object_get_string_member (object, stencil[0]);
00275   if (name)
00276     {
00277 #if DEBUG_EXPERIMENT
00278       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00279                name, stencil[0]);
00280 #endif
00281       ++experiment->ninputs;
00282     }
00283   else
00284     {
00285       experiment_error (experiment, _("no template"));
00286       goto exit_on_error;
00287     }
00288   experiment->stencil[0] = g_strdup (name);
00289   for (i = 1; i < MAX_NINPUTS; ++i)
00290     {
00291 #if DEBUG_EXPERIMENT
00292       fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00293 #endif
00294       if (json_object_get_member (object, stencil[i]))
00295         {
00296           if (ninputs && ninputs <= i)
00297             {
00298               experiment_error (experiment, _("bad templates number"));
00299               goto exit_on_error;
00300             }
00301           name = json_object_get_string_member (object, stencil[i]);
00302 #if DEBUG_EXPERIMENT
00303           fprintf (stderr,
00304                    "experiment_open_json: experiment=%s stencil%u=%s\n",
00305                    experiment->nexperiments, name, stencil[i]);
00306 #endif
```

```
00307           experiment->stencil[i] = g_strdup (name);
00308           ++experiment->ninputs;
00309         }
00310       else if (ninputs && ninputs > i)
00311         {
00312           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00313           experiment_error (experiment, buffer);
00314           goto exit_on_error;
00315         }
00316       else
00317         break;
00318     }
00319
00320 #if DEBUG_EXPERIMENT
00321   fprintf (stderr, "experiment_open_json: end\n");
00322 #endif
00323   return 1;
00324
00325 exit_on_error:
00326   experiment_free (experiment, INPUT_TYPE_JSON);
00327 #if DEBUG_EXPERIMENT
00328   fprintf (stderr, "experiment_open_json: end\n");
00329 #endif
00330   return 0;
00331 }
```

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

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.

```
00111 {
00112   char buffer[64];
00113   if (!experiment->name)
00114     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00117               experiment->name, message);
00118   error_message = g_strdup (buffer);
00119 }
```

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.

```
00082 {
00083   unsigned int i;
00084 #if DEBUG_EXPERIMENT
00085   fprintf (stderr, "experiment_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     {
00089       for (i = 0; i < experiment->ninputs; ++i)
00090         xmlFree (experiment->stencil[i]);
00091       xmlFree (experiment->name);
00092     }
00093   else
00094     {
00095       for (i = 0; i < experiment->ninputs; ++i)
00096         g_free (experiment->stencil[i]);
00097       g_free (experiment->name);
00098     }
00099   experiment->ninputs = 0;
00100 #if DEBUG_EXPERIMENT
00101   fprintf (stderr, "experiment_free: end\n");
00102 #endif
00103 }
```

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.

```
00062 {
00063   unsigned int i;
00064 #if DEBUG_EXPERIMENT
00065   fprintf (stderr, "experiment_new: start\n");
00066 #endif
00067   experiment->name = NULL;
00068   experiment->ninputs = 0;
00069   for (i = 0; i < MAX_NINPUTS; ++i)
00070     experiment->stencil[i] = NULL;
00071 #if DEBUG_EXPERIMENT
00072   fprintf (stderr, "input_new: end\n");
00073 #endif
00074 }
```

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

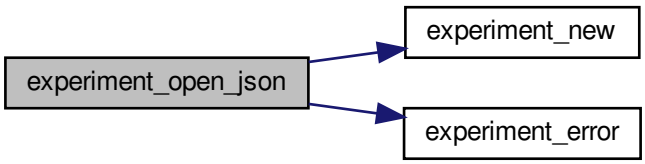
experiment	Experiment struct.
node	JSON node.
ninputs	Number of the simulator input files.

Definition at line 231 of file experiment.c.

```
00235 {
00236   char buffer[64];
00237   JsonObject *object;
00238   const char *name;
00239   int error_code;
00240   unsigned int i;
00241
00242 #if DEBUG_EXPERIMENT
00243   fprintf (stderr, "experiment_open_json: start\n");
00244 #endif
00245
00246   // Resetting experiment data
00247   experiment_new (experiment);
00248
00249   // Getting JSON object
00250   object = json_node_get_object (node);
00251
00252   // Reading the experimental data
00253   name = json_object_get_string_member (object, LABEL_NAME);
00254   if (!name)
00255     {
00256       experiment_error (experiment, _("no data file name"));
00257       goto exit_on_error;
00258     }
00259   experiment->name = g_strdup (name);
00260 #if DEBUG_EXPERIMENT
00261   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00262 #endif
00263   experiment->weight
00264     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00265                                           &error_code);
00266   if (error_code)
00267     {
00268       experiment_error (experiment, _("bad weight"));
00269       goto exit_on_error;
00270     }
00271 #if DEBUG_EXPERIMENT
00272   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00273 #endif
00274   name = json_object_get_string_member (object, stencil[0]);
00275   if (name)
00276     {
00277 #if DEBUG_EXPERIMENT
00278       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00279                name, stencil[0]);
00280 #endif
00281       ++experiment->ninputs;
00282     }
00283   else
```

```
00284     {
00285       experiment_error (experiment, _("no template"));
00286       goto exit_on_error;
00287     }
00288   experiment->stencil[0] = g_strdup (name);
00289   for (i = 1; i < MAX_NINPUTS; ++i)
00290     {
00291 #if DEBUG_EXPERIMENT
00292       fprintf (stderr, "experiment_open_json: stencil%u\n", i + 1);
00293 #endif
00294       if (json_object_get_member (object, stencil[i]))
00295         {
00296           if (ninputs && ninputs <= i)
00297             {
00298               experiment_error (experiment, _("bad templates number"));
00299               goto exit_on_error;
00300             }
00301           name = json_object_get_string_member (object, stencil[i]);
00302 #if DEBUG_EXPERIMENT
00303           fprintf (stderr,
00304                    "experiment_open_json: experiment=%s stencil%u=%s\n",
00305                    experiment->nexperiments, name, stencil[i]);
00306 #endif
00307           experiment->stencil[i] = g_strdup (name);
00308           ++experiment->ninputs;
00309         }
00310       else if (ninputs && ninputs > i)
00311         {
00312           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00313           experiment_error (experiment, buffer);
00314           goto exit_on_error;
00315         }
00316       else
00317         break;
00318     }
00319
00320 #if DEBUG_EXPERIMENT
00321   fprintf (stderr, "experiment_open_json: end\n");
00322 #endif
00323   return 1;
00324
00325 exit_on_error:
00326   experiment_free (experiment, INPUT_TYPE_JSON);
00327 #if DEBUG_EXPERIMENT
00328   fprintf (stderr, "experiment_open_json: end\n");
00329 #endif
00330   return 0;
00331 }
```

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

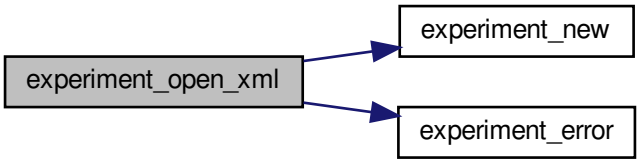
experiment	Experiment struct.
node	XML node.
ninputs	Number of the simulator input files.

Definition at line 127 of file experiment.c.

```
00131 {
00132   char buffer[64];
00133   int error_code;
00134   unsigned int i;
00135 
00136 #if DEBUG_EXPERIMENT
00137   fprintf (stderr, "experiment_open_xml: start\n");
00138 #endif
00139 
00140   // Resetting experiment data
00141   experiment_new (experiment);
00142 
00143   // Reading the experimental data
00144   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145   if (!experiment->name)
00146     {
00147       experiment_error (experiment, _("no data file name"));
00148       goto exit_on_error;
00149     }
00150 #if DEBUG_EXPERIMENT
00151   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00152 #endif
00153   experiment->weight
00154     =
00155     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00156                                      &error_code);
00157   if (error_code)
00158     {
00159       experiment_error (experiment, _("bad weight"));
00160       goto exit_on_error;
00161     }
00162 #if DEBUG_EXPERIMENT
00163   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00164 #endif
00165   experiment->stencil[0]
00166     = (char *) xmlGetProp (node, (const xmlChar *) stencil[0]);
00167   if (experiment->stencil[0])
00168     {
00169 #if DEBUG_EXPERIMENT
00170       fprintf (stderr, "experiment_open_xml: experiment=%s stencil1=%s\n",
00171                experiment->name, stencil[0]);
00172 #endif
00173       ++experiment->ninputs;
00174     }
00175   else
00176     {
00177       experiment_error (experiment, _("no template"));
00178       goto exit_on_error;
00179     }
00180   for (i = 1; i < MAX_NINPUTS; ++i)
00181     {
00182 #if DEBUG_EXPERIMENT
00183       fprintf (stderr, "experiment_open_xml: stencil%u\n", i + 1);
00184 #endif
00185       if (xmlHasProp (node, (const xmlChar *) stencil[i]))
00186         {
```

```
00187           if (ninputs && ninputs <= i)
00188             {
00189               experiment_error (experiment, _("bad templates number"));
00190               goto exit_on_error;
00191             }
00192           experiment->stencil[i]
00193             = (char *) xmlGetProp (node, (const xmlChar *) stencil[i]);
00194 #if DEBUG_EXPERIMENT
00195           fprintf (stderr,
00196                    "experiment_open_xml: experiment=%s stencil%u=%s\n",
00197                    experiment->nexperiments, experiment->name,
00198                    experiment->stencil[i]);
00199 #endif
00200           ++experiment->ninputs;
00201         }
00202       else if (ninputs && ninputs > i)
00203         {
00204           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00205           experiment_error (experiment, buffer);
00206           goto exit_on_error;
00207         }
00208       else
00209         break;
00210     }
00211
00212 #if DEBUG_EXPERIMENT
00213   fprintf (stderr, "experiment_open_xml: end\n");
00214 #endif
00215   return 1;
00216
00217 exit_on_error:
00218   experiment_free (experiment, INPUT_TYPE_XML);
00219 #if DEBUG_EXPERIMENT
00220   fprintf (stderr, "experiment_open_xml: end\n");
00221 #endif
00222   return 0;
00223 }
```

Here is the call graph for this function:



4.6 experiment.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
```

```
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef EXPERIMENT__H
00039 #define EXPERIMENT__H 1
00040
00045 typedef struct
00046 {
00047   char *name;
00048   char *stencil[MAX_NINPUTS];
00049   double weight;
00050   unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const char *stencil[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_new (Experiment * experiment);
00057 void experiment_free (Experiment * experiment, unsigned int type);
00058 void experiment_error (Experiment * experiment, char *message);
00059 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00060                          unsigned int ninputs);
00061 int experiment_open_json (Experiment * experiment, JsonNode * node,
00062                           unsigned int ninputs);
00063
00064 #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

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.

```
00120 {
00121   char buffer[64];
00122   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123   error_message = g_strdup (buffer);
00124 }
```

4.7.2.2 input_free()

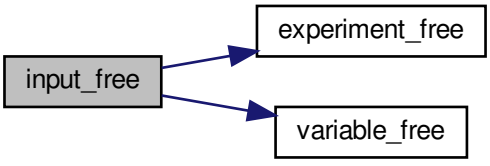
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 81 of file input.c.

```
00082 {
00083   unsigned int i;
00084 #if DEBUG_INPUT
00085   fprintf (stderr, "input_free: start\n");
00086 #endif
00087   g_free (input->name);
00088   g_free (input->directory);
00089   for (i = 0; i < input->nexperiments; ++i)
00090     experiment_free (input->experiment + i, input->
      type);
00091   for (i = 0; i < input->nvariables; ++i)
00092     variable_free (input->variable + i, input->
      type);
00093   g_free (input->experiment);
00094   g_free (input->variable);
00095   if (input->type == INPUT_TYPE_XML)
00096     {
00097       xmlFree (input->evaluator);
00098       xmlFree (input->simulator);
00099       xmlFree (input->result);
00100       xmlFree (input->variables);
00101     }
00102   else
00103     {
00104       g_free (input->evaluator);
00105       g_free (input->simulator);
00106       g_free (input->result);
00107       g_free (input->variables);
00108     }
00109   input->nexperiments = input->nvariables =
      input->nsteps = 0;
00110 #if DEBUG_INPUT
00111   fprintf (stderr, "input_free: end\n");
00112 #endif
00113 }
```

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.

```
00064 {
00065 #if DEBUG_INPUT
00066   fprintf (stderr, "input_new: start\n");
00067 #endif
00068   input->nvariables = input->nexperiments =
      input->nsteps = 0;
00069   input->simulator = input->evaluator = input->
      directory = input->name = NULL;
00070   input->experiment = NULL;
00071   input->variable = NULL;
00072 #if DEBUG_INPUT
00073   fprintf (stderr, "input_new: end\n");
00074 #endif
00075 }
```

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

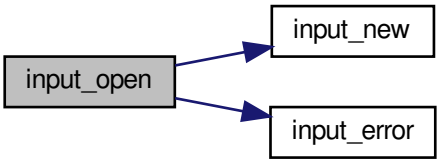
filename	Input data file name.

Definition at line 953 of file input.c.

```
00954 {
00955   xmlDoc *doc;
00956   JsonParser *parser;
00957
00958 #if DEBUG_INPUT
00959   fprintf (stderr, "input_open: start\n");
00960 #endif
00961
00962   // Resetting input data
00963   input_new ();
00964
00965   // Opening input file
00966 #if DEBUG_INPUT
00967   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00968   fprintf (stderr, "input_open: trying XML format\n");
00969 #endif
00970   doc = xmlParseFile (filename);
00971   if (!doc)
00972     {
00973 #if DEBUG_INPUT
```

```
00974       fprintf (stderr, "input_open: trying JSON format\n");
00975 #endif
00976       parser = json_parser_new ();
00977       if (!json_parser_load_from_file (parser, filename, NULL))
00978         {
00979           input_error (_("Unable to parse the input file"));
00980           goto exit_on_error;
00981         }
00982       if (!input_open_json (parser))
00983         goto exit_on_error;
00984     }
00985   else if (!input_open_xml (doc))
00986     goto exit_on_error;
00987
00988   // Getting the working directory
00989   input->directory = g_path_get_dirname (filename);
00990   input->name = g_path_get_basename (filename);
00991
00992 #if DEBUG_INPUT
00993   fprintf (stderr, "input_open: end\n");
00994 #endif
00995   return 1;
00996
00997 exit_on_error:
00998   show_error (error_message);
00999   g_free (error_message);
01000   input_free ();
01001 #if DEBUG_INPUT
01002   fprintf (stderr, "input_open: end\n");
01003 #endif
01004   return 0;
01005 }
```

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 563 of file input.c.

```
00564 {
00565   JsonNode *node, *child;
00566   JsonObject *object;
00567   JsonArray *array;
00568   const char *buffer;
00569   int error_code;
00570   unsigned int i, n;
00571
00572 #if DEBUG_INPUT
00573   fprintf (stderr, "input_open_json: start\n");
00574 #endif
00575
00576   // Resetting input data
00577   input->type = INPUT_TYPE_JSON;
00578
00579   // Getting the root node
00580 #if DEBUG_INPUT
00581   fprintf (stderr, "input_open_json: getting the root node\n");
00582 #endif
00583   node = json_parser_get_root (parser);
00584   object = json_node_get_object (node);
00585
00586   // Getting result and variables file names
00587   if (!input->result)
00588     {
00589       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00590       if (!buffer)
00591         buffer = result_name;
00592       input->result = g_strdup (buffer);
00593     }
00594   else
00595     input->result = g_strdup (result_name);
00596   if (!input->variables)
00597     {
00598       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00599       if (!buffer)
00600         buffer = variables_name;
00601       input->variables = g_strdup (buffer);
00602     }
00603   else
00604     input->variables = g_strdup (variables_name);
00605
00606   // Opening simulator program name
00607   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00608   if (!buffer)
00609     {
00610       input_error (_("Bad simulator program"));
00611       goto exit_on_error;
00612     }
00613   input->simulator = g_strdup (buffer);
00614
00615   // Opening evaluator program name
00616   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00617   if (buffer)
00618     input->evaluator = g_strdup (buffer);
00619
00620   // Obtaining pseudo-random numbers generator seed
00621   input->seed
00622     = json_object_get_uint_with_default (object,
    LABEL_SEED,
00623                                          DEFAULT_RANDOM_SEED, &error_code);
00624   if (error_code)
00625     {
00626       input_error (_("Bad pseudo-random numbers generator seed"));
00627       goto exit_on_error;
00628     }
00629
00630   // Opening algorithm
00631   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00632   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00633     {
00634       input->algorithm = ALGORITHM_MONTE_CARLO;
00635
00636       // Obtaining simulations number
00637       input->nsimulations
00638         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
    );
00639       if (error_code)
00640         {
00641           input_error (_("Bad simulations number"));
00642           goto exit_on_error;
00643         }
00644     }
00645   else if (!strcmp (buffer, LABEL_SWEEP))
```

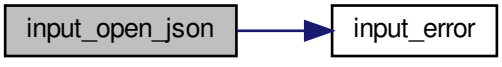
```
00646     input->algorithm = ALGORITHM_SWEEP;
00647   else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00648     input->algorithm = ALGORITHM_ORTHOGONAL;
00649   else if (!strcmp (buffer, LABEL_GENETIC))
00650     {
00651       input->algorithm = ALGORITHM_GENETIC;
00652
00653       // Obtaining population
00654       if (json_object_get_member (object, LABEL_NPOPULATION))
00655         {
00656           input->nsimulations
00657             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00658           if (error_code || input->nsimulations < 3)
00659             {
00660               input_error (_("Invalid population number"));
00661               goto exit_on_error;
00662             }
00663         }
00664       else
00665         {
00666           input_error (_("No population number"));
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining generations
00671       if (json_object_get_member (object, LABEL_NGENERATIONS))
00672         {
00673           input->niterations
00674             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00675           if (error_code || !input->niterations)
00676             {
00677               input_error (_("Invalid generations number"));
00678               goto exit_on_error;
00679             }
00680         }
00681       else
00682         {
00683           input_error (_("No generations number"));
00684           goto exit_on_error;
00685         }
00686
00687       // Obtaining mutation probability
00688       if (json_object_get_member (object, LABEL_MUTATION))
00689         {
00690           input->mutation_ratio
00691             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00692           if (error_code || input->mutation_ratio < 0.
00693               || input->mutation_ratio >= 1.)
00694             {
00695               input_error (_("Invalid mutation probability"));
00696               goto exit_on_error;
00697             }
00698         }
00699       else
00700         {
00701           input_error (_("No mutation probability"));
00702           goto exit_on_error;
00703         }
00704
00705       // Obtaining reproduction probability
00706       if (json_object_get_member (object, LABEL_REPRODUCTION))
00707         {
00708           input->reproduction_ratio
00709             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00710           if (error_code || input->reproduction_ratio < 0.
00711               || input->reproduction_ratio >= 1.0)
00712             {
00713               input_error (_("Invalid reproduction probability"));
00714               goto exit_on_error;
00715             }
00716         }
00717       else
00718         {
00719           input_error (_("No reproduction probability"));
00720           goto exit_on_error;
00721         }
00722
00723       // Obtaining adaptation probability
00724       if (json_object_get_member (object, LABEL_ADAPTATION))
00725         {
00726           input->adaptation_ratio
00727             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
```

```
00728           if (error_code || input->adaptation_ratio < 0.
00729               || input->adaptation_ratio >= 1.)
00730             {
00731               input_error (_("Invalid adaptation probability"));
00732               goto exit_on_error;
00733             }
00734         }
00735       else
00736         {
00737           input_error (_("No adaptation probability"));
00738           goto exit_on_error;
00739         }
00740
00741       // Checking survivals
00742       i = input->mutation_ratio * input->nsimulations;
00743       i += input->reproduction_ratio * input->
      nsimulations;
00744       i += input->adaptation_ratio * input->
      nsimulations;
00745       if (i > input->nsimulations - 2)
00746         {
00747           input_error
00748             (_("No enough survival entities to reproduce the population"));
00749           goto exit_on_error;
00750         }
00751     }
00752   else
00753     {
00754       input_error (_("Unknown algorithm"));
00755       goto exit_on_error;
00756     }
00757
00758   if (input->algorithm == ALGORITHM_MONTE_CARLO
00759       || input->algorithm == ALGORITHM_SWEEP
00760       || input->algorithm == ALGORITHM_ORTHOGONAL)
00761     {
00762
00763       // Obtaining iterations number
00764       input->niterations
00765         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00766       if (error_code == 1)
00767         input->niterations = 1;
00768       else if (error_code)
00769         {
00770           input_error (_("Bad iterations number"));
00771           goto exit_on_error;
00772         }
00773
00774       // Obtaining best number
00775       input->nbest
00776         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00777                                              &error_code);
00778       if (error_code || !input->nbest)
00779         {
00780           input_error (_("Invalid best number"));
00781           goto exit_on_error;
00782         }
00783
00784       // Obtaining tolerance
00785       input->tolerance
00786         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00787                                               &error_code);
00788       if (error_code || input->tolerance < 0.)
00789         {
00790           input_error (_("Invalid tolerance"));
00791           goto exit_on_error;
00792         }
00793
00794       // Getting direction search method parameters
00795       if (json_object_get_member (object, LABEL_NSTEPS))
00796         {
00797           input->nsteps
00798             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00799           if (error_code)
00800             {
00801               input_error (_("Invalid steps number"));
00802               goto exit_on_error;
00803             }
00804           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00805           if (!strcmp (buffer, LABEL_COORDINATES))
00806             input->direction = DIRECTION_METHOD_COORDINATES;
00807           else if (!strcmp (buffer, LABEL_RANDOM))
00808             {
00809               input->direction = DIRECTION_METHOD_RANDOM;
```

```
00810               input->nestimates
00811                 = json_object_get_uint (object,
    LABEL_NESTIMATES, &error_code);
00812               if (error_code || !input->nestimates)
00813                 {
00814                   input_error (_("Invalid estimates number"));
00815                   goto exit_on_error;
00816                 }
00817             }
00818           else
00819             {
00820               input_error
00821                 (_("Unknown method to estimate the direction search"));
00822               goto exit_on_error;
00823             }
00824           input->relaxation
00825             = json_object_get_float_with_default (object,
    LABEL_RELAXATION,
00826                                                   DEFAULT_RELAXATION,
00827                                                   &error_code);
00828           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00829             {
00830               input_error (_("Invalid relaxation parameter"));
00831               goto exit_on_error;
00832             }
00833         }
00834       else
00835         input->nsteps = 0;
00836     }
00837   // Obtaining the threshold
00838   input->threshold
00839     = json_object_get_float_with_default (object,
    LABEL_THRESHOLD, 0.,
00840                                           &error_code);
00841   if (error_code)
00842     {
00843       input_error (_("Invalid threshold"));
00844       goto exit_on_error;
00845     }
00846
00847   // Reading the experimental data
00848   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00849   n = json_array_get_length (array);
00850   input->experiment = (Experiment *) g_malloc (n * sizeof (
    Experiment));
00851   for (i = 0; i < n; ++i)
00852     {
00853 #if DEBUG_INPUT
00854       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00855                input->nexperiments);
00856 #endif
00857       child = json_array_get_element (array, i);
00858       if (!input->nexperiments)
00859         {
00860           if (!experiment_open_json (input->experiment, child, 0))
00861             goto exit_on_error;
00862         }
00863       else
00864         {
00865           if (!experiment_open_json (input->experiment +
    input->nexperiments,
00866                                      child, input->experiment->
    ninputs))
00867             goto exit_on_error;
00868         }
00869       ++input->nexperiments;
00870 #if DEBUG_INPUT
00871       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00872                input->nexperiments);
00873 #endif
00874     }
00875   if (!input->nexperiments)
00876     {
00877       input_error (_("No optimization experiments"));
00878       goto exit_on_error;
00879     }
00880
00881   // Reading the variables data
00882   array = json_object_get_array_member (object, LABEL_VARIABLES);
00883   n = json_array_get_length (array);
00884   input->variable = (Variable *) g_malloc (n * sizeof (
    Variable));
00885   for (i = 0; i < n; ++i)
00886     {
00887 #if DEBUG_INPUT
00888       fprintf (stderr, "input_open_json: nvariables=%u\n", input->
```

```
      nvariables);
00889 #endif
00890       child = json_array_get_element (array, i);
00891       if (!variable_open_json (input->variable +
      input->nvariables, child,
00892                                input->algorithm, input->
      nsteps))
00893         goto exit_on_error;
00894       ++input->nvariables;
00895     }
00896   if (!input->nvariables)
00897     {
00898       input_error (_("No optimization variables"));
00899       goto exit_on_error;
00900     }
00901
00902   // Obtaining the error norm
00903   if (json_object_get_member (object, LABEL_NORM))
00904     {
00905       buffer = json_object_get_string_member (object, LABEL_NORM);
00906       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00907         input->norm = ERROR_NORM_EUCLIDIAN;
00908       else if (!strcmp (buffer, LABEL_MAXIMUM))
00909         input->norm = ERROR_NORM_MAXIMUM;
00910       else if (!strcmp (buffer, LABEL_P))
00911         {
00912           input->norm = ERROR_NORM_P;
00913           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00914           if (!error_code)
00915             {
00916               input_error (_("Bad P parameter"));
00917               goto exit_on_error;
00918             }
00919         }
00920       else if (!strcmp (buffer, LABEL_TAXICAB))
00921         input->norm = ERROR_NORM_TAXICAB;
00922       else
00923         {
00924           input_error (_("Unknown error norm"));
00925           goto exit_on_error;
00926         }
00927     }
00928   else
00929     input->norm = ERROR_NORM_EUCLIDIAN;
00930
00931   // Closing the JSON document
00932   g_object_unref (parser);
00933
00934 #if DEBUG_INPUT
00935   fprintf (stderr, "input_open_json: end\n");
00936 #endif
00937   return 1;
00938
00939 exit_on_error:
00940   g_object_unref (parser);
00941 #if DEBUG_INPUT
00942   fprintf (stderr, "input_open_json: end\n");
00943 #endif
00944   return 0;
00945 }
```

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.

```
00133 {
00134   char buffer2[64];
00135   xmlNode *node, *child;
00136   xmlChar *buffer;
00137   int error_code;
00138   unsigned int i;
00139
00140 #if DEBUG_INPUT
00141   fprintf (stderr, "input_open_xml: start\n");
00142 #endif
00143
00144   // Resetting input data
00145   buffer = NULL;
00146   input->type = INPUT_TYPE_XML;
00147
00148   // Getting the root node
00149 #if DEBUG_INPUT
00150   fprintf (stderr, "input_open_xml: getting the root node\n");
00151 #endif
00152   node = xmlDocGetRootElement (doc);
00153   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155       input_error (_("Bad root XML node"));
00156       goto exit_on_error;
00157     }
00158
00159   // Getting result and variables file names
00160   if (!input->result)
00161     {
00162       input->result =
00163         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164       if (!input->result)
00165         input->result = (char *) xmlStrdup ((const xmlChar *)
      result_name);
00166     }
00167 #if DEBUG_INPUT
00168   fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00169 #endif
00170   if (!input->variables)
00171     {
00172       input->variables =
00173         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00174       if (!input->variables)
00175         input->variables =
00176           (char *) xmlStrdup ((const xmlChar *) variables_name);
00177     }
00178 #if DEBUG_INPUT
00179   fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00180 #endif
00181
00182   // Opening simulator program name
00183   input->simulator =
00184     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00185   if (!input->simulator)
00186     {
00187       input_error (_("Bad simulator program"));
00188       goto exit_on_error;
```

```
00189     }
00190
00191   // Opening evaluator program name
00192   input->evaluator =
00193     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195   // Obtaining pseudo-random numbers generator seed
00196   input->seed
00197     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_SEED,
00198                                       DEFAULT_RANDOM_SEED, &error_code);
00199   if (error_code)
00200     {
00201       input_error (_("Bad pseudo-random numbers generator seed"));
00202       goto exit_on_error;
00203     }
00204
00205   // Opening algorithm
00206   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209       input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211       // Obtaining simulations number
00212       input->nsimulations
00213         = xml_node_get_int (node, (const xmlChar *)
      LABEL_NSIMULATIONS,
00214                             &error_code);
00215       if (error_code)
00216         {
00217           input_error (_("Bad simulations number"));
00218           goto exit_on_error;
00219         }
00220     }
00221   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00222     input->algorithm = ALGORITHM_SWEEP;
00223   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00224     input->algorithm = ALGORITHM_ORTHOGONAL;
00225   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226     {
00227       input->algorithm = ALGORITHM_GENETIC;
00228
00229       // Obtaining population
00230       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231         {
00232           input->nsimulations
00233             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00234                                  &error_code);
00235           if (error_code || input->nsimulations < 3)
00236             {
00237               input_error (_("Invalid population number"));
00238               goto exit_on_error;
00239             }
00240         }
00241       else
00242         {
00243           input_error (_("No population number"));
00244           goto exit_on_error;
00245         }
00246
00247       // Obtaining generations
00248       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00249         {
00250           input->niterations
00251             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00252                                  &error_code);
00253           if (error_code || !input->niterations)
00254             {
00255               input_error (_("Invalid generations number"));
00256               goto exit_on_error;
00257             }
00258         }
00259       else
00260         {
00261           input_error (_("No generations number"));
00262           goto exit_on_error;
00263         }
00264
00265       // Obtaining mutation probability
00266       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00267         {
00268           input->mutation_ratio
00269             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00270                                   &error_code);
00271           if (error_code || input->mutation_ratio < 0.
00272               || input->mutation_ratio >= 1.)
00273             {
```

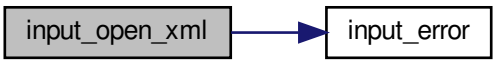
```
00274               input_error (_("Invalid mutation probability"));
00275               goto exit_on_error;
00276             }
00277         }
00278       else
00279         {
00280           input_error (_("No mutation probability"));
00281           goto exit_on_error;
00282         }
00283
00284       // Obtaining reproduction probability
00285       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286         {
00287           input->reproduction_ratio
00288             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00289                                   &error_code);
00290           if (error_code || input->reproduction_ratio < 0.
00291               || input->reproduction_ratio >= 1.0)
00292             {
00293               input_error (_("Invalid reproduction probability"));
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           input_error (_("No reproduction probability"));
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining adaptation probability
00304       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305         {
00306           input->adaptation_ratio
00307             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                                   &error_code);
00309           if (error_code || input->adaptation_ratio < 0.
00310               || input->adaptation_ratio >= 1.)
00311             {
00312               input_error (_("Invalid adaptation probability"));
00313               goto exit_on_error;
00314             }
00315         }
00316       else
00317         {
00318           input_error (_("No adaptation probability"));
00319           goto exit_on_error;
00320         }
00321
00322       // Checking survivals
00323       i = input->mutation_ratio * input->nsimulations;
00324       i += input->reproduction_ratio * input->
      nsimulations;
00325       i += input->adaptation_ratio * input->
      nsimulations;
00326       if (i > input->nsimulations - 2)
00327         {
00328           input_error
00329             (_("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (_("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP
00343       || input->algorithm == ALGORITHM_ORTHOGONAL)
00344     {
00345
00346       // Obtaining iterations number
00347       input->niterations
00348         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00349                              &error_code);
00350       if (error_code == 1)
00351         input->niterations = 1;
00352       else if (error_code)
00353         {
00354           input_error (_("Bad iterations number"));
00355           goto exit_on_error;
00356         }
00357
```

```
00358       // Obtaining best number
00359       input->nbest
00360         = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_NBEST,
00361                                           1, &error_code);
00362       if (error_code || !input->nbest)
00363         {
00364           input_error (_("Invalid best number"));
00365           goto exit_on_error;
00366         }
00367       if (input->nbest > input->nsimulations)
00368         {
00369           input_error (_("Best number higher than simulations number"));
00370           goto exit_on_error;
00371         }
00372
00373       // Obtaining tolerance
00374       input->tolerance
00375         = xml_node_get_float_with_default (node,
00376                                            (const xmlChar *) LABEL_TOLERANCE,
00377                                            0., &error_code);
00378       if (error_code || input->tolerance < 0.)
00379         {
00380           input_error (_("Invalid tolerance"));
00381           goto exit_on_error;
00382         }
00383
00384       // Getting direction search method parameters
00385       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00386         {
00387           input->nsteps =
00388             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00389                                &error_code);
00390           if (error_code)
00391             {
00392               input_error (_("Invalid steps number"));
00393               goto exit_on_error;
00394             }
00395 #if DEBUG_INPUT
00396           fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00397 #endif
00398           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00399           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00400             input->direction = DIRECTION_METHOD_COORDINATES;
00401           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00402             {
00403               input->direction = DIRECTION_METHOD_RANDOM;
00404               input->nestimates
00405                 = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00406                                      &error_code);
00407               if (error_code || !input->nestimates)
00408                 {
00409                   input_error (_("Invalid estimates number"));
00410                   goto exit_on_error;
00411                 }
00412             }
00413           else
00414             {
00415               input_error
00416                 (_("Unknown method to estimate the direction search"));
00417               goto exit_on_error;
00418             }
00419           xmlFree (buffer);
00420           buffer = NULL;
00421           input->relaxation
00422             = xml_node_get_float_with_default (node,
00423                                                (const xmlChar *)
00424                                                LABEL_RELAXATION,
00425                                                DEFAULT_RELAXATION, &error_code);
00426           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00427             {
00428               input_error (_("Invalid relaxation parameter"));
00429               goto exit_on_error;
00430             }
00431         }
00432       else
00433         input->nsteps = 0;
00434     }
00435   // Obtaining the threshold
00436   input->threshold =
00437     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00438                                      0., &error_code);
00439   if (error_code)
00440     {
```

```
00441       input_error (_("Invalid threshold"));
00442       goto exit_on_error;
00443     }
00444
00445   // Reading the experimental data
00446   for (child = node->children; child; child = child->next)
00447     {
00448       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00449         break;
00450 #if DEBUG_INPUT
00451       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00452                input->nexperiments);
00453 #endif
00454       input->experiment = (Experiment *)
00455         g_realloc (input->experiment,
00456                    (1 + input->nexperiments) * sizeof (
      Experiment));
00457       if (!input->nexperiments)
00458         {
00459           if (!experiment_open_xml (input->experiment, child, 0))
00460             goto exit_on_error;
00461         }
00462       else
00463         {
00464           if (!experiment_open_xml (input->experiment +
      input->nexperiments,
00465                                     child, input->experiment->
      ninputs))
00466             goto exit_on_error;
00467         }
00468       ++input->nexperiments;
00469 #if DEBUG_INPUT
00470       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00471                input->nexperiments);
00472 #endif
00473     }
00474   if (!input->nexperiments)
00475     {
00476       input_error (_("No optimization experiments"));
00477       goto exit_on_error;
00478     }
00479   buffer = NULL;
00480
00481   // Reading the variables data
00482   for (; child; child = child->next)
00483     {
00484 #if DEBUG_INPUT
00485       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00486 #endif
00487       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00488         {
00489           snprintf (buffer2, 64, "%s %u: %s",
00490                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00491           input_error (buffer2);
00492           goto exit_on_error;
00493         }
00494       input->variable = (Variable *)
00495         g_realloc (input->variable,
00496                    (1 + input->nvariables) * sizeof (Variable));
00497       if (!variable_open_xml (input->variable +
      input->nvariables, child,
00498                               input->algorithm, input->nsteps))
00499         goto exit_on_error;
00500       ++input->nvariables;
00501     }
00502   if (!input->nvariables)
00503     {
00504       input_error (_("No optimization variables"));
00505       goto exit_on_error;
00506     }
00507   buffer = NULL;
00508
00509   // Obtaining the error norm
00510   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00511     {
00512       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00513       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00514         input->norm = ERROR_NORM_EUCLIDIAN;
00515       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00516         input->norm = ERROR_NORM_MAXIMUM;
00517       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00518         {
00519           input->norm = ERROR_NORM_P;
00520           input->p
00521             = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00522           if (!error_code)
00523             {
```

```
00524               input_error (_("Bad P parameter"));
00525               goto exit_on_error;
00526             }
00527         }
00528       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00529         input->norm = ERROR_NORM_TAXICAB;
00530       else
00531         {
00532           input_error (_("Unknown error norm"));
00533           goto exit_on_error;
00534         }
00535       xmlFree (buffer);
00536     }
00537   else
00538     input->norm = ERROR_NORM_EUCLIDIAN;
00539
00540   // Closing the XML document
00541   xmlFreeDoc (doc);
00542
00543 #if DEBUG_INPUT
00544   fprintf (stderr, "input_open_xml: end\n");
00545 #endif
00546   return 1;
00547
00548 exit_on_error:
00549   xmlFree (buffer);
00550   xmlFreeDoc (doc);
00551 #if DEBUG_INPUT
00552   fprintf (stderr, "input_open_xml: end\n");
00553 #endif
00554   return 0;
00555 }
```

Here is the call graph for this function:



4.8 input.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
```

```
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <string.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <glib/gstdio.h>
00046 #include <json-glib/json-glib.h>
00047 #include "utils.h"
00048 #include "experiment.h"
00049 #include "variable.h"
00050 #include "input.h"
00051
00052 #define DEBUG_INPUT 0
00053
00054 Input input[1];
00055
00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
00062 void
00063 input_new ()
00064 {
00065 #if DEBUG_INPUT
00066   fprintf (stderr, "input_new: start\n");
00067 #endif
00068   input->nvariables = input->nexperiments = input->nsteps = 0;
00069   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00070   input->experiment = NULL;
00071   input->variable = NULL;
00072 #if DEBUG_INPUT
00073   fprintf (stderr, "input_new: end\n");
00074 #endif
00075 }
00076
00080 void
00081 input_free ()
00082 {
00083   unsigned int i;
00084 #if DEBUG_INPUT
00085   fprintf (stderr, "input_free: start\n");
00086 #endif
00087   g_free (input->name);
00088   g_free (input->directory);
00089   for (i = 0; i < input->nexperiments; ++i)
00090     experiment_free (input->experiment + i, input->type);
00091   for (i = 0; i < input->nvariables; ++i)
00092     variable_free (input->variable + i, input->type);
00093   g_free (input->experiment);
00094   g_free (input->variable);
00095   if (input->type == INPUT_TYPE_XML)
00096     {
00097       xmlFree (input->evaluator);
00098       xmlFree (input->simulator);
00099       xmlFree (input->result);
00100       xmlFree (input->variables);
00101     }
00102   else
00103     {
00104       g_free (input->evaluator);
00105       g_free (input->simulator);
00106       g_free (input->result);
00107       g_free (input->variables);
00108     }
00109   input->nexperiments = input->nvariables = input->nsteps = 0;
00110 #if DEBUG_INPUT
00111   fprintf (stderr, "input_free: end\n");
00112 #endif
00113 }
00114
00118 void
00119 input_error (char *message)
00120 {
00121   char buffer[64];
00122   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123   error_message = g_strdup (buffer);
00124 }
00125
00131 int
00132 input_open_xml (xmlDoc * doc)
00133 {
```

```
00134   char buffer2[64];
00135   xmlNode *node, *child;
00136   xmlChar *buffer;
00137   int error_code;
00138   unsigned int i;
00139
00140 #if DEBUG_INPUT
00141   fprintf (stderr, "input_open_xml: start\n");
00142 #endif
00143
00144   // Resetting input data
00145   buffer = NULL;
00146   input->type = INPUT_TYPE_XML;
00147
00148   // Getting the root node
00149 #if DEBUG_INPUT
00150   fprintf (stderr, "input_open_xml: getting the root node\n");
00151 #endif
00152   node = xmlDocGetRootElement (doc);
00153   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155       input_error (_("Bad root XML node"));
00156       goto exit_on_error;
00157     }
00158
00159   // Getting result and variables file names
00160   if (!input->result)
00161     {
00162       input->result =
00163         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164       if (!input->result)
00165         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00166     }
00167 #if DEBUG_INPUT
00168   fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00169 #endif
00170   if (!input->variables)
00171     {
00172       input->variables =
00173         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00174       if (!input->variables)
00175         input->variables =
00176           (char *) xmlStrdup ((const xmlChar *) variables_name);
00177     }
00178 #if DEBUG_INPUT
00179   fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00180 #endif
00181
00182   // Opening simulator program name
00183   input->simulator =
00184     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00185   if (!input->simulator)
00186     {
00187       input_error (_("Bad simulator program"));
00188       goto exit_on_error;
00189     }
00190
00191   // Opening evaluator program name
00192   input->evaluator =
00193     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195   // Obtaining pseudo-random numbers generator seed
00196   input->seed
00197     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_SEED,
00198                                       DEFAULT_RANDOM_SEED, &error_code);
00199   if (error_code)
00200     {
00201       input_error (_("Bad pseudo-random numbers generator seed"));
00202       goto exit_on_error;
00203     }
00204
00205   // Opening algorithm
00206   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209       input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211       // Obtaining simulations number
00212       input->nsimulations
00213         = xml_node_get_int (node, (const xmlChar *)
      LABEL_NSIMULATIONS,
00214                             &error_code);
00215       if (error_code)
00216         {
00217           input_error (_("Bad simulations number"));
00218           goto exit_on_error;
```

```
00219         }
00220     }
00221   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00222     input->algorithm = ALGORITHM_SWEEP;
00223   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00224     input->algorithm = ALGORITHM_ORTHOGONAL;
00225   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226     {
00227       input->algorithm = ALGORITHM_GENETIC;
00228
00229       // Obtaining population
00230       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231         {
00232           input->nsimulations
00233             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00234                                  &error_code);
00235           if (error_code || input->nsimulations < 3)
00236             {
00237               input_error (_("Invalid population number"));
00238               goto exit_on_error;
00239             }
00240         }
00241       else
00242         {
00243           input_error (_("No population number"));
00244           goto exit_on_error;
00245         }
00246
00247       // Obtaining generations
00248       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00249         {
00250           input->niterations
00251             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00252                                  &error_code);
00253           if (error_code || !input->niterations)
00254             {
00255               input_error (_("Invalid generations number"));
00256               goto exit_on_error;
00257             }
00258         }
00259       else
00260         {
00261           input_error (_("No generations number"));
00262           goto exit_on_error;
00263         }
00264
00265       // Obtaining mutation probability
00266       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00267         {
00268           input->mutation_ratio
00269             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00270                                   &error_code);
00271           if (error_code || input->mutation_ratio < 0.
00272               || input->mutation_ratio >= 1.)
00273             {
00274               input_error (_("Invalid mutation probability"));
00275               goto exit_on_error;
00276             }
00277         }
00278       else
00279         {
00280           input_error (_("No mutation probability"));
00281           goto exit_on_error;
00282         }
00283
00284       // Obtaining reproduction probability
00285       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286         {
00287           input->reproduction_ratio
00288             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00289                                   &error_code);
00290           if (error_code || input->reproduction_ratio < 0.
00291               || input->reproduction_ratio >= 1.0)
00292             {
00293               input_error (_("Invalid reproduction probability"));
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           input_error (_("No reproduction probability"));
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining adaptation probability
00304       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305         {
```

```
00306           input->adaptation_ratio
00307             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                                   &error_code);
00309           if (error_code || input->adaptation_ratio < 0.
00310               || input->adaptation_ratio >= 1.)
00311             {
00312               input_error (_("Invalid adaptation probability"));
00313               goto exit_on_error;
00314             }
00315         }
00316       else
00317         {
00318           input_error (_("No adaptation probability"));
00319           goto exit_on_error;
00320         }
00321
00322       // Checking survivals
00323       i = input->mutation_ratio * input->nsimulations;
00324       i += input->reproduction_ratio * input->nsimulations;
00325       i += input->adaptation_ratio * input->nsimulations;
00326       if (i > input->nsimulations - 2)
00327         {
00328           input_error
00329             (_("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (_("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP
00343       || input->algorithm == ALGORITHM_ORTHOGONAL)
00344     {
00345
00346       // Obtaining iterations number
00347       input->niterations
00348         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00349                              &error_code);
00350       if (error_code == 1)
00351         input->niterations = 1;
00352       else if (error_code)
00353         {
00354           input_error (_("Bad iterations number"));
00355           goto exit_on_error;
00356         }
00357
00358       // Obtaining best number
00359       input->nbest
00360         = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_NBEST,
00361                                           1, &error_code);
00362       if (error_code || !input->nbest)
00363         {
00364           input_error (_("Invalid best number"));
00365           goto exit_on_error;
00366         }
00367       if (input->nbest > input->nsimulations)
00368         {
00369           input_error (_("Best number higher than simulations number"));
00370           goto exit_on_error;
00371         }
00372
00373       // Obtaining tolerance
00374       input->tolerance
00375         = xml_node_get_float_with_default (node,
00376                                            (const xmlChar *) LABEL_TOLERANCE,
00377                                            0., &error_code);
00378       if (error_code || input->tolerance < 0.)
00379         {
00380           input_error (_("Invalid tolerance"));
00381           goto exit_on_error;
00382         }
00383
00384       // Getting direction search method parameters
00385       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00386         {
00387           input->nsteps =
00388             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00389                                &error_code);
00390           if (error_code)
```

```
00391             {
00392               input_error (_("Invalid steps number"));
00393               goto exit_on_error;
00394             }
00395 #if DEBUG_INPUT
00396           fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00397 #endif
00398           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00399           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00400             input->direction = DIRECTION_METHOD_COORDINATES;
00401           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00402             {
00403               input->direction = DIRECTION_METHOD_RANDOM;
00404               input->nestimates
00405                 = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00406                                      &error_code);
00407               if (error_code || !input->nestimates)
00408                 {
00409                   input_error (_("Invalid estimates number"));
00410                   goto exit_on_error;
00411                 }
00412             }
00413           else
00414             {
00415               input_error
00416                 (_("Unknown method to estimate the direction search"));
00417               goto exit_on_error;
00418             }
00419           xmlFree (buffer);
00420           buffer = NULL;
00421           input->relaxation
00422             = xml_node_get_float_with_default (node,
00423                                                (const xmlChar *)
00424                                                LABEL_RELAXATION,
00425                                                DEFAULT_RELAXATION, &error_code);
00426           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00427             {
00428               input_error (_("Invalid relaxation parameter"));
00429               goto exit_on_error;
00430             }
00431         }
00432       else
00433         input->nsteps = 0;
00434     }
00435   // Obtaining the threshold
00436   input->threshold =
00437     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00438                                      0., &error_code);
00439   if (error_code)
00440     {
00441       input_error (_("Invalid threshold"));
00442       goto exit_on_error;
00443     }
00444
00445   // Reading the experimental data
00446   for (child = node->children; child; child = child->next)
00447     {
00448       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00449         break;
00450 #if DEBUG_INPUT
00451       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00452                input->nexperiments);
00453 #endif
00454       input->experiment = (Experiment *)
00455         g_realloc (input->experiment,
00456                    (1 + input->nexperiments) * sizeof (Experiment));
00457       if (!input->nexperiments)
00458         {
00459           if (!experiment_open_xml (input->experiment, child, 0))
00460             goto exit_on_error;
00461         }
00462       else
00463         {
00464           if (!experiment_open_xml (input->experiment + input->
      nexperiments,
00465                                     child, input->experiment->ninputs))
00466             goto exit_on_error;
00467         }
00468       ++input->nexperiments;
00469 #if DEBUG_INPUT
00470       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00471                input->nexperiments);
00472 #endif
00473     }
```

```
00474   if (!input->nexperiments)
00475     {
00476       input_error (_("No optimization experiments"));
00477       goto exit_on_error;
00478     }
00479   buffer = NULL;
00480
00481   // Reading the variables data
00482   for (; child; child = child->next)
00483     {
00484 #if DEBUG_INPUT
00485       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00486 #endif
00487       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00488         {
00489           snprintf (buffer2, 64, "%s %u: %s",
00490                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00491           input_error (buffer2);
00492           goto exit_on_error;
00493         }
00494       input->variable = (Variable *)
00495         g_realloc (input->variable,
00496                    (1 + input->nvariables) * sizeof (Variable));
00497       if (!variable_open_xml (input->variable + input->
      nvariables, child,
00498                               input->algorithm, input->nsteps))
00499         goto exit_on_error;
00500       ++input->nvariables;
00501     }
00502   if (!input->nvariables)
00503     {
00504       input_error (_("No optimization variables"));
00505       goto exit_on_error;
00506     }
00507   buffer = NULL;
00508
00509   // Obtaining the error norm
00510   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00511     {
00512       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00513       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00514         input->norm = ERROR_NORM_EUCLIDIAN;
00515       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00516         input->norm = ERROR_NORM_MAXIMUM;
00517       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00518         {
00519           input->norm = ERROR_NORM_P;
00520           input->p
00521             = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00522           if (!error_code)
00523             {
00524               input_error (_("Bad P parameter"));
00525               goto exit_on_error;
00526             }
00527         }
00528       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00529         input->norm = ERROR_NORM_TAXICAB;
00530       else
00531         {
00532           input_error (_("Unknown error norm"));
00533           goto exit_on_error;
00534         }
00535       xmlFree (buffer);
00536     }
00537   else
00538     input->norm = ERROR_NORM_EUCLIDIAN;
00539
00540   // Closing the XML document
00541   xmlFreeDoc (doc);
00542
00543 #if DEBUG_INPUT
00544   fprintf (stderr, "input_open_xml: end\n");
00545 #endif
00546   return 1;
00547
00548 exit_on_error:
00549   xmlFree (buffer);
00550   xmlFreeDoc (doc);
00551 #if DEBUG_INPUT
00552   fprintf (stderr, "input_open_xml: end\n");
00553 #endif
00554   return 0;
00555 }
00556
00562 int
00563 input_open_json (JsonParser * parser)
00564 {
```

```
00565   JsonNode *node, *child;
00566   JsonObject *object;
00567   JsonArray *array;
00568   const char *buffer;
00569   int error_code;
00570   unsigned int i, n;
00571
00572 #if DEBUG_INPUT
00573   fprintf (stderr, "input_open_json: start\n");
00574 #endif
00575
00576   // Resetting input data
00577   input->type = INPUT_TYPE_JSON;
00578
00579   // Getting the root node
00580 #if DEBUG_INPUT
00581   fprintf (stderr, "input_open_json: getting the root node\n");
00582 #endif
00583   node = json_parser_get_root (parser);
00584   object = json_node_get_object (node);
00585
00586   // Getting result and variables file names
00587   if (!input->result)
00588     {
00589       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00590       if (!buffer)
00591         buffer = result_name;
00592       input->result = g_strdup (buffer);
00593     }
00594   else
00595     input->result = g_strdup (result_name);
00596   if (!input->variables)
00597     {
00598       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00599       if (!buffer)
00600         buffer = variables_name;
00601       input->variables = g_strdup (buffer);
00602     }
00603   else
00604     input->variables = g_strdup (variables_name);
00605
00606   // Opening simulator program name
00607   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00608   if (!buffer)
00609     {
00610       input_error (_("Bad simulator program"));
00611       goto exit_on_error;
00612     }
00613   input->simulator = g_strdup (buffer);
00614
00615   // Opening evaluator program name
00616   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00617   if (buffer)
00618     input->evaluator = g_strdup (buffer);
00619
00620   // Obtaining pseudo-random numbers generator seed
00621   input->seed
00622     = json_object_get_uint_with_default (object,
    LABEL_SEED,
00623                                          DEFAULT_RANDOM_SEED, &error_code);
00624   if (error_code)
00625     {
00626       input_error (_("Bad pseudo-random numbers generator seed"));
00627       goto exit_on_error;
00628     }
00629
00630   // Opening algorithm
00631   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00632   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00633     {
00634       input->algorithm = ALGORITHM_MONTE_CARLO;
00635
00636       // Obtaining simulations number
00637       input->nsimulations
00638         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
    );
00639       if (error_code)
00640         {
00641           input_error (_("Bad simulations number"));
00642           goto exit_on_error;
00643         }
00644     }
00645   else if (!strcmp (buffer, LABEL_SWEEP))
00646     input->algorithm = ALGORITHM_SWEEP;
00647   else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00648     input->algorithm = ALGORITHM_ORTHOGONAL;
00649   else if (!strcmp (buffer, LABEL_GENETIC))
```

```
00650     {
00651       input->algorithm = ALGORITHM_GENETIC;
00652
00653       // Obtaining population
00654       if (json_object_get_member (object, LABEL_NPOPULATION))
00655         {
00656           input->nsimulations
00657             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00658           if (error_code || input->nsimulations < 3)
00659             {
00660               input_error (_("Invalid population number"));
00661               goto exit_on_error;
00662             }
00663         }
00664       else
00665         {
00666           input_error (_("No population number"));
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining generations
00671       if (json_object_get_member (object, LABEL_NGENERATIONS))
00672         {
00673           input->niterations
00674             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00675           if (error_code || !input->niterations)
00676             {
00677               input_error (_("Invalid generations number"));
00678               goto exit_on_error;
00679             }
00680         }
00681       else
00682         {
00683           input_error (_("No generations number"));
00684           goto exit_on_error;
00685         }
00686
00687       // Obtaining mutation probability
00688       if (json_object_get_member (object, LABEL_MUTATION))
00689         {
00690           input->mutation_ratio
00691             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00692           if (error_code || input->mutation_ratio < 0.
00693               || input->mutation_ratio >= 1.)
00694             {
00695               input_error (_("Invalid mutation probability"));
00696               goto exit_on_error;
00697             }
00698         }
00699       else
00700         {
00701           input_error (_("No mutation probability"));
00702           goto exit_on_error;
00703         }
00704
00705       // Obtaining reproduction probability
00706       if (json_object_get_member (object, LABEL_REPRODUCTION))
00707         {
00708           input->reproduction_ratio
00709             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00710           if (error_code || input->reproduction_ratio < 0.
00711               || input->reproduction_ratio >= 1.0)
00712             {
00713               input_error (_("Invalid reproduction probability"));
00714               goto exit_on_error;
00715             }
00716         }
00717       else
00718         {
00719           input_error (_("No reproduction probability"));
00720           goto exit_on_error;
00721         }
00722
00723       // Obtaining adaptation probability
00724       if (json_object_get_member (object, LABEL_ADAPTATION))
00725         {
00726           input->adaptation_ratio
00727             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00728           if (error_code || input->adaptation_ratio < 0.
00729               || input->adaptation_ratio >= 1.)
00730             {
00731               input_error (_("Invalid adaptation probability"));
```

```
00732               goto exit_on_error;
00733             }
00734         }
00735       else
00736         {
00737           input_error (_("No adaptation probability"));
00738           goto exit_on_error;
00739         }
00740
00741       // Checking survivals
00742       i = input->mutation_ratio * input->nsimulations;
00743       i += input->reproduction_ratio * input->nsimulations;
00744       i += input->adaptation_ratio * input->nsimulations;
00745       if (i > input->nsimulations - 2)
00746         {
00747           input_error
00748             (_("No enough survival entities to reproduce the population"));
00749           goto exit_on_error;
00750         }
00751     }
00752   else
00753     {
00754       input_error (_("Unknown algorithm"));
00755       goto exit_on_error;
00756     }
00757
00758   if (input->algorithm == ALGORITHM_MONTE_CARLO
00759       || input->algorithm == ALGORITHM_SWEEP
00760       || input->algorithm == ALGORITHM_ORTHOGONAL)
00761     {
00762
00763       // Obtaining iterations number
00764       input->niterations
00765         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00766       if (error_code == 1)
00767         input->niterations = 1;
00768       else if (error_code)
00769         {
00770           input_error (_("Bad iterations number"));
00771           goto exit_on_error;
00772         }
00773
00774       // Obtaining best number
00775       input->nbest
00776         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00777                                              &error_code);
00778       if (error_code || !input->nbest)
00779         {
00780           input_error (_("Invalid best number"));
00781           goto exit_on_error;
00782         }
00783
00784       // Obtaining tolerance
00785       input->tolerance
00786         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00787                                               &error_code);
00788       if (error_code || input->tolerance < 0.)
00789         {
00790           input_error (_("Invalid tolerance"));
00791           goto exit_on_error;
00792         }
00793
00794       // Getting direction search method parameters
00795       if (json_object_get_member (object, LABEL_NSTEPS))
00796         {
00797           input->nsteps
00798             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00799           if (error_code)
00800             {
00801               input_error (_("Invalid steps number"));
00802               goto exit_on_error;
00803             }
00804           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00805           if (!strcmp (buffer, LABEL_COORDINATES))
00806             input->direction = DIRECTION_METHOD_COORDINATES;
00807           else if (!strcmp (buffer, LABEL_RANDOM))
00808             {
00809               input->direction = DIRECTION_METHOD_RANDOM;
00810               input->nestimates
00811                 = json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00812               if (error_code || !input->nestimates)
00813                 {
00814                   input_error (_("Invalid estimates number"));
```

```
00815                   goto exit_on_error;
00816                 }
00817             }
00818           else
00819             {
00820               input_error
00821                 (_("Unknown method to estimate the direction search"));
00822               goto exit_on_error;
00823             }
00824           input->relaxation
00825             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00826                                                   DEFAULT_RELAXATION,
00827                                                   &error_code);
00828           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00829             {
00830               input_error (_("Invalid relaxation parameter"));
00831               goto exit_on_error;
00832             }
00833         }
00834       else
00835         input->nsteps = 0;
00836     }
00837   // Obtaining the threshold
00838   input->threshold
00839     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00840                                           &error_code);
00841   if (error_code)
00842     {
00843       input_error (_("Invalid threshold"));
00844       goto exit_on_error;
00845     }
00846
00847   // Reading the experimental data
00848   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00849   n = json_array_get_length (array);
00850   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00851   for (i = 0; i < n; ++i)
00852     {
00853 #if DEBUG_INPUT
00854       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00855                input->nexperiments);
00856 #endif
00857       child = json_array_get_element (array, i);
00858       if (!input->nexperiments)
00859         {
00860           if (!experiment_open_json (input->experiment, child, 0))
00861             goto exit_on_error;
00862         }
00863       else
00864         {
00865           if (!experiment_open_json (input->experiment + input->
      nexperiments,
00866                                      child, input->experiment->ninputs))
00867             goto exit_on_error;
00868         }
00869       ++input->nexperiments;
00870 #if DEBUG_INPUT
00871       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00872                input->nexperiments);
00873 #endif
00874     }
00875   if (!input->nexperiments)
00876     {
00877       input_error (_("No optimization experiments"));
00878       goto exit_on_error;
00879     }
00880
00881   // Reading the variables data
00882   array = json_object_get_array_member (object, LABEL_VARIABLES);
00883   n = json_array_get_length (array);
00884   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00885   for (i = 0; i < n; ++i)
00886     {
00887 #if DEBUG_INPUT
00888       fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00889 #endif
00890       child = json_array_get_element (array, i);
00891       if (!variable_open_json (input->variable + input->
      nvariables, child,
00892                                input->algorithm, input->nsteps))
00893         goto exit_on_error;
00894       ++input->nvariables;
00895     }
```

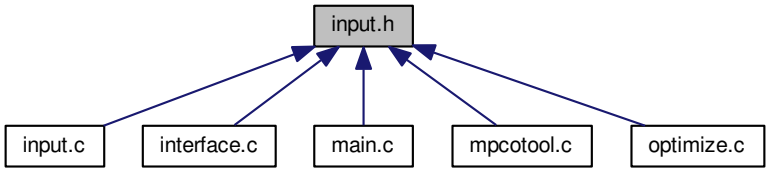
```
00896   if (!input->nvariables)
00897     {
00898       input_error (_("No optimization variables"));
00899       goto exit_on_error;
00900     }
00901
00902   // Obtaining the error norm
00903   if (json_object_get_member (object, LABEL_NORM))
00904     {
00905       buffer = json_object_get_string_member (object, LABEL_NORM);
00906       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00907         input->norm = ERROR_NORM_EUCLIDIAN;
00908       else if (!strcmp (buffer, LABEL_MAXIMUM))
00909         input->norm = ERROR_NORM_MAXIMUM;
00910       else if (!strcmp (buffer, LABEL_P))
00911         {
00912           input->norm = ERROR_NORM_P;
00913           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00914           if (!error_code)
00915             {
00916               input_error (_("Bad P parameter"));
00917               goto exit_on_error;
00918             }
00919         }
00920       else if (!strcmp (buffer, LABEL_TAXICAB))
00921         input->norm = ERROR_NORM_TAXICAB;
00922       else
00923         {
00924           input_error (_("Unknown error norm"));
00925           goto exit_on_error;
00926         }
00927     }
00928   else
00929     input->norm = ERROR_NORM_EUCLIDIAN;
00930
00931   // Closing the JSON document
00932   g_object_unref (parser);
00933
00934 #if DEBUG_INPUT
00935   fprintf (stderr, "input_open_json: end\n");
00936 #endif
00937   return 1;
00938
00939 exit_on_error:
00940   g_object_unref (parser);
00941 #if DEBUG_INPUT
00942   fprintf (stderr, "input_open_json: end\n");
00943 #endif
00944   return 0;
00945 }
00946
00952 int
00953 input_open (char *filename)
00954 {
00955   xmlDoc *doc;
00956   JsonParser *parser;
00957
00958 #if DEBUG_INPUT
00959   fprintf (stderr, "input_open: start\n");
00960 #endif
00961
00962   // Resetting input data
00963   input_new ();
00964
00965   // Opening input file
00966 #if DEBUG_INPUT
00967   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00968   fprintf (stderr, "input_open: trying XML format\n");
00969 #endif
00970   doc = xmlParseFile (filename);
00971   if (!doc)
00972     {
00973 #if DEBUG_INPUT
00974       fprintf (stderr, "input_open: trying JSON format\n");
00975 #endif
00976       parser = json_parser_new ();
00977       if (!json_parser_load_from_file (parser, filename, NULL))
00978         {
00979           input_error (_("Unable to parse the input file"));
00980           goto exit_on_error;
00981         }
00982       if (!input_open_json (parser))
00983         goto exit_on_error;
00984     }
00985   else if (!input_open_xml (doc))
00986     goto exit_on_error;
```

```
00987
00988   // Getting the working directory
00989   input->directory = g_path_get_dirname (filename);
00990   input->name = g_path_get_basename (filename);
00991
00992 #if DEBUG_INPUT
00993   fprintf (stderr, "input_open: end\n");
00994 #endif
00995   return 1;
00996
00997 exit_on_error:
00998   show_error (error_message);
00999   g_free (error_message);
01000   input_free ();
01001 #if DEBUG_INPUT
01002   fprintf (stderr, "input_open: end\n");
01003 #endif
01004   return 0;
01005 }
```

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 DirectionMethod { DIRECTION_METHOD_COORDINATES = 0, DIRECTION_METHOD_RANDOM = 1 }

  Enum to define the methods to estimate the direction search.
- 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 DirectionMethod

```
enum DirectionMethod
```

Enum to define the methods to estimate the direction search.

Enumerator

DIRECTION_METHOD_COORDINATES	Coordinates descent method.
DIRECTION_METHOD_RANDOM	Random method.

Definition at line 45 of file input.h.

```
00046 {
00047   DIRECTION_METHOD_COORDINATES = 0,
00048   DIRECTION_METHOD_RANDOM = 1,
00049 };
```

4.9.2.2 ErrorNorm

```
enum ErrorNorm
```

Enum to define the error norm.

Enumerator

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

Definition at line 55 of file input.h.

```
00056 {
00057   ERROR_NORM_EUCLIDIAN = 0,
00059   ERROR_NORM_MAXIMUM = 1,
00061   ERROR_NORM_P = 2,
00063   ERROR_NORM_TAXICAB = 3
00065 };
```

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.

```
00120 {
00121   char buffer[64];
00122   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00123   error_message = g_strdup (buffer);
00124 }
```

4.9.3.2 input_free()

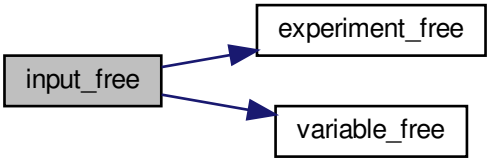
```
void input_free ( )
```

Function to free the memory of the input file data.

Definition at line 81 of file input.c.

```
00082 {
00083   unsigned int i;
00084 #if DEBUG_INPUT
00085   fprintf (stderr, "input_free: start\n");
00086 #endif
00087   g_free (input->name);
00088   g_free (input->directory);
00089   for (i = 0; i < input->nexperiments; ++i)
00090     experiment_free (input->experiment + i, input->
      type);
00091   for (i = 0; i < input->nvariables; ++i)
00092     variable_free (input->variable + i, input->
      type);
00093   g_free (input->experiment);
00094   g_free (input->variable);
00095   if (input->type == INPUT_TYPE_XML)
00096     {
00097       xmlFree (input->evaluator);
00098       xmlFree (input->simulator);
00099       xmlFree (input->result);
00100       xmlFree (input->variables);
00101     }
00102   else
00103     {
00104       g_free (input->evaluator);
00105       g_free (input->simulator);
00106       g_free (input->result);
00107       g_free (input->variables);
00108     }
00109   input->nexperiments = input->nvariables =
      input->nsteps = 0;
00110 #if DEBUG_INPUT
00111   fprintf (stderr, "input_free: end\n");
00112 #endif
00113 }
```

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.

```
00064 {
00065 #if DEBUG_INPUT
00066   fprintf (stderr, "input_new: start\n");
00067 #endif
00068   input->nvariables = input->nexperiments =
      input->nsteps = 0;
00069   input->simulator = input->evaluator = input->
      directory = input->name = NULL;
00070   input->experiment = NULL;
00071   input->variable = NULL;
00072 #if DEBUG_INPUT
00073   fprintf (stderr, "input_new: end\n");
00074 #endif
00075 }
```

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

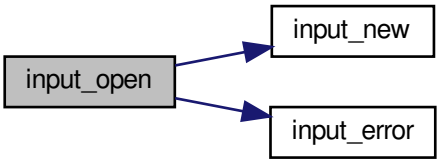
filename	Input data file name.

Definition at line 953 of file input.c.

```
00954 {
00955   xmlDoc *doc;
00956   JsonParser *parser;
00957
00958 #if DEBUG_INPUT
00959   fprintf (stderr, "input_open: start\n");
00960 #endif
00961
00962   // Resetting input data
00963   input_new ();
00964
00965   // Opening input file
00966 #if DEBUG_INPUT
00967   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00968   fprintf (stderr, "input_open: trying XML format\n");
00969 #endif
00970   doc = xmlParseFile (filename);
00971   if (!doc)
00972     {
00973 #if DEBUG_INPUT
```

```
00974       fprintf (stderr, "input_open: trying JSON format\n");
00975 #endif
00976       parser = json_parser_new ();
00977       if (!json_parser_load_from_file (parser, filename, NULL))
00978         {
00979           input_error (_("Unable to parse the input file"));
00980           goto exit_on_error;
00981         }
00982       if (!input_open_json (parser))
00983         goto exit_on_error;
00984     }
00985   else if (!input_open_xml (doc))
00986     goto exit_on_error;
00987
00988   // Getting the working directory
00989   input->directory = g_path_get_dirname (filename);
00990   input->name = g_path_get_basename (filename);
00991
00992 #if DEBUG_INPUT
00993   fprintf (stderr, "input_open: end\n");
00994 #endif
00995   return 1;
00996
00997 exit_on_error:
00998   show_error (error_message);
00999   g_free (error_message);
01000   input_free ();
01001 #if DEBUG_INPUT
01002   fprintf (stderr, "input_open: end\n");
01003 #endif
01004   return 0;
01005 }
```

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 563 of file input.c.

```
00564 {
00565   JsonNode *node, *child;
00566   JsonObject *object;
00567   JsonArray *array;
00568   const char *buffer;
00569   int error_code;
00570   unsigned int i, n;
00571
00572 #if DEBUG_INPUT
00573   fprintf (stderr, "input_open_json: start\n");
00574 #endif
00575
00576   // Resetting input data
00577   input->type = INPUT_TYPE_JSON;
00578
00579   // Getting the root node
00580 #if DEBUG_INPUT
00581   fprintf (stderr, "input_open_json: getting the root node\n");
00582 #endif
00583   node = json_parser_get_root (parser);
00584   object = json_node_get_object (node);
00585
00586   // Getting result and variables file names
00587   if (!input->result)
00588     {
00589       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00590       if (!buffer)
00591         buffer = result_name;
00592       input->result = g_strdup (buffer);
00593     }
00594   else
00595     input->result = g_strdup (result_name);
00596   if (!input->variables)
00597     {
00598       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00599       if (!buffer)
00600         buffer = variables_name;
00601       input->variables = g_strdup (buffer);
00602     }
00603   else
00604     input->variables = g_strdup (variables_name);
00605
00606   // Opening simulator program name
00607   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00608   if (!buffer)
00609     {
00610       input_error (_("Bad simulator program"));
00611       goto exit_on_error;
00612     }
00613   input->simulator = g_strdup (buffer);
00614
00615   // Opening evaluator program name
00616   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00617   if (buffer)
00618     input->evaluator = g_strdup (buffer);
00619
00620   // Obtaining pseudo-random numbers generator seed
00621   input->seed
00622     = json_object_get_uint_with_default (object,
    LABEL_SEED,
00623                                          DEFAULT_RANDOM_SEED, &error_code);
00624   if (error_code)
00625     {
00626       input_error (_("Bad pseudo-random numbers generator seed"));
00627       goto exit_on_error;
00628     }
00629
00630   // Opening algorithm
00631   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00632   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00633     {
00634       input->algorithm = ALGORITHM_MONTE_CARLO;
00635
00636       // Obtaining simulations number
00637       input->nsimulations
00638         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
    );
00639       if (error_code)
00640         {
00641           input_error (_("Bad simulations number"));
00642           goto exit_on_error;
00643         }
00644     }
00645   else if (!strcmp (buffer, LABEL_SWEEP))
```

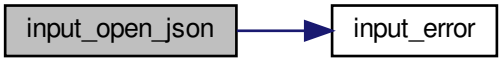
```
00646     input->algorithm = ALGORITHM_SWEEP;
00647   else if (!strcmp (buffer, LABEL_ORTHOGONAL))
00648     input->algorithm = ALGORITHM_ORTHOGONAL;
00649   else if (!strcmp (buffer, LABEL_GENETIC))
00650     {
00651       input->algorithm = ALGORITHM_GENETIC;
00652
00653       // Obtaining population
00654       if (json_object_get_member (object, LABEL_NPOPULATION))
00655         {
00656           input->nsimulations
00657             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00658           if (error_code || input->nsimulations < 3)
00659             {
00660               input_error (_("Invalid population number"));
00661               goto exit_on_error;
00662             }
00663         }
00664       else
00665         {
00666           input_error (_("No population number"));
00667           goto exit_on_error;
00668         }
00669
00670       // Obtaining generations
00671       if (json_object_get_member (object, LABEL_NGENERATIONS))
00672         {
00673           input->niterations
00674             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00675           if (error_code || !input->niterations)
00676             {
00677               input_error (_("Invalid generations number"));
00678               goto exit_on_error;
00679             }
00680         }
00681       else
00682         {
00683           input_error (_("No generations number"));
00684           goto exit_on_error;
00685         }
00686
00687       // Obtaining mutation probability
00688       if (json_object_get_member (object, LABEL_MUTATION))
00689         {
00690           input->mutation_ratio
00691             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00692           if (error_code || input->mutation_ratio < 0.
00693               || input->mutation_ratio >= 1.)
00694             {
00695               input_error (_("Invalid mutation probability"));
00696               goto exit_on_error;
00697             }
00698         }
00699       else
00700         {
00701           input_error (_("No mutation probability"));
00702           goto exit_on_error;
00703         }
00704
00705       // Obtaining reproduction probability
00706       if (json_object_get_member (object, LABEL_REPRODUCTION))
00707         {
00708           input->reproduction_ratio
00709             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00710           if (error_code || input->reproduction_ratio < 0.
00711               || input->reproduction_ratio >= 1.0)
00712             {
00713               input_error (_("Invalid reproduction probability"));
00714               goto exit_on_error;
00715             }
00716         }
00717       else
00718         {
00719           input_error (_("No reproduction probability"));
00720           goto exit_on_error;
00721         }
00722
00723       // Obtaining adaptation probability
00724       if (json_object_get_member (object, LABEL_ADAPTATION))
00725         {
00726           input->adaptation_ratio
00727             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
```

```
00728           if (error_code || input->adaptation_ratio < 0.
00729               || input->adaptation_ratio >= 1.)
00730             {
00731               input_error (_("Invalid adaptation probability"));
00732               goto exit_on_error;
00733             }
00734         }
00735       else
00736         {
00737           input_error (_("No adaptation probability"));
00738           goto exit_on_error;
00739         }
00740
00741       // Checking survivals
00742       i = input->mutation_ratio * input->nsimulations;
00743       i += input->reproduction_ratio * input->
      nsimulations;
00744       i += input->adaptation_ratio * input->
      nsimulations;
00745       if (i > input->nsimulations - 2)
00746         {
00747           input_error
00748             (_("No enough survival entities to reproduce the population"));
00749           goto exit_on_error;
00750         }
00751     }
00752   else
00753     {
00754       input_error (_("Unknown algorithm"));
00755       goto exit_on_error;
00756     }
00757
00758   if (input->algorithm == ALGORITHM_MONTE_CARLO
00759       || input->algorithm == ALGORITHM_SWEEP
00760       || input->algorithm == ALGORITHM_ORTHOGONAL)
00761     {
00762
00763       // Obtaining iterations number
00764       input->niterations
00765         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00766       if (error_code == 1)
00767         input->niterations = 1;
00768       else if (error_code)
00769         {
00770           input_error (_("Bad iterations number"));
00771           goto exit_on_error;
00772         }
00773
00774       // Obtaining best number
00775       input->nbest
00776         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00777                                              &error_code);
00778       if (error_code || !input->nbest)
00779         {
00780           input_error (_("Invalid best number"));
00781           goto exit_on_error;
00782         }
00783
00784       // Obtaining tolerance
00785       input->tolerance
00786         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00787                                               &error_code);
00788       if (error_code || input->tolerance < 0.)
00789         {
00790           input_error (_("Invalid tolerance"));
00791           goto exit_on_error;
00792         }
00793
00794       // Getting direction search method parameters
00795       if (json_object_get_member (object, LABEL_NSTEPS))
00796         {
00797           input->nsteps
00798             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00799           if (error_code)
00800             {
00801               input_error (_("Invalid steps number"));
00802               goto exit_on_error;
00803             }
00804           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00805           if (!strcmp (buffer, LABEL_COORDINATES))
00806             input->direction = DIRECTION_METHOD_COORDINATES;
00807           else if (!strcmp (buffer, LABEL_RANDOM))
00808             {
00809               input->direction = DIRECTION_METHOD_RANDOM;
```

```
00810               input->nestimates
00811                 = json_object_get_uint (object,
    LABEL_NESTIMATES, &error_code);
00812               if (error_code || !input->nestimates)
00813                 {
00814                   input_error (_("Invalid estimates number"));
00815                   goto exit_on_error;
00816                 }
00817             }
00818           else
00819             {
00820               input_error
00821                 (_("Unknown method to estimate the direction search"));
00822               goto exit_on_error;
00823             }
00824           input->relaxation
00825             = json_object_get_float_with_default (object,
    LABEL_RELAXATION,
00826                                                   DEFAULT_RELAXATION,
00827                                                   &error_code);
00828           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00829             {
00830               input_error (_("Invalid relaxation parameter"));
00831               goto exit_on_error;
00832             }
00833         }
00834       else
00835         input->nsteps = 0;
00836     }
00837   // Obtaining the threshold
00838   input->threshold
00839     = json_object_get_float_with_default (object,
    LABEL_THRESHOLD, 0.,
00840                                           &error_code);
00841   if (error_code)
00842     {
00843       input_error (_("Invalid threshold"));
00844       goto exit_on_error;
00845     }
00846
00847   // Reading the experimental data
00848   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00849   n = json_array_get_length (array);
00850   input->experiment = (Experiment *) g_malloc (n * sizeof (
    Experiment));
00851   for (i = 0; i < n; ++i)
00852     {
00853 #if DEBUG_INPUT
00854       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00855                input->nexperiments);
00856 #endif
00857       child = json_array_get_element (array, i);
00858       if (!input->nexperiments)
00859         {
00860           if (!experiment_open_json (input->experiment, child, 0))
00861             goto exit_on_error;
00862         }
00863       else
00864         {
00865           if (!experiment_open_json (input->experiment +
    input->nexperiments,
00866                                      child, input->experiment->
    ninputs))
00867             goto exit_on_error;
00868         }
00869       ++input->nexperiments;
00870 #if DEBUG_INPUT
00871       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00872                input->nexperiments);
00873 #endif
00874     }
00875   if (!input->nexperiments)
00876     {
00877       input_error (_("No optimization experiments"));
00878       goto exit_on_error;
00879     }
00880
00881   // Reading the variables data
00882   array = json_object_get_array_member (object, LABEL_VARIABLES);
00883   n = json_array_get_length (array);
00884   input->variable = (Variable *) g_malloc (n * sizeof (
    Variable));
00885   for (i = 0; i < n; ++i)
00886     {
00887 #if DEBUG_INPUT
00888       fprintf (stderr, "input_open_json: nvariables=%u\n", input->
```

```
      nvariables);
00889 #endif
00890       child = json_array_get_element (array, i);
00891       if (!variable_open_json (input->variable +
      input->nvariables, child,
00892                                input->algorithm, input->
      nsteps))
00893         goto exit_on_error;
00894       ++input->nvariables;
00895     }
00896   if (!input->nvariables)
00897     {
00898       input_error (_("No optimization variables"));
00899       goto exit_on_error;
00900     }
00901
00902   // Obtaining the error norm
00903   if (json_object_get_member (object, LABEL_NORM))
00904     {
00905       buffer = json_object_get_string_member (object, LABEL_NORM);
00906       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00907         input->norm = ERROR_NORM_EUCLIDIAN;
00908       else if (!strcmp (buffer, LABEL_MAXIMUM))
00909         input->norm = ERROR_NORM_MAXIMUM;
00910       else if (!strcmp (buffer, LABEL_P))
00911         {
00912           input->norm = ERROR_NORM_P;
00913           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00914           if (!error_code)
00915             {
00916               input_error (_("Bad P parameter"));
00917               goto exit_on_error;
00918             }
00919         }
00920       else if (!strcmp (buffer, LABEL_TAXICAB))
00921         input->norm = ERROR_NORM_TAXICAB;
00922       else
00923         {
00924           input_error (_("Unknown error norm"));
00925           goto exit_on_error;
00926         }
00927     }
00928   else
00929     input->norm = ERROR_NORM_EUCLIDIAN;
00930
00931   // Closing the JSON document
00932   g_object_unref (parser);
00933
00934 #if DEBUG_INPUT
00935   fprintf (stderr, "input_open_json: end\n");
00936 #endif
00937   return 1;
00938
00939 exit_on_error:
00940   g_object_unref (parser);
00941 #if DEBUG_INPUT
00942   fprintf (stderr, "input_open_json: end\n");
00943 #endif
00944   return 0;
00945 }
```

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.

```
00133 {
00134   char buffer2[64];
00135   xmlNode *node, *child;
00136   xmlChar *buffer;
00137   int error_code;
00138   unsigned int i;
00139
00140 #if DEBUG_INPUT
00141   fprintf (stderr, "input_open_xml: start\n");
00142 #endif
00143
00144   // Resetting input data
00145   buffer = NULL;
00146   input->type = INPUT_TYPE_XML;
00147
00148   // Getting the root node
00149 #if DEBUG_INPUT
00150   fprintf (stderr, "input_open_xml: getting the root node\n");
00151 #endif
00152   node = xmlDocGetRootElement (doc);
00153   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00154     {
00155       input_error (_("Bad root XML node"));
00156       goto exit_on_error;
00157     }
00158
00159   // Getting result and variables file names
00160   if (!input->result)
00161     {
00162       input->result =
00163         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00164       if (!input->result)
00165         input->result = (char *) xmlStrdup ((const xmlChar *)
      result_name);
00166     }
00167 #if DEBUG_INPUT
00168   fprintf (stderr, "input_open_xml: result file=%s\n", input->result);
00169 #endif
00170   if (!input->variables)
00171     {
00172       input->variables =
00173         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00174       if (!input->variables)
00175         input->variables =
00176           (char *) xmlStrdup ((const xmlChar *) variables_name);
00177     }
00178 #if DEBUG_INPUT
00179   fprintf (stderr, "input_open_xml: variables file=%s\n", input->variables);
00180 #endif
00181
00182   // Opening simulator program name
00183   input->simulator =
00184     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00185   if (!input->simulator)
00186     {
00187       input_error (_("Bad simulator program"));
00188       goto exit_on_error;
```

```
00189     }
00190
00191   // Opening evaluator program name
00192   input->evaluator =
00193     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00194
00195   // Obtaining pseudo-random numbers generator seed
00196   input->seed
00197     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_SEED,
00198                                       DEFAULT_RANDOM_SEED, &error_code);
00199   if (error_code)
00200     {
00201       input_error (_("Bad pseudo-random numbers generator seed"));
00202       goto exit_on_error;
00203     }
00204
00205   // Opening algorithm
00206   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209       input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211       // Obtaining simulations number
00212       input->nsimulations
00213         = xml_node_get_int (node, (const xmlChar *)
      LABEL_NSIMULATIONS,
00214                             &error_code);
00215       if (error_code)
00216         {
00217           input_error (_("Bad simulations number"));
00218           goto exit_on_error;
00219         }
00220     }
00221   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00222     input->algorithm = ALGORITHM_SWEEP;
00223   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_ORTHOGONAL))
00224     input->algorithm = ALGORITHM_ORTHOGONAL;
00225   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226     {
00227       input->algorithm = ALGORITHM_GENETIC;
00228
00229       // Obtaining population
00230       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231         {
00232           input->nsimulations
00233             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00234                                  &error_code);
00235           if (error_code || input->nsimulations < 3)
00236             {
00237               input_error (_("Invalid population number"));
00238               goto exit_on_error;
00239             }
00240         }
00241       else
00242         {
00243           input_error (_("No population number"));
00244           goto exit_on_error;
00245         }
00246
00247       // Obtaining generations
00248       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00249         {
00250           input->niterations
00251             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00252                                  &error_code);
00253           if (error_code || !input->niterations)
00254             {
00255               input_error (_("Invalid generations number"));
00256               goto exit_on_error;
00257             }
00258         }
00259       else
00260         {
00261           input_error (_("No generations number"));
00262           goto exit_on_error;
00263         }
00264
00265       // Obtaining mutation probability
00266       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00267         {
00268           input->mutation_ratio
00269             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00270                                   &error_code);
00271           if (error_code || input->mutation_ratio < 0.
00272               || input->mutation_ratio >= 1.)
00273             {
```

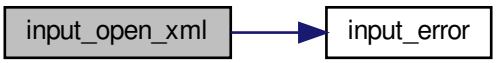
```
00274               input_error (_("Invalid mutation probability"));
00275               goto exit_on_error;
00276             }
00277         }
00278       else
00279         {
00280           input_error (_("No mutation probability"));
00281           goto exit_on_error;
00282         }
00283
00284       // Obtaining reproduction probability
00285       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286         {
00287           input->reproduction_ratio
00288             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00289                                   &error_code);
00290           if (error_code || input->reproduction_ratio < 0.
00291               || input->reproduction_ratio >= 1.0)
00292             {
00293               input_error (_("Invalid reproduction probability"));
00294               goto exit_on_error;
00295             }
00296         }
00297       else
00298         {
00299           input_error (_("No reproduction probability"));
00300           goto exit_on_error;
00301         }
00302
00303       // Obtaining adaptation probability
00304       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00305         {
00306           input->adaptation_ratio
00307             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00308                                   &error_code);
00309           if (error_code || input->adaptation_ratio < 0.
00310               || input->adaptation_ratio >= 1.)
00311             {
00312               input_error (_("Invalid adaptation probability"));
00313               goto exit_on_error;
00314             }
00315         }
00316       else
00317         {
00318           input_error (_("No adaptation probability"));
00319           goto exit_on_error;
00320         }
00321
00322       // Checking survivals
00323       i = input->mutation_ratio * input->nsimulations;
00324       i += input->reproduction_ratio * input->
      nsimulations;
00325       i += input->adaptation_ratio * input->
      nsimulations;
00326       if (i > input->nsimulations - 2)
00327         {
00328           input_error
00329             (_("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (_("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP
00343       || input->algorithm == ALGORITHM_ORTHOGONAL)
00344     {
00345
00346       // Obtaining iterations number
00347       input->niterations
00348         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00349                              &error_code);
00350       if (error_code == 1)
00351         input->niterations = 1;
00352       else if (error_code)
00353         {
00354           input_error (_("Bad iterations number"));
00355           goto exit_on_error;
00356         }
00357
```

```
00358       // Obtaining best number
00359       input->nbest
00360         = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_NBEST,
00361                                           1, &error_code);
00362       if (error_code || !input->nbest)
00363         {
00364           input_error (_("Invalid best number"));
00365           goto exit_on_error;
00366         }
00367       if (input->nbest > input->nsimulations)
00368         {
00369           input_error (_("Best number higher than simulations number"));
00370           goto exit_on_error;
00371         }
00372
00373       // Obtaining tolerance
00374       input->tolerance
00375         = xml_node_get_float_with_default (node,
00376                                            (const xmlChar *) LABEL_TOLERANCE,
00377                                            0., &error_code);
00378       if (error_code || input->tolerance < 0.)
00379         {
00380           input_error (_("Invalid tolerance"));
00381           goto exit_on_error;
00382         }
00383
00384       // Getting direction search method parameters
00385       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00386         {
00387           input->nsteps =
00388             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00389                                &error_code);
00390           if (error_code)
00391             {
00392               input_error (_("Invalid steps number"));
00393               goto exit_on_error;
00394             }
00395 #if DEBUG_INPUT
00396           fprintf (stderr, "input_open_xml: nsteps=%u\n", input->nsteps);
00397 #endif
00398           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00399           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00400             input->direction = DIRECTION_METHOD_COORDINATES;
00401           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00402             {
00403               input->direction = DIRECTION_METHOD_RANDOM;
00404               input->nestimates
00405                 = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00406                                      &error_code);
00407               if (error_code || !input->nestimates)
00408                 {
00409                   input_error (_("Invalid estimates number"));
00410                   goto exit_on_error;
00411                 }
00412             }
00413           else
00414             {
00415               input_error
00416                 (_("Unknown method to estimate the direction search"));
00417               goto exit_on_error;
00418             }
00419           xmlFree (buffer);
00420           buffer = NULL;
00421           input->relaxation
00422             = xml_node_get_float_with_default (node,
00423                                                (const xmlChar *)
00424                                                LABEL_RELAXATION,
00425                                                DEFAULT_RELAXATION, &error_code);
00426           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00427             {
00428               input_error (_("Invalid relaxation parameter"));
00429               goto exit_on_error;
00430             }
00431         }
00432       else
00433         input->nsteps = 0;
00434     }
00435   // Obtaining the threshold
00436   input->threshold =
00437     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00438                                      0., &error_code);
00439   if (error_code)
00440     {
```

```
00441       input_error (_("Invalid threshold"));
00442       goto exit_on_error;
00443     }
00444
00445   // Reading the experimental data
00446   for (child = node->children; child; child = child->next)
00447     {
00448       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00449         break;
00450 #if DEBUG_INPUT
00451       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00452                input->nexperiments);
00453 #endif
00454       input->experiment = (Experiment *)
00455         g_realloc (input->experiment,
00456                    (1 + input->nexperiments) * sizeof (
      Experiment));
00457       if (!input->nexperiments)
00458         {
00459           if (!experiment_open_xml (input->experiment, child, 0))
00460             goto exit_on_error;
00461         }
00462       else
00463         {
00464           if (!experiment_open_xml (input->experiment +
      input->nexperiments,
00465                                     child, input->experiment->
      ninputs))
00466             goto exit_on_error;
00467         }
00468       ++input->nexperiments;
00469 #if DEBUG_INPUT
00470       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00471                input->nexperiments);
00472 #endif
00473     }
00474   if (!input->nexperiments)
00475     {
00476       input_error (_("No optimization experiments"));
00477       goto exit_on_error;
00478     }
00479   buffer = NULL;
00480
00481   // Reading the variables data
00482   for (; child; child = child->next)
00483     {
00484 #if DEBUG_INPUT
00485       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00486 #endif
00487       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00488         {
00489           snprintf (buffer2, 64, "%s %u: %s",
00490                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00491           input_error (buffer2);
00492           goto exit_on_error;
00493         }
00494       input->variable = (Variable *)
00495         g_realloc (input->variable,
00496                    (1 + input->nvariables) * sizeof (Variable));
00497       if (!variable_open_xml (input->variable +
      input->nvariables, child,
00498                               input->algorithm, input->nsteps))
00499         goto exit_on_error;
00500       ++input->nvariables;
00501     }
00502   if (!input->nvariables)
00503     {
00504       input_error (_("No optimization variables"));
00505       goto exit_on_error;
00506     }
00507   buffer = NULL;
00508
00509   // Obtaining the error norm
00510   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00511     {
00512       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00513       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00514         input->norm = ERROR_NORM_EUCLIDIAN;
00515       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00516         input->norm = ERROR_NORM_MAXIMUM;
00517       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00518         {
00519           input->norm = ERROR_NORM_P;
00520           input->p
00521             = xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00522           if (!error_code)
00523             {
```

```
00524               input_error (_("Bad P parameter"));
00525               goto exit_on_error;
00526             }
00527         }
00528       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00529         input->norm = ERROR_NORM_TAXICAB;
00530       else
00531         {
00532           input_error (_("Unknown error norm"));
00533           goto exit_on_error;
00534         }
00535       xmlFree (buffer);
00536     }
00537   else
00538     input->norm = ERROR_NORM_EUCLIDIAN;
00539
00540   // Closing the XML document
00541   xmlFreeDoc (doc);
00542
00543 #if DEBUG_INPUT
00544   fprintf (stderr, "input_open_xml: end\n");
00545 #endif
00546   return 1;
00547
00548 exit_on_error:
00549   xmlFree (buffer);
00550   xmlFreeDoc (doc);
00551 #if DEBUG_INPUT
00552   fprintf (stderr, "input_open_xml: end\n");
00553 #endif
00554   return 0;
00555 }
```

Here is the call graph for this function:



4.10 input.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
```

```
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INPUT__H
00039 #define INPUT__H 1
00040
00045 enum DirectionMethod
00046 {
00047   DIRECTION_METHOD_COORDINATES = 0,
00048   DIRECTION_METHOD_RANDOM = 1,
00049 };
00050
00055 enum ErrorNorm
00056 {
00057   ERROR_NORM_EUCLIDIAN = 0,
00059   ERROR_NORM_MAXIMUM = 1,
00061   ERROR_NORM_P = 2,
00063   ERROR_NORM_TAXICAB = 3
00065 };
00066
00071 typedef struct
00072 {
00073   Experiment *experiment;
00074   Variable *variable;
00075   char *result;
00076   char *variables;
00077   char *simulator;
00078   char *evaluator;
00080   char *directory;
00081   char *name;
00082   double tolerance;
00083   double mutation_ratio;
00084   double reproduction_ratio;
00085   double adaptation_ratio;
00086   double relaxation;
00087   double p;
00088   double threshold;
00089   unsigned long int seed;
00091   unsigned int nvariables;
00092   unsigned int nexperiments;
00093   unsigned int nsimulations;
00094   unsigned int algorithm;
00095   unsigned int nsteps;
00097   unsigned int direction;
00098   unsigned int nestimates;
00100   unsigned int niterations;
00101   unsigned int nbest;
00102   unsigned int norm;
00103   unsigned int type;
00104 } Input;
00105
00106 extern Input input[1];
00107 extern const char *result_name;
00108 extern const char *variables_name;
00109
00110 // Public functions
00111 void input_new ();
00112 void input_free ();
00113 void input_error (char *message);
00114 int input_open_xml (xmlDoc * doc);
00115 int input_open_json (JsonParser * parser);
00116 int input_open (char *filename);
00117
00118 #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_direction_xml (xmlNode *node)
- void input_save_direction_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_direction ()
- unsigned int window_get_norm ()
- void window_save_direction ()
- int window_save ()
- void window_run ()
- void window_help ()
- void window_about ()
- void window_update_direction ()
- 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

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.

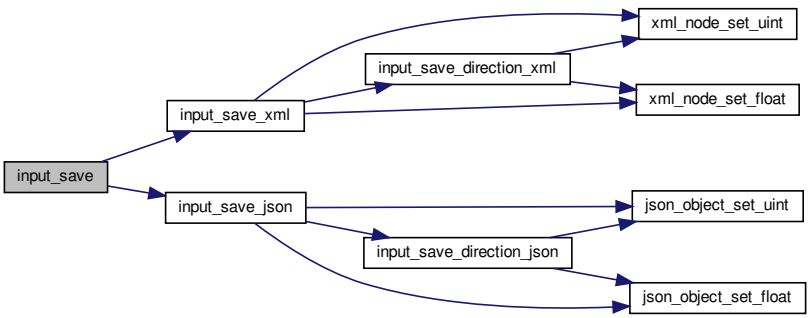
Parameters

filename	Input file name.

Definition at line 584 of file interface.c.

```
00585 {
00586   xmlDoc *doc;
00587   JsonGenerator *generator;
00588
00589 #if DEBUG_INTERFACE
00590   fprintf (stderr, "input_save: start\n");
00591 #endif
00592
00593   // Getting the input file directory
00594   input->name = g_path_get_basename (filename);
00595   input->directory = g_path_get_dirname (filename);
00596
00597   if (input->type == INPUT_TYPE_XML)
00598     {
00599       // Opening the input file
00600       doc = xmlNewDoc ((const xmlChar *) "1.0");
00601       input_save_xml (doc);
00602
00603       // Saving the XML file
00604       xmlSaveFormatFile (filename, doc, 1);
00605
00606       // Freeing memory
00607       xmlFreeDoc (doc);
00608     }
00609   else
00610     {
00611       // Opening the input file
00612       generator = json_generator_new ();
00613       json_generator_set_pretty (generator, TRUE);
00614       input_save_json (generator);
00615
00616       // Saving the JSON file
00617       json_generator_to_file (generator, filename, NULL);
00618
00619       // Freeing memory
00620       g_object_unref (generator);
00621     }
00622
00623 #if DEBUG_INTERFACE
00624   fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }
```

Here is the call graph for this function:



4.11.2.2 input_save_direction_json()

```
void input_save_direction_json (
              JsonNode * node )
```

Function to save the direction search method data in a JSON node.

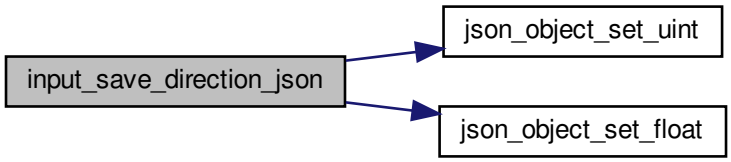
Parameters

node	JSON node.

Definition at line 201 of file interface.c.

```
00202 {
00203   JsonObject *object;
00204 #if DEBUG_INTERFACE
00205   fprintf (stderr, "input_save_direction_json: start\n");
00206 #endif
00207   object = json_node_get_object (node);
00208   if (input->nsteps)
00209     {
00210       json_object_set_uint (object, LABEL_NSTEPS,
      input->nsteps);
00211       if (input->relaxation != DEFAULT_RELAXATION)
00212         json_object_set_float (object, LABEL_RELAXATION,
      input->relaxation);
00213       switch (input->direction)
00214         {
00215         case DIRECTION_METHOD_COORDINATES:
00216           json_object_set_string_member (object, LABEL_DIRECTION,
00217                                          LABEL_COORDINATES);
00218           break;
00219         default:
00220           json_object_set_string_member (object, LABEL_DIRECTION,
      LABEL_RANDOM);
00221           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00222         }
00223     }
00224 #if DEBUG_INTERFACE
00225   fprintf (stderr, "input_save_direction_json: end\n");
00226 #endif
00227 }
```

Here is the call graph for this function:



4.11.2.3 input_save_direction_xml()

```
void input_save_direction_xml (
              xmlNode * node )
```

Function to save the direction search method data in a XML node.

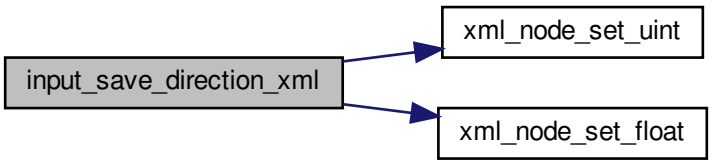
Parameters

node	XML node.

Definition at line 168 of file interface.c.

```
00169 {
00170 #if DEBUG_INTERFACE
00171   fprintf (stderr, "input_save_direction_xml: start\n");
00172 #endif
00173   if (input->nsteps)
00174     {
00175       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00176       if (input->relaxation != DEFAULT_RELAXATION)
00177         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00178                             input->relaxation);
00179       switch (input->direction)
00180         {
00181         case DIRECTION_METHOD_COORDINATES:
00182           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00183                       (const xmlChar *) LABEL_COORDINATES);
00184           break;
00185         default:
00186           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00187                       (const xmlChar *) LABEL_RANDOM);
00188           xml_node_set_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00189                              input->nestimates);
00190         }
00191     }
00192 #if DEBUG_INTERFACE
00193   fprintf (stderr, "input_save_direction_xml: end\n");
00194 #endif
00195 }
```

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.

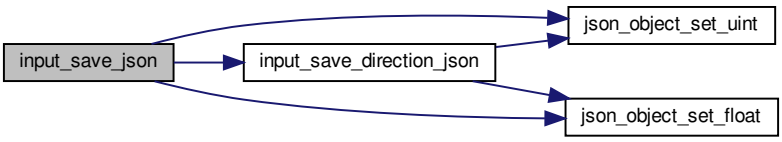
Definition at line 413 of file interface.c.

```
00414 {
00415   unsigned int i, j;
00416   char *buffer;
00417   JsonNode *node, *child;
00418   JsonObject *object;
00419   JsonArray *array;
00420   GFile *file, *file2;
00421
00422 #if DEBUG_INTERFACE
00423   fprintf (stderr, "input_save_json: start\n");
00424 #endif
00425
00426   // Setting root JSON node
00427   node = json_node_new (JSON_NODE_OBJECT);
00428   object = json_node_get_object (node);
00429   json_generator_set_root (generator, node);
00430
00431   // Adding properties to the root JSON node
00432   if (strcmp (input->result, result_name))
00433     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00434   if (strcmp (input->variables, variables_name))
00435     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00436                                    input->variables);
00437   file = g_file_new_for_path (input->directory);
00438   file2 = g_file_new_for_path (input->simulator);
00439   buffer = g_file_get_relative_path (file, file2);
00440   g_object_unref (file2);
00441   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00442   g_free (buffer);
00443   if (input->evaluator)
00444     {
00445       file2 = g_file_new_for_path (input->evaluator);
00446       buffer = g_file_get_relative_path (file, file2);
00447       g_object_unref (file2);
00448       if (strlen (buffer))
00449         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00450       g_free (buffer);
00451     }
00452   if (input->seed != DEFAULT_RANDOM_SEED)
00453     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00454
00455   // Setting the algorithm
00456   buffer = (char *) g_slice_alloc (64);
00457   switch (input->algorithm)
00458     {
00459     case ALGORITHM_MONTE_CARLO:
00460       json_object_set_string_member (object, LABEL_ALGORITHM,
00461                                      LABEL_MONTE_CARLO);
00462       snprintf (buffer, 64, "%u", input->nsimulations);
00463       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00464       snprintf (buffer, 64, "%u", input->niterations);
00465       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00466       snprintf (buffer, 64, "%.3lg", input->tolerance);
00467       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00468       snprintf (buffer, 64, "%u", input->nbest);
00469       json_object_set_string_member (object, LABEL_NBEST, buffer);
00470       input_save_direction_json (node);
00471       break;
00472     case ALGORITHM_SWEEP:
00473       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00474       snprintf (buffer, 64, "%u", input->niterations);
00475       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00476       snprintf (buffer, 64, "%.3lg", input->tolerance);
00477       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00478       snprintf (buffer, 64, "%u", input->nbest);
00479       json_object_set_string_member (object, LABEL_NBEST, buffer);
00480       input_save_direction_json (node);
00481       break;
00482     case ALGORITHM_ORTHOGONAL:
00483       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_ORTHOGONAL);
00484       snprintf (buffer, 64, "%u", input->niterations);
00485       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
```

```
00486       snprintf (buffer, 64, "%.3lg", input->tolerance);
00487       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00488       snprintf (buffer, 64, "%u", input->nbest);
00489       json_object_set_string_member (object, LABEL_NBEST, buffer);
00490       input_save_direction_json (node);
00491       break;
00492     default:
00493       json_object_set_string_member (object, LABEL_ALGORITHM,
    LABEL_GENETIC);
00494       snprintf (buffer, 64, "%u", input->nsimulations);
00495       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00496       snprintf (buffer, 64, "%u", input->niterations);
00497       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00498       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00499       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00500       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00501       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00502       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00503       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00504       break;
00505     }
00506   g_slice_free1 (64, buffer);
00507   if (input->threshold != 0.)
00508     json_object_set_float (object, LABEL_THRESHOLD,
    input->threshold);
00509
00510   // Setting the experimental data
00511   array = json_array_new ();
00512   for (i = 0; i < input->nexperiments; ++i)
00513     {
00514       child = json_node_new (JSON_NODE_OBJECT);
00515       object = json_node_get_object (child);
00516       json_object_set_string_member (object, LABEL_NAME,
00517                                      input->experiment[i].name);
00518       if (input->experiment[i].weight != 1.)
00519         json_object_set_float (object, LABEL_WEIGHT,
00520                                input->experiment[i].weight);
00521       for (j = 0; j < input->experiment->ninputs; ++j)
00522         json_object_set_string_member (object, stencil[j],
00523                                        input->experiment[i].
    stencil[j]);
00524       json_array_add_element (array, child);
00525     }
00526   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00527
00528   // Setting the variables data
00529   array = json_array_new ();
00530   for (i = 0; i < input->nvariables; ++i)
00531     {
00532       child = json_node_new (JSON_NODE_OBJECT);
00533       object = json_node_get_object (child);
00534       json_object_set_string_member (object, LABEL_NAME,
00535                                      input->variable[i].name);
00536       json_object_set_float (object, LABEL_MINIMUM,
00537                              input->variable[i].rangemin);
00538       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00539         json_object_set_float (object,
    LABEL_ABSOLUTE_MINIMUM,
00540                                input->variable[i].rangeminabs);
00541       json_object_set_float (object, LABEL_MAXIMUM,
00542                              input->variable[i].rangemax);
00543       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00544         json_object_set_float (object,
    LABEL_ABSOLUTE_MAXIMUM,
00545                                input->variable[i].rangemaxabs);
00546       if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00547         json_object_set_uint (object, LABEL_PRECISION,
00548                               input->variable[i].precision);
00549       if (input->algorithm == ALGORITHM_SWEEP
00550           || input->algorithm == ALGORITHM_ORTHOGONAL)
00551         json_object_set_uint (object, LABEL_NSWEEPS,
00552                               input->variable[i].nsweeps);
00553       else if (input->algorithm == ALGORITHM_GENETIC)
00554         json_object_set_uint (object, LABEL_NBITS,
    input->variable[i].nbits);
00555       if (input->nsteps)
00556         json_object_set_float (object, LABEL_STEP,
    input->variable[i].step);
00557       json_array_add_element (array, child);
00558     }
00559   json_object_set_array_member (object, LABEL_VARIABLES, array);
00560
00561   // Saving the error norm
00562   switch (input->norm)
00563     {
00564     case ERROR_NORM_MAXIMUM:
```

```
00565       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00566       break;
00567     case ERROR_NORM_P:
00568       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00569       json_object_set_float (object, LABEL_P, input->
      p);
00570       break;
00571     case ERROR_NORM_TAXICAB:
00572       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00573     }
00574
00575 #if DEBUG_INTERFACE
00576   fprintf (stderr, "input_save_json: end\n");
00577 #endif
00578 }
```

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.

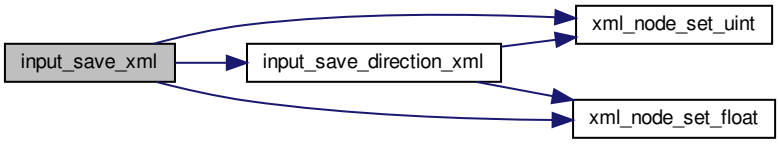
Definition at line 233 of file interface.c.

```
00234 {
00235   unsigned int i, j;
00236   char *buffer;
00237   xmlNode *node, *child;
00238   GFile *file, *file2;
00239
00240 #if DEBUG_INTERFACE
00241   fprintf (stderr, "input_save_xml: start\n");
00242 #endif
00243
00244   // Setting root XML node
00245   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00246   xmlDocSetRootElement (doc, node);
00247
00248   // Adding properties to the root XML node
00249   if (xmlStrcmp
00250       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00251     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00252                 (xmlChar *) input->result);
00253   if (xmlStrcmp
```

```
00254       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00255     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00256                 (xmlChar *) input->variables);
00257   file = g_file_new_for_path (input->directory);
00258   file2 = g_file_new_for_path (input->simulator);
00259   buffer = g_file_get_relative_path (file, file2);
00260   g_object_unref (file2);
00261   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00262   g_free (buffer);
00263   if (input->evaluator)
00264     {
00265       file2 = g_file_new_for_path (input->evaluator);
00266       buffer = g_file_get_relative_path (file, file2);
00267       g_object_unref (file2);
00268       if (xmlStrlen ((xmlChar *) buffer))
00269         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00270                     (xmlChar *) buffer);
00271       g_free (buffer);
00272     }
00273   if (input->seed != DEFAULT_RANDOM_SEED)
00274     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00275
00276   // Setting the algorithm
00277   buffer = (char *) g_slice_alloc (64);
00278   switch (input->algorithm)
00279     {
00280     case ALGORITHM_MONTE_CARLO:
00281       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00282                   (const xmlChar *) LABEL_MONTE_CARLO);
00283       snprintf (buffer, 64, "%u", input->nsimulations);
00284       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00285                   (xmlChar *) buffer);
00286       snprintf (buffer, 64, "%u", input->niterations);
00287       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00288                   (xmlChar *) buffer);
00289       snprintf (buffer, 64, "%.3lg", input->tolerance);
00290       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00291       snprintf (buffer, 64, "%u", input->nbest);
00292       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00293       input_save_direction_xml (node);
00294       break;
00295     case ALGORITHM_SWEEP:
00296       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00297                   (const xmlChar *) LABEL_SWEEP);
00298       snprintf (buffer, 64, "%u", input->niterations);
00299       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00300                   (xmlChar *) buffer);
00301       snprintf (buffer, 64, "%.3lg", input->tolerance);
00302       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00303       snprintf (buffer, 64, "%u", input->nbest);
00304       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00305       input_save_direction_xml (node);
00306       break;
00307     case ALGORITHM_ORTHOGONAL:
00308       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00309                   (const xmlChar *) LABEL_ORTHOGONAL);
00310       snprintf (buffer, 64, "%u", input->niterations);
00311       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00312                   (xmlChar *) buffer);
00313       snprintf (buffer, 64, "%.3lg", input->tolerance);
00314       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00315       snprintf (buffer, 64, "%u", input->nbest);
00316       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317       input_save_direction_xml (node);
00318       break;
00319     default:
00320       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321                   (const xmlChar *) LABEL_GENETIC);
00322       snprintf (buffer, 64, "%u", input->nsimulations);
00323       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324                   (xmlChar *) buffer);
00325       snprintf (buffer, 64, "%u", input->niterations);
00326       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327                   (xmlChar *) buffer);
00328       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332                   (xmlChar *) buffer);
00333       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00335       break;
00336     }
00337   g_slice_free1 (64, buffer);
00338   if (input->threshold != 0.)
```

```
00339     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00340                         input->threshold);
00341
00342   // Setting the experimental data
00343   for (i = 0; i < input->nexperiments; ++i)
00344     {
00345       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00346       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00347                   (xmlChar *) input->experiment[i].name);
00348       if (input->experiment[i].weight != 1.)
00349         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00350                             input->experiment[i].weight);
00351       for (j = 0; j < input->experiment->ninputs; ++j)
00352         xmlSetProp (child, (const xmlChar *) stencil[j],
00353                     (xmlChar *) input->experiment[i].stencil[j]);
00354     }
00355
00356   // Setting the variables data
00357   for (i = 0; i < input->nvariables; ++i)
00358     {
00359       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00360       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                   (xmlChar *) input->variable[i].name);
00362       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00363                           input->variable[i].rangemin);
00364       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00366                             input->variable[i].rangeminabs);
00367       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00368                           input->variable[i].rangemax);
00369       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00371                             input->variable[i].rangemaxabs);
00372       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00373         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00374                            input->variable[i].precision);
00375       if (input->algorithm == ALGORITHM_SWEEP
00376           || input->algorithm == ALGORITHM_ORTHOGONAL)
00377         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00378                            input->variable[i].nsweeps);
00379       else if (input->algorithm == ALGORITHM_GENETIC)
00380         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381                            input->variable[i].nbits);
00382       if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00384                             input->variable[i].step);
00385     }
00386
00387   // Saving the error norm
00388   switch (input->norm)
00389     {
00390     case ERROR_NORM_MAXIMUM:
00391       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                   (const xmlChar *) LABEL_MAXIMUM);
00393       break;
00394     case ERROR_NORM_P:
00395       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                   (const xmlChar *) LABEL_P);
00397       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00398       break;
00399     case ERROR_NORM_TAXICAB:
00400       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                   (const xmlChar *) LABEL_TAXICAB);
00402     }
00403
00404 #if DEBUG_INTERFACE
00405   fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }
```

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.

```
00633 {
00634 #if DEBUG_INTERFACE
00635   fprintf (stderr, "options_new: start\n");
00636 #endif
00637   options->label_seed = (GtkLabel *)
00638     gtk_label_new (_("Pseudo-random numbers generator seed"));
00639   options->spin_seed = (GtkSpinButton *)
00640     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641   gtk_widget_set_tooltip_text
00642     (GTK_WIDGET (options->spin_seed),
00643      _("Seed to init the pseudo-random numbers generator"));
00644   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00645   options->label_threads = (GtkLabel *)
00646     gtk_label_new (_("Threads number for the stochastic algorithm"));
00647   options->spin_threads
00648     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649   gtk_widget_set_tooltip_text
00650     (GTK_WIDGET (options->spin_threads),
00651      _("Number of threads to perform the calibration/optimization for "
00652        "the stochastic algorithm"));
00653   gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00654   options->label_direction = (GtkLabel *)
00655     gtk_label_new (_("Threads number for the direction search method"));
00656   options->spin_direction =
00657     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658   gtk_widget_set_tooltip_text
00659     (GTK_WIDGET (options->spin_direction),
00660      _("Number of threads to perform the calibration/optimization for the "
00661        "direction search method"));
00662   gtk_spin_button_set_value (options->spin_direction,
00663                              (gdouble) nthreads_direction);
00664   options->grid = (GtkGrid *) gtk_grid_new ();
00665   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_seed), 0, 0, 1, 1);
00666   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_seed), 1, 0, 1, 1);
00667   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_threads), 0, 1,
00668                    1, 1);
00669   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_threads), 1, 1, 1,
00670                    1);
00671   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_direction), 0, 2,
00672                    1, 1);
00673   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_direction), 1, 2,
```

```
00674                   1, 1);
00675   gtk_widget_show_all (GTK_WIDGET (options->grid));
00676   options->dialog = (GtkDialog *)
00677     gtk_dialog_new_with_buttons (_("Options"),
00678                                  window->window,
00679                                  GTK_DIALOG_MODAL,
00680                                  _("_OK"), GTK_RESPONSE_OK,
00681                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682   gtk_container_add
00683     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684      GTK_WIDGET (options->grid));
00685   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687       input->seed
00688         = (unsigned long int) gtk_spin_button_get_value (options->
      spin_seed);
00689       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690       nthreads_direction
00691         = gtk_spin_button_get_value_as_int (options->spin_direction);
00692     }
00693   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694 #if DEBUG_INTERFACE
00695   fprintf (stderr, "options_new: end\n");
00696 #endif
00697 }
```

4.11.2.7 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file interface.c.

```
00704 {
00705 #if DEBUG_INTERFACE
00706   fprintf (stderr, "running_new: start\n");
00707 #endif
00708   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710   running->grid = (GtkGrid *) gtk_grid_new ();
00711   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713   running->dialog = (GtkDialog *)
00714     gtk_dialog_new_with_buttons (_("Calculating"),
00715                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716   gtk_container_add (GTK_CONTAINER
00717                      (gtk_dialog_get_content_area (running->dialog)),
00718                      GTK_WIDGET (running->grid));
00719   gtk_spinner_start (running->spinner);
00720   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721 #if DEBUG_INTERFACE
00722   fprintf (stderr, "running_new: end\n");
00723 #endif
00724 }
```

4.11.2.8 window_about()

```
void window_about ( )
```

Function to show an about dialog.

Definition at line 1058 of file interface.c.

```
01059 {
01060   static const gchar *authors[] = {
01061     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01062     "Borja Latorre Garcés <borja.latorre@csic.es>",
01063     NULL
01064   };
01065 #if DEBUG_INTERFACE
01066   fprintf (stderr, "window_about: start\n");
01067 #endif
01068   gtk_show_about_dialog
01069     (window->window,
01070      "program_name", "MPCOTool",
01071      "comments",
01072      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01073        "A software to perform calibrations or optimizations of empirical "
01074        "parameters"),
01075      "authors", authors,
01076      "translator-credits",
01077      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01078      "(english, french and spanish)\n"
01079      "Uğur Çayoğlu (german)",
01080      "version", "3.4.5",
01081      "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01082      "logo", window->logo,
01083      "website", "https://github.com/jburguete/mpcotool",
01084      "license-type", GTK_LICENSE_BSD, NULL);
01085 #if DEBUG_INTERFACE
01086   fprintf (stderr, "window_about: end\n");
01087 #endif
01088 }
```

4.11.2.9 window_add_experiment()

```
void window_add_experiment ( )
```

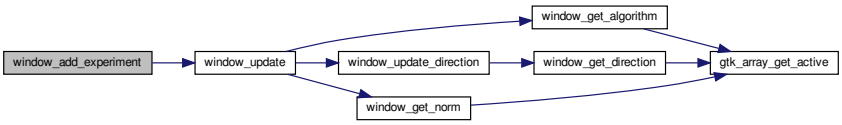
Function to add an experiment in the main window.

Definition at line 1394 of file interface.c.

```
01395 {
01396   unsigned int i, j;
01397 #if DEBUG_INTERFACE
01398   fprintf (stderr, "window_add_experiment: start\n");
01399 #endif
01400   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01401   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01402   gtk_combo_box_text_insert_text
01403     (window->combo_experiment, i, input->experiment[i].
      name);
01404   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01405   input->experiment = (Experiment *) g_realloc
01406     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01407   for (j = input->nexperiments - 1; j > i; --j)
01408     memcpy (input->experiment + j + 1, input->experiment + j,
01409             sizeof (Experiment));
01410   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01411   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01412   if (input->type == INPUT_TYPE_XML)
01413     {
01414       input->experiment[j + 1].name
01415         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01416       for (j = 0; j < input->experiment->ninputs; ++j)
01417         input->experiment[i + 1].stencil[j]
01418           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      stencil[j]);
01419     }
01420   else
01421     {
01422       input->experiment[j + 1].name = g_strdup (input->
```

```
      experiment[j].name);
01423       for (j = 0; j < input->experiment->ninputs; ++j)
01424         input->experiment[i + 1].stencil[j]
01425           = g_strdup (input->experiment[i].stencil[j]);
01426     }
01427   ++input->nexperiments;
01428   for (j = 0; j < input->experiment->ninputs; ++j)
01429     g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01430   g_signal_handler_block
01431     (window->button_experiment, window->
      id_experiment_name);
01432   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01433   g_signal_handler_unblock
01434     (window->button_experiment, window->
      id_experiment_name);
01435   for (j = 0; j < input->experiment->ninputs; ++j)
01436     g_signal_handler_unblock (window->button_template[j],
      window->id_input[j]);
01437   window_update ();
01438 #if DEBUG_INTERFACE
01439   fprintf (stderr, "window_add_experiment: end\n");
01440 #endif
01441 }
```

Here is the call graph for this function:



4.11.2.10 window_add_variable()

```
void window_add_variable ( )
```

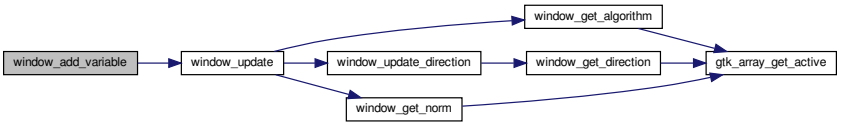
Function to add a variable in the main window.

Definition at line 1657 of file interface.c.

```
01658 {
01659   unsigned int i, j;
01660 #if DEBUG_INTERFACE
01661   fprintf (stderr, "window_add_variable: start\n");
01662 #endif
01663   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01664   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01665   gtk_combo_box_text_insert_text (window->combo_variable, i,
01666                                   input->variable[i].name);
01667   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01668   input->variable = (Variable *) g_realloc
01669     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01670   for (j = input->nvariables - 1; j > i; --j)
01671     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01672   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01673   if (input->type == INPUT_TYPE_XML)
01674     input->variable[j + 1].name
01675       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01676   else
01677     input->variable[j + 1].name = g_strdup (input->
```

```
      variable[j].name);
01678   ++input->nvariables;
01679   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01680   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01681   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01682   window_update ();
01683 #if DEBUG_INTERFACE
01684   fprintf (stderr, "window_add_variable: end\n");
01685 #endif
01686 }
```

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.

```
00733 {
00734   unsigned int i;
00735 #if DEBUG_INTERFACE
00736   fprintf (stderr, "window_get_algorithm: start\n");
00737 #endif
00738   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00739 #if DEBUG_INTERFACE
00740   fprintf (stderr, "window_get_algorithm: %u\n", i);
00741   fprintf (stderr, "window_get_algorithm: end\n");
00742 #endif
00743   return i;
00744 }
```

Here is the call graph for this function:

4.11.2.12 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 752 of file interface.c.

```
00753 {
00754   unsigned int i;
00755 #if DEBUG_INTERFACE
00756   fprintf (stderr, "window_get_direction: start\n");
00757 #endif
00758   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00759 #if DEBUG_INTERFACE
00760   fprintf (stderr, "window_get_direction: %u\n", i);
00761   fprintf (stderr, "window_get_direction: end\n");
00762 #endif
00763   return i;
00764 }
```

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.

```
00773 {
00774   unsigned int i;
00775 #if DEBUG_INTERFACE
00776   fprintf (stderr, "window_get_norm: start\n");
00777 #endif
00778   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00779 #if DEBUG_INTERFACE
00780   fprintf (stderr, "window_get_norm: %u\n", i);
00781   fprintf (stderr, "window_get_norm: end\n");
00782 #endif
00783   return i;
00784 }
```

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 1030 of file interface.c.

```
01031 {
01032   char *buffer, *buffer2;
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_help: start\n");
01035 #endif
01036   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01037                               _("user-manual.pdf"), NULL);
01038   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01039   g_free (buffer2);
01040 #if GTK_MINOR_VERSION >= 22
01041   gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01042 #else
01043   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01044 #endif
01045 #if DEBUG_INTERFACE
01046   fprintf (stderr, "window_help: uri=%s\n", buffer);
01047 #endif
01048   g_free (buffer);
01049 #if DEBUG_INTERFACE
01050   fprintf (stderr, "window_help: end\n");
01051 #endif
01052 }
```

4.11.2.15 window_inputs_experiment()

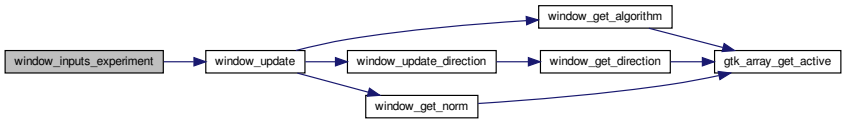
```
void window_inputs_experiment ( )
```

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

Definition at line 1494 of file interface.c.

```
01495 {
01496   unsigned int j;
01497 #if DEBUG_INTERFACE
01498   fprintf (stderr, "window_inputs_experiment: start\n");
01499 #endif
01500   j = input->experiment->ninputs - 1;
01501   if (j
01502       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01503                                         (window->check_template[j])))
01504     --input->experiment->ninputs;
01505   if (input->experiment->ninputs < MAX_NINPUTS
01506       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01507                                        (window->check_template[j])))
01508     ++input->experiment->ninputs;
01509   window_update ();
01510 #if DEBUG_INTERFACE
01511   fprintf (stderr, "window_inputs_experiment: end\n");
01512 #endif
01513 }
```

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 1692 of file interface.c.

```
01693 {
01694   unsigned int i;
01695   const char *buffer;
01696 #if DEBUG_INTERFACE
01697   fprintf (stderr, "window_label_variable: start\n");
01698 #endif
01699   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01700   buffer = gtk_entry_get_text (window->entry_variable);
01701   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01702   gtk_combo_box_text_remove (window->combo_variable, i);
01703   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01704   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01705   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01706 #if DEBUG_INTERFACE
01707   fprintf (stderr, "window_label_variable: end\n");
01708 #endif
01709 }
```

4.11.2.17 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1447 of file interface.c.

```
01448 {
01449   unsigned int i;
01450   char *buffer;
01451   GFile *file1, *file2;
01452 #if DEBUG_INTERFACE
01453   fprintf (stderr, "window_name_experiment: start\n");
01454 #endif
01455   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01456   file1
01457     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
      button_experiment));
01458   file2 = g_file_new_for_path (input->directory);
01459   buffer = g_file_get_relative_path (file2, file1);
01460   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01461   gtk_combo_box_text_remove (window->combo_experiment, i);
01462   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01463   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01464   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01465   g_free (buffer);
01466   g_object_unref (file2);
01467   g_object_unref (file1);
01468 #if DEBUG_INTERFACE
01469   fprintf (stderr, "window_name_experiment: end\n");
01470 #endif
01471 }
```

4.11.2.18 window_new()

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

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2067 of file interface.c.

```
02068 {
02069   unsigned int i;
02070   char *buffer, *buffer2, buffer3[64];
02071   char *label_algorithm[NALGORITHMS] = {
02072     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02073   };
02074   char *tip_algorithm[NALGORITHMS] = {
02075     _("Monte-Carlo brute force algorithm"),
02076     _("Sweep brute force algorithm"),
02077     _("Genetic algorithm"),
02078     _("Orthogonal sampling brute force algorithm"),
02079   };
02080   char *label_direction[NDIRECTIONS] = {
02081     _("_Coordinates descent"), _("_Random")
02082   };
```

```
02083   char *tip_direction[NDIRECTIONS] = {
02084     _("Coordinates direction estimate method"),
02085     _("Random direction estimate method")
02086   };
02087   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02088   char *tip_norm[NNORMS] = {
02089     _("Euclidean error norm (L2)"),
02090     _("Maximum error norm (L)"),
02091     _("P error norm (Lp)"),
02092     _("Taxicab error norm (L1)")
02093   };
02094
02095 #if DEBUG_INTERFACE
02096   fprintf (stderr, "window_new: start\n");
02097 #endif
02098
02099   // Creating the window
02100   window->window = main_window
02101     = (GtkWindow *) gtk_application_window_new (application);
02102
02103   // Finish when closing the window
02104   g_signal_connect_swapped (window->window, "delete-event",
02105                             G_CALLBACK (g_application_quit),
02106                             G_APPLICATION (application));
02107
02108   // Setting the window title
02109   gtk_window_set_title (window->window, "MPCOTool");
02110
02111   // Creating the open button
02112   window->button_open = (GtkToolButton *) gtk_tool_button_new
02113     (gtk_image_new_from_icon_name ("document-open",
02114                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02115   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02116
02117   // Creating the save button
02118   window->button_save = (GtkToolButton *) gtk_tool_button_new
02119     (gtk_image_new_from_icon_name ("document-save",
02120                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02121   g_signal_connect (window->button_save, "clicked", (GCallback)
      window_save,
02122                     NULL);
02123
02124   // Creating the run button
02125   window->button_run = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("system-run",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02128   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02129
02130   // Creating the options button
02131   window->button_options = (GtkToolButton *) gtk_tool_button_new
02132     (gtk_image_new_from_icon_name ("preferences-system",
02133                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02134   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02135
02136   // Creating the help button
02137   window->button_help = (GtkToolButton *) gtk_tool_button_new
02138     (gtk_image_new_from_icon_name ("help-browser",
02139                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02140   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02141
02142   // Creating the about button
02143   window->button_about = (GtkToolButton *) gtk_tool_button_new
02144     (gtk_image_new_from_icon_name ("help-about",
02145                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02146   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02147
02148   // Creating the exit button
02149   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02150     (gtk_image_new_from_icon_name ("application-exit",
02151                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02152   g_signal_connect_swapped (window->button_exit, "clicked",
02153                             G_CALLBACK (g_application_quit),
02154                             G_APPLICATION (application));
02155
02156   // Creating the buttons bar
02157   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02158   gtk_toolbar_insert
02159     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02160   gtk_toolbar_insert
02161     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02162   gtk_toolbar_insert
02163     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
```

```
02164   gtk_toolbar_insert
02165     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02166   gtk_toolbar_insert
02167     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02168   gtk_toolbar_insert
02169     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02170   gtk_toolbar_insert
02171     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02172   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02173
02174   // Creating the simulator program label and entry
02175   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02176   window->button_simulator = (GtkFileChooserButton *)
02177     gtk_file_chooser_button_new (_("Simulator program"),
02178                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02179   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02180                                _("Simulator program executable file"));
02181   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02182
02183   // Creating the evaluator program label and entry
02184   window->check_evaluator = (GtkCheckButton *)
02185     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02186   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02187   window->button_evaluator = (GtkFileChooserButton *)
02188     gtk_file_chooser_button_new (_("Evaluator program"),
02189                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02190   gtk_widget_set_tooltip_text
02191     (GTK_WIDGET (window->button_evaluator),
02192      _("Optional evaluator program executable file"));
02193
02194   // Creating the results files labels and entries
02195   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02196   window->entry_result = (GtkEntry *) gtk_entry_new ();
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->entry_result), _("Best results file"));
02199   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02200   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02201   gtk_widget_set_tooltip_text
02202     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02203
02204   // Creating the files grid and attaching widgets
02205   window->grid_files = (GtkGrid *) gtk_grid_new ();
02206   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02207                    0, 0, 1, 1);
02208   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02209                    1, 0, 1, 1);
02210   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02211                    0, 1, 1, 1);
02212   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02213                    1, 1, 1, 1);
02214   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02215                    0, 2, 1, 1);
02216   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02217                    1, 2, 1, 1);
02218   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02219                    0, 3, 1, 1);
02220   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02221                    1, 3, 1, 1);
02222
02223   // Creating the algorithm properties
02224   window->label_simulations = (GtkLabel *) gtk_label_new
02225     (_("Simulations number"));
02226   window->spin_simulations
02227     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02228   gtk_widget_set_tooltip_text
02229     (GTK_WIDGET (window->spin_simulations),
02230      _("Number of simulations to perform for each iteration"));
02231   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02232   window->label_iterations = (GtkLabel *)
02233     gtk_label_new (_("Iterations number"));
02234   window->spin_iterations
02235     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02236   gtk_widget_set_tooltip_text
02237     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
```

```
02238   g_signal_connect
02239     (window->spin_iterations, "value-changed",
      window_update, NULL);
02240   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02241   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02242   window->spin_tolerance =
02243     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02244   gtk_widget_set_tooltip_text
02245     (GTK_WIDGET (window->spin_tolerance),
02246      _("Tolerance to set the variable interval on the next iteration"));
02247   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02248   window->spin_bests
02249     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02250   gtk_widget_set_tooltip_text
02251     (GTK_WIDGET (window->spin_bests),
02252      _("Number of best simulations used to set the variable interval "
02253        "on the next iteration"));
02254   window->label_population
02255     = (GtkLabel *) gtk_label_new (_("Population number"));
02256   window->spin_population
02257     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02258   gtk_widget_set_tooltip_text
02259     (GTK_WIDGET (window->spin_population),
02260      _("Number of population for the genetic algorithm"));
02261   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02262   window->label_generations
02263     = (GtkLabel *) gtk_label_new (_("Generations number"));
02264   window->spin_generations
02265     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02266   gtk_widget_set_tooltip_text
02267     (GTK_WIDGET (window->spin_generations),
02268      _("Number of generations for the genetic algorithm"));
02269   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02270   window->spin_mutation
02271     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02272   gtk_widget_set_tooltip_text
02273     (GTK_WIDGET (window->spin_mutation),
02274      _("Ratio of mutation for the genetic algorithm"));
02275   window->label_reproduction
02276     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02277   window->spin_reproduction
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279   gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_reproduction),
02281      _("Ratio of reproduction for the genetic algorithm"));
02282   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02283   window->spin_adaptation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285   gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_adaptation),
02287      _("Ratio of adaptation for the genetic algorithm"));
02288   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02289   window->spin_threshold = (GtkSpinButton *)
02290     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02291                                     precision[DEFAULT_PRECISION]);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_threshold),
02294      _("Threshold in the objective function to finish the simulations"));
02295   window->scrolled_threshold =
02296     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02297   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02298                      GTK_WIDGET (window->spin_threshold));
02299 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02300 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02301 //                         GTK_ALIGN_FILL);
02302
02303   // Creating the direction search method properties
02304   window->check_direction = (GtkCheckButton *)
02305     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02306   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02307   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02308   window->button_direction[0] = (GtkRadioButton *)
02309     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02310   gtk_grid_attach (window->grid_direction,
02311                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02312   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02313                     NULL);
02314   for (i = 0; ++i < NDIRECTIONS;)
02315     {
02316       window->button_direction[i] = (GtkRadioButton *)
02317         gtk_radio_button_new_with_mnemonic
02318         (gtk_radio_button_get_group (window->button_direction[0]),
02319          label_direction[i]);
02320       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02321                                    tip_direction[i]);
```

```
02322       gtk_grid_attach (window->grid_direction,
02323                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02324       g_signal_connect (window->button_direction[i], "clicked",
02325                         window_update, NULL);
02326     }
02327   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02328   window->spin_steps = (GtkSpinButton *)
02329     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02330   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02331   window->label_estimates
02332     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02333   window->spin_estimates = (GtkSpinButton *)
02334     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02335   window->label_relaxation
02336     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02337   window->spin_relaxation = (GtkSpinButton *)
02338     gtk_spin_button_new_with_range (0., 2., 0.001);
02339   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->label_steps),
02340                    0, NDIRECTIONS, 1, 1);
02341   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->spin_steps),
02342                    1, NDIRECTIONS, 1, 1);
02343   gtk_grid_attach (window->grid_direction,
02344                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02345                    1, 1);
02346   gtk_grid_attach (window->grid_direction,
02347                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02348                    1);
02349   gtk_grid_attach (window->grid_direction,
02350                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02351                    1, 1);
02352   gtk_grid_attach (window->grid_direction,
02353                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02354                    1, 1);
02355
02356   // Creating the array of algorithms
02357   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02358   window->button_algorithm[0] = (GtkRadioButton *)
02359     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02360   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02361                                tip_algorithm[0]);
02362   gtk_grid_attach (window->grid_algorithm,
02363                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02364   g_signal_connect (window->button_algorithm[0], "clicked",
02365                     window_set_algorithm, NULL);
02366   for (i = 0; ++i < NALGORITHMS;)
02367     {
02368       window->button_algorithm[i] = (GtkRadioButton *)
02369         gtk_radio_button_new_with_mnemonic
02370         (gtk_radio_button_get_group (window->button_algorithm[0]),
02371          label_algorithm[i]);
02372       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02373                                    tip_algorithm[i]);
02374       gtk_grid_attach (window->grid_algorithm,
02375                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02376       g_signal_connect (window->button_algorithm[i], "clicked",
02377                         window_set_algorithm, NULL);
02378     }
02379   gtk_grid_attach (window->grid_algorithm,
02380                    GTK_WIDGET (window->label_simulations),
02381                    0, NALGORITHMS, 1, 1);
02382   gtk_grid_attach (window->grid_algorithm,
02383                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02384   gtk_grid_attach (window->grid_algorithm,
02385                    GTK_WIDGET (window->label_iterations),
02386                    0, NALGORITHMS + 1, 1, 1);
02387   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_iterations),
02388                    1, NALGORITHMS + 1, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_tolerance),
02390                    0, NALGORITHMS + 2, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_tolerance),
02392                    1, NALGORITHMS + 2, 1, 1);
02393   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests), 0,
02394                    NALGORITHMS + 3, 1, 1);
02395   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02396                    NALGORITHMS + 3, 1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_population),
02399                    0, NALGORITHMS + 4, 1, 1);
02400   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_population),
```

```
02401                     1, NALGORITHMS + 4, 1, 1);
02402   gtk_grid_attach (window->grid_algorithm,
02403                    GTK_WIDGET (window->label_generations),
02404                    0, NALGORITHMS + 5, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->spin_generations),
02407                    1, NALGORITHMS + 5, 1, 1);
02408   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_mutation),
02409                    0, NALGORITHMS + 6, 1, 1);
02410   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02411                    1, NALGORITHMS + 6, 1, 1);
02412   gtk_grid_attach (window->grid_algorithm,
02413                    GTK_WIDGET (window->label_reproduction),
02414                    0, NALGORITHMS + 7, 1, 1);
02415   gtk_grid_attach (window->grid_algorithm,
02416                    GTK_WIDGET (window->spin_reproduction),
02417                    1, NALGORITHMS + 7, 1, 1);
02418   gtk_grid_attach (window->grid_algorithm,
02419                    GTK_WIDGET (window->label_adaptation),
02420                    0, NALGORITHMS + 8, 1, 1);
02421   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_adaptation),
02422                    1, NALGORITHMS + 8, 1, 1);
02423   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->check_direction),
02424                    0, NALGORITHMS + 9, 2, 1);
02425   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->grid_direction),
02426                    0, NALGORITHMS + 10, 2, 1);
02427   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_threshold),
02428                    0, NALGORITHMS + 11, 1, 1);
02429   gtk_grid_attach (window->grid_algorithm,
02430                    GTK_WIDGET (window->scrolled_threshold),
02431                    1, NALGORITHMS + 11, 1, 1);
02432   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02433   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02434                      GTK_WIDGET (window->grid_algorithm));
02435
02436   // Creating the variable widgets
02437   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02438   gtk_widget_set_tooltip_text
02439     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02440   window->id_variable = g_signal_connect
02441     (window->combo_variable, "changed", window_set_variable, NULL);
02442   window->button_add_variable
02443     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02444                                                    GTK_ICON_SIZE_BUTTON);
02445   g_signal_connect
02446     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02447   gtk_widget_set_tooltip_text
02448     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02449   window->button_remove_variable
02450     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02451                                                    GTK_ICON_SIZE_BUTTON);
02452   g_signal_connect
02453     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02456   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02457   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02458   gtk_widget_set_tooltip_text
02459     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02460   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02461   window->id_variable_label = g_signal_connect
02462     (window->entry_variable, "changed",
      window_label_variable, NULL);
02463   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02464   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02465     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02466   gtk_widget_set_tooltip_text
02467     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02468   window->scrolled_min
02469     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02470   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02471                      GTK_WIDGET (window->spin_min));
02472   g_signal_connect (window->spin_min, "value-changed",
02473                     window_rangemin_variable, NULL);
02474   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02475   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02476     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
```

```
02479   window->scrolled_max
02480     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02481   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02482                      GTK_WIDGET (window->spin_max));
02483   g_signal_connect (window->spin_max, "value-changed",
02484                     window_rangemax_variable, NULL);
02485   window->check_minabs = (GtkCheckButton *)
02486     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02487   g_signal_connect (window->check_minabs, "toggled",
      window_update, NULL);
02488   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02489     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02490   gtk_widget_set_tooltip_text
02491     (GTK_WIDGET (window->spin_minabs),
02492      _("Minimum allowed value of the variable"));
02493   window->scrolled_minabs
02494     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02495   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02496                      GTK_WIDGET (window->spin_minabs));
02497   g_signal_connect (window->spin_minabs, "value-changed",
02498                     window_rangeminabs_variable, NULL);
02499   window->check_maxabs = (GtkCheckButton *)
02500     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02501   g_signal_connect (window->check_maxabs, "toggled",
      window_update, NULL);
02502   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02503     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02504   gtk_widget_set_tooltip_text
02505     (GTK_WIDGET (window->spin_maxabs),
02506      _("Maximum allowed value of the variable"));
02507   window->scrolled_maxabs
02508     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02509   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02510                      GTK_WIDGET (window->spin_maxabs));
02511   g_signal_connect (window->spin_maxabs, "value-changed",
02512                     window_rangemaxabs_variable, NULL);
02513   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02514   window->spin_precision = (GtkSpinButton *)
02515     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02516   gtk_widget_set_tooltip_text
02517     (GTK_WIDGET (window->spin_precision),
02518      _("Number of precision floating point digits\n"
02519        "0 is for integer numbers"));
02520   g_signal_connect (window->spin_precision, "value-changed",
02521                     window_precision_variable, NULL);
02522   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02523   window->spin_sweeps =
02524     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02525   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02526                                _("Number of steps sweeping the variable"));
02527   g_signal_connect (window->spin_sweeps, "value-changed",
02528                     window_update_variable, NULL);
02529   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02530   window->spin_bits
02531     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_bits),
02534      _("Number of bits to encode the variable"));
02535   g_signal_connect
02536     (window->spin_bits, "value-changed", window_update_variable, NULL)
    ;
02537   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02538   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02539     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02540   gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_step),
02542      _("Initial step size for the direction search method"));
02543   window->scrolled_step
02544     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02545   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02546                      GTK_WIDGET (window->spin_step));
02547   g_signal_connect
02548     (window->spin_step, "value-changed", window_step_variable, NULL);
02549   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02550   gtk_grid_attach (window->grid_variable,
02551                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02552   gtk_grid_attach (window->grid_variable,
02553                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02554   gtk_grid_attach (window->grid_variable,
02555                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02556   gtk_grid_attach (window->grid_variable,
02557                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02558   gtk_grid_attach (window->grid_variable,
02559                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02560   gtk_grid_attach (window->grid_variable,
02561                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02562   gtk_grid_attach (window->grid_variable,
```

```
02563                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02564   gtk_grid_attach (window->grid_variable,
02565                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02592   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02593   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02594                      GTK_WIDGET (window->grid_variable));
02595
02596   // Creating the experiment widgets
02597   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02598   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02599                                _("Experiment selector"));
02600   window->id_experiment = g_signal_connect
02601     (window->combo_experiment, "changed",
      window_set_experiment, NULL);
02602   window->button_add_experiment
02603     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02604                                                    GTK_ICON_SIZE_BUTTON);
02605   g_signal_connect
02606     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02607   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02608                                _("Add experiment"));
02609   window->button_remove_experiment
02610     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02611                                                    GTK_ICON_SIZE_BUTTON);
02612   g_signal_connect (window->button_remove_experiment, "clicked",
02613                     window_remove_experiment, NULL);
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02615                                _("Remove experiment"));
02616   window->label_experiment
02617     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02618   window->button_experiment = (GtkFileChooserButton *)
02619     gtk_file_chooser_button_new (_("Experimental data file"),
02620                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02621   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02622                                _("Experimental data file"));
02623   window->id_experiment_name
02624     = g_signal_connect (window->button_experiment, "selection-changed",
02625                         window_name_experiment, NULL);
02626   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02627   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02628   window->spin_weight
02629     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02630   gtk_widget_set_tooltip_text
02631     (GTK_WIDGET (window->spin_weight),
02632      _("Weight factor to build the objective function"));
02633   g_signal_connect
02634     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02635   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02636   gtk_grid_attach (window->grid_experiment,
02637                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02638   gtk_grid_attach (window->grid_experiment,
02639                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02640   gtk_grid_attach (window->grid_experiment,
02641                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02642   gtk_grid_attach (window->grid_experiment,
02643                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02644   gtk_grid_attach (window->grid_experiment,
```

```
02645                   GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02646   gtk_grid_attach (window->grid_experiment,
02647                   GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02648   gtk_grid_attach (window->grid_experiment,
02649                   GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02650   for (i = 0; i < MAX_NINPUTS; ++i)
02651     {
02652       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02653       window->check_template[i] = (GtkCheckButton *)
02654         gtk_check_button_new_with_label (buffer3);
02655       window->id_template[i]
02656         = g_signal_connect (window->check_template[i], "toggled",
02657                             window_inputs_experiment, NULL);
02658       gtk_grid_attach (window->grid_experiment,
02659                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02660       window->button_template[i] =
02661         (GtkFileChooserButton *)
02662         gtk_file_chooser_button_new (_("Input template"),
02663                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02664       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02665                                    _("Experimental input template file"));
02666       window->id_input[i] =
02667         g_signal_connect_swapped (window->button_template[i],
02668                                   "selection-changed",
02669                                   (GCallback) window_template_experiment,
02670                                   (void *) (size_t) i);
02671       gtk_grid_attach (window->grid_experiment,
02672                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02673     }
02674   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02675   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02676                      GTK_WIDGET (window->grid_experiment));
02677
02678   // Creating the error norm widgets
02679   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02680   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02681   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02682                      GTK_WIDGET (window->grid_norm));
02683   window->button_norm[0] = (GtkRadioButton *)
02684     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02685   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02686                                tip_norm[0]);
02687   gtk_grid_attach (window->grid_norm,
02688                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02689   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02690   for (i = 0; ++i < NNORMS;)
02691     {
02692       window->button_norm[i] = (GtkRadioButton *)
02693         gtk_radio_button_new_with_mnemonic
02694         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02695       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02696                                    tip_norm[i]);
02697       gtk_grid_attach (window->grid_norm,
02698                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02699       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02700     }
02701   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02702   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1, 1);
02703   window->spin_p =
02704     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02705                                                       G_MAXDOUBLE, 0.01);
02706   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02707                                _("P parameter for the P error norm"));
02708   window->scrolled_p =
02709     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02710   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02711                      GTK_WIDGET (window->spin_p));
02712   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02713   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02714   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      scrolled_p),
02715                    1, 2, 1, 2);
02716
02717   // Creating the grid and attaching the widgets to the grid
02718   window->grid = (GtkGrid *) gtk_grid_new ();
02719   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02720   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02721   gtk_grid_attach (window->grid,
02722                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02723   gtk_grid_attach (window->grid,
02724                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02725   gtk_grid_attach (window->grid,
02726                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02727   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
```

```
02728   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
      window->grid));
02729
02730   // Setting the window logo
02731   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02732   gtk_window_set_icon (window->window, window->logo);
02733
02734   // Showing the window
02735   gtk_widget_show_all (GTK_WIDGET (window->window));
02736
02737   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02738 #if GTK_MINOR_VERSION >= 16
02739   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02740   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02741   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02742   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02743   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02744   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02745   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02746 #endif
02747
02748   // Reading initial example
02749   input_new ();
02750   buffer2 = g_get_current_dir ();
02751   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02752   g_free (buffer2);
02753   window_read (buffer);
02754   g_free (buffer);
02755
02756 #if DEBUG_INTERFACE
02757   fprintf (stderr, "window_new: start\n");
02758 #endif
02759 }
```

4.11.2.19 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1981 of file interface.c.

```
01982 {
01983   GtkFileChooserDialog *dlg;
01984   GtkFileFilter *filter;
01985   char *buffer, *directory, *name;
01986
01987 #if DEBUG_INTERFACE
01988   fprintf (stderr, "window_open: start\n");
01989 #endif
01990
01991   // Saving a backup of the current input file
01992   directory = g_strdup (input->directory);
01993   name = g_strdup (input->name);
01994
01995   // Opening dialog
01996   dlg = (GtkFileChooserDialog *)
01997     gtk_file_chooser_dialog_new (_("Open input file"),
01998                                  window->window,
01999                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02000                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02001                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02002
02003   // Adding XML filter
02004   filter = (GtkFileFilter *) gtk_file_filter_new ();
02005   gtk_file_filter_set_name (filter, "XML");
02006   gtk_file_filter_add_pattern (filter, "*.xml");
02007   gtk_file_filter_add_pattern (filter, "*.XML");
02008   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02009
02010   // Adding JSON filter
02011   filter = (GtkFileFilter *) gtk_file_filter_new ();
02012   gtk_file_filter_set_name (filter, "JSON");
02013   gtk_file_filter_add_pattern (filter, "*.json");
02014   gtk_file_filter_add_pattern (filter, "*.JSON");
```

```
02015   gtk_file_filter_add_pattern (filter, "*.js");
02016   gtk_file_filter_add_pattern (filter, "*.JS");
02017   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019   // If OK saving
02020   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02021     {
02022
02023       // Traying to open the input file
02024       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02025       if (!window_read (buffer))
02026         {
02027 #if DEBUG_INTERFACE
02028           fprintf (stderr, "window_open: error reading input file\n");
02029 #endif
02030           g_free (buffer);
02031
02032           // Reading backup file on error
02033           buffer = g_build_filename (directory, name, NULL);
02034           input->result = input->variables = NULL;
02035           if (!input_open (buffer))
02036             {
02037
02038               // Closing on backup file reading error
02039 #if DEBUG_INTERFACE
02040               fprintf (stderr, "window_read: error reading backup file\n");
02041 #endif
02042               g_free (buffer);
02043               break;
02044             }
02045           g_free (buffer);
02046         }
02047       else
02048         {
02049           g_free (buffer);
02050           break;
02051         }
02052     }
02053
02054   // Freeing and closing
02055   g_free (name);
02056   g_free (directory);
02057   gtk_widget_destroy (GTK_WIDGET (dlg));
02058 #if DEBUG_INTERFACE
02059   fprintf (stderr, "window_open: end\n");
02060 #endif
02061 }
```

4.11.2.20 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1715 of file interface.c.

```
01716 {
01717   unsigned int i;
01718 #if DEBUG_INTERFACE
01719   fprintf (stderr, "window_precision_variable: start\n");
01720 #endif
01721   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01722   input->variable[i].precision
01723     = (unsigned int) gtk_spin_button_get_value_as_int (window->
      spin_precision);
01724   gtk_spin_button_set_digits (window->spin_min, input->
      variable[i].precision);
01725   gtk_spin_button_set_digits (window->spin_max, input->
      variable[i].precision);
01726   gtk_spin_button_set_digits (window->spin_minabs,
01727                               input->variable[i].precision);
01728   gtk_spin_button_set_digits (window->spin_maxabs,
01729                               input->variable[i].precision);
01730 #if DEBUG_INTERFACE
01731   fprintf (stderr, "window_precision_variable: end\n");
01732 #endif
01733 }
```

4.11.2.21 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1756 of file interface.c.

```
01757 {
01758   unsigned int i;
01759 #if DEBUG_INTERFACE
01760   fprintf (stderr, "window_rangemax_variable: start\n");
01761 #endif
01762   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01763   input->variable[i].rangemax = gtk_spin_button_get_value (
      window->spin_max);
01764 #if DEBUG_INTERFACE
01765   fprintf (stderr, "window_rangemax_variable: end\n");
01766 #endif
01767 }
```

4.11.2.22 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1791 of file interface.c.

```
01792 {
01793   unsigned int i;
01794 #if DEBUG_INTERFACE
01795   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01796 #endif
01797   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01798   input->variable[i].rangemaxabs
01799     = gtk_spin_button_get_value (window->spin_maxabs);
01800 #if DEBUG_INTERFACE
01801   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01802 #endif
01803 }
```

4.11.2.23 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1739 of file interface.c.

```
01740 {
01741   unsigned int i;
01742 #if DEBUG_INTERFACE
01743   fprintf (stderr, "window_rangemin_variable: start\n");
01744 #endif
01745   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01746   input->variable[i].rangemin = gtk_spin_button_get_value (
      window->spin_min);
01747 #if DEBUG_INTERFACE
01748   fprintf (stderr, "window_rangemin_variable: end\n");
01749 #endif
01750 }
```

4.11.2.24 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1773 of file interface.c.

```
01774 {
01775   unsigned int i;
01776 #if DEBUG_INTERFACE
01777   fprintf (stderr, "window_rangeminabs_variable: start\n");
01778 #endif
01779   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01780   input->variable[i].rangeminabs
01781     = gtk_spin_button_get_value (window->spin_minabs);
01782 #if DEBUG_INTERFACE
01783   fprintf (stderr, "window_rangeminabs_variable: end\n");
01784 #endif
01785 }
```

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.

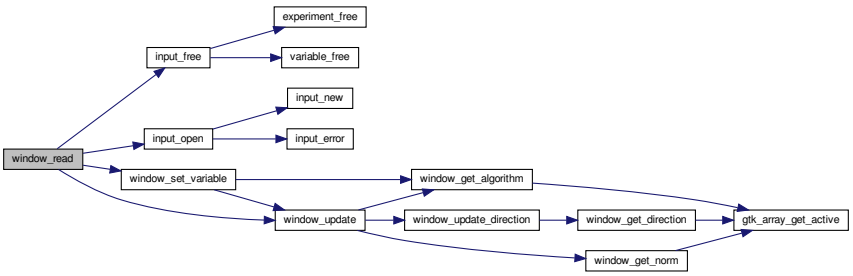
Definition at line 1865 of file interface.c.

```
01866 {
01867   unsigned int i;
01868   char *buffer;
01869 #if DEBUG_INTERFACE
01870   fprintf (stderr, "window_read: start\n");
01871 #endif
01872
01873   // Reading new input file
01874   input_free ();
01875   input->result = input->variables = NULL;
01876   if (!input_open (filename))
01877     {
01878 #if DEBUG_INTERFACE
01879       fprintf (stderr, "window_read: end\n");
01880 #endif
01881       return 0;
01882     }
01883
01884   // Setting GTK+ widgets data
01885   gtk_entry_set_text (window->entry_result, input->result);
01886   gtk_entry_set_text (window->entry_variables, input->
      variables);
```

```
01887   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01888   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01889                                  (window->button_simulator), buffer);
01890   g_free (buffer);
01891   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01892                                 (size_t) input->evaluator);
01893   if (input->evaluator)
01894     {
01895       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01896       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01897                                      (window->button_evaluator), buffer);
01898       g_free (buffer);
01899     }
01900   gtk_toggle_button_set_active
01901     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01902   switch (input->algorithm)
01903     {
01904     case ALGORITHM_MONTE_CARLO:
01905       gtk_spin_button_set_value (window->spin_simulations,
01906                                  (gdouble) input->nsimulations);
01907       // fallthrough
01908     case ALGORITHM_SWEEP:
01909     case ALGORITHM_ORTHOGONAL:
01910       gtk_spin_button_set_value (window->spin_iterations,
01911                                  (gdouble) input->niterations);
01912       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01913       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01914       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01915                                     (window->check_direction),
      input->nsteps);
01916       if (input->nsteps)
01917         {
01918           gtk_toggle_button_set_active
01919             (GTK_TOGGLE_BUTTON (window->button_direction
01920                                 [input->direction]), TRUE);
01921           gtk_spin_button_set_value (window->spin_steps,
01922                                      (gdouble) input->nsteps);
01923           gtk_spin_button_set_value (window->spin_relaxation,
01924                                      (gdouble) input->relaxation);
01925           switch (input->direction)
01926             {
01927             case DIRECTION_METHOD_RANDOM:
01928               gtk_spin_button_set_value (window->spin_estimates,
01929                                          (gdouble) input->nestimates);
01930             }
01931         }
01932       break;
01933     default:
01934       gtk_spin_button_set_value (window->spin_population,
01935                                  (gdouble) input->nsimulations);
01936       gtk_spin_button_set_value (window->spin_generations,
01937                                  (gdouble) input->niterations);
01938       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01939       gtk_spin_button_set_value (window->spin_reproduction,
01940                                  input->reproduction_ratio);
01941       gtk_spin_button_set_value (window->spin_adaptation,
01942                                  input->adaptation_ratio);
01943     }
01944   gtk_toggle_button_set_active
01945     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01946   gtk_spin_button_set_value (window->spin_p, input->p);
01947   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01948   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01949   g_signal_handler_block (window->button_experiment,
01950                           window->id_experiment_name);
01951   gtk_combo_box_text_remove_all (window->combo_experiment);
01952   for (i = 0; i < input->nexperiments; ++i)
01953     gtk_combo_box_text_append_text (window->combo_experiment,
01954                                     input->experiment[i].name);
01955   g_signal_handler_unblock
01956     (window->button_experiment, window->
      id_experiment_name);
01957   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01958   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01959   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01960   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
```

```
01961   gtk_combo_box_text_remove_all (window->combo_variable);
01962   for (i = 0; i < input->nvariables; ++i)
01963     gtk_combo_box_text_append_text (window->combo_variable,
01964                                     input->variable[i].name);
01965   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01966   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01967   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01968   window_set_variable ();
01969   window_update ();
01970
01971 #if DEBUG_INTERFACE
01972   fprintf (stderr, "window_read: end\n");
01973 #endif
01974   return 1;
01975 }
```

Here is the call graph for this function:



4.11.2.26 window_remove_experiment()

```
void window_remove_experiment ( )
```

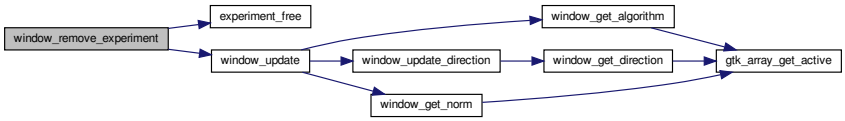
Function to remove an experiment in the main window.

Definition at line 1357 of file interface.c.

```
01358 {
01359   unsigned int i, j;
01360 #if DEBUG_INTERFACE
01361   fprintf (stderr, "window_remove_experiment: start\n");
01362 #endif
01363   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01364   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01365   gtk_combo_box_text_remove (window->combo_experiment, i);
01366   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01367   experiment_free (input->experiment + i, input->
      type);
01368   --input->nexperiments;
01369   for (j = i; j < input->nexperiments; ++j)
01370     memcpy (input->experiment + j, input->experiment + j + 1,
01371             sizeof (Experiment));
01372   j = input->nexperiments - 1;
01373   if (i > j)
01374     i = j;
01375   for (j = 0; j < input->experiment->ninputs; ++j)
01376     g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01377   g_signal_handler_block
01378     (window->button_experiment, window->
```

```
      id_experiment_name);
01379   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01380   g_signal_handler_unblock
01381     (window->button_experiment, window->
      id_experiment_name);
01382   for (j = 0; j < input->experiment->ninputs; ++j)
01383     g_signal_handler_unblock (window->button_template[j],
      window->id_input[j]);
01384   window_update ();
01385 #if DEBUG_INTERFACE
01386   fprintf (stderr, "window_remove_experiment: end\n");
01387 #endif
01388 }
```

Here is the call graph for this function:



4.11.2.27 window_remove_variable()

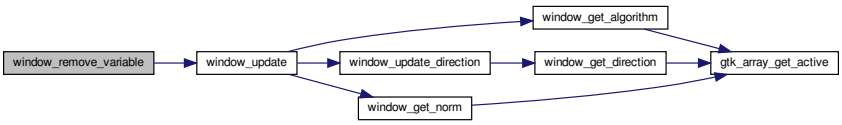
```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

Definition at line 1627 of file interface.c.

```
01628 {
01629   unsigned int i, j;
01630 #if DEBUG_INTERFACE
01631   fprintf (stderr, "window_remove_variable: start\n");
01632 #endif
01633   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01634   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01635   gtk_combo_box_text_remove (window->combo_variable, i);
01636   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01637   xmlFree (input->variable[i].name);
01638   --input->nvariables;
01639   for (j = i; j < input->nvariables; ++j)
01640     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01641   j = input->nvariables - 1;
01642   if (i > j)
01643     i = j;
01644   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01645   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01646   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01647   window_update ();
01648 #if DEBUG_INTERFACE
01649   fprintf (stderr, "window_remove_variable: end\n");
01650 #endif
01651 }
```

Here is the call graph for this function:



4.11.2.28 window_run()

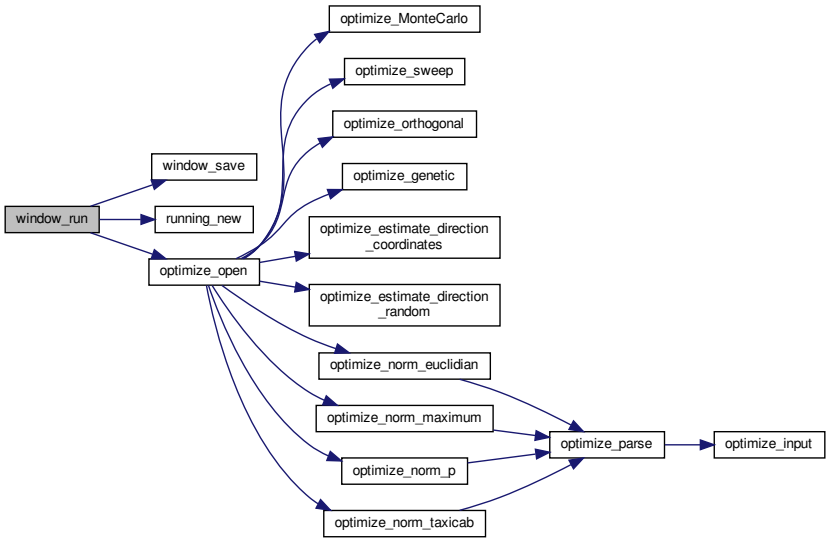
```
void window_run ( )
```

Function to run a optimization.

Definition at line 975 of file interface.c.

```
00976 {
00977   unsigned int i;
00978   char *msg, *msg2, buffer[64], buffer2[64];
00979 #if DEBUG_INTERFACE
00980   fprintf (stderr, "window_run: start\n");
00981 #endif
00982   if (!window_save ())
00983     {
00984 #if DEBUG_INTERFACE
00985       fprintf (stderr, "window_run: end\n");
00986 #endif
00987       return;
00988     }
00989   running_new ();
00990   while (gtk_events_pending ())
00991     gtk_main_iteration ();
00992   optimize_open ();
00993 #if DEBUG_INTERFACE
00994   fprintf (stderr, "window_run: closing running dialog\n");
00995 #endif
00996   gtk_spinner_stop (running->spinner);
00997   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00998 #if DEBUG_INTERFACE
00999   fprintf (stderr, "window_run: displaying results\n");
01000 #endif
01001   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01002   msg2 = g_strdup (buffer);
01003   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01004     {
01005       snprintf (buffer, 64, "%s = %s\n",
01006                 input->variable[i].name, format[input->
      variable[i].precision]);
01007       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008       msg = g_strconcat (msg2, buffer2, NULL);
01009       g_free (msg2);
01010     }
01011   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01012             optimize->calculation_time);
01013   msg = g_strconcat (msg2, buffer, NULL);
01014   g_free (msg2);
01015   show_message (_("Best result"), msg, INFO_TYPE);
01016   g_free (msg);
01017 #if DEBUG_INTERFACE
01018   fprintf (stderr, "window_run: freeing memory\n");
01019 #endif
01020   optimize_free ();
01021 #if DEBUG_INTERFACE
01022   fprintf (stderr, "window_run: end\n");
01023 #endif
01024 }
```

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 824 of file interface.c.

```
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (_("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00840                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00841   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00842   buffer = g_build_filename (input->directory, input->name, NULL);
00843   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00844   g_free (buffer);
00845
00846   // Adding XML filter
00847   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00848   gtk_file_filter_set_name (filter1, "XML");
```

```
00849   gtk_file_filter_add_pattern (filter1, "*.xml");
00850   gtk_file_filter_add_pattern (filter1, "*.XML");
00851   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00852
00853   // Adding JSON filter
00854   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00855   gtk_file_filter_set_name (filter2, "JSON");
00856   gtk_file_filter_add_pattern (filter2, "*.json");
00857   gtk_file_filter_add_pattern (filter2, "*.JSON");
00858   gtk_file_filter_add_pattern (filter2, "*.js");
00859   gtk_file_filter_add_pattern (filter2, "*.JS");
00860   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00861
00862   if (input->type == INPUT_TYPE_XML)
00863     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00864   else
00865     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00866
00867   // If OK response then saving
00868   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00869     {
00870       // Setting input file type
00871       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00872       buffer = (char *) gtk_file_filter_get_name (filter1);
00873       if (!strcmp (buffer, "XML"))
00874         input->type = INPUT_TYPE_XML;
00875       else
00876         input->type = INPUT_TYPE_JSON;
00877
00878       // Adding properties to the root XML node
00879       input->simulator = gtk_file_chooser_get_filename
00880         (GTK_FILE_CHOOSER (window->button_simulator));
00881       if (gtk_toggle_button_get_active
00882           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00883         input->evaluator = gtk_file_chooser_get_filename
00884           (GTK_FILE_CHOOSER (window->button_evaluator));
00885       else
00886         input->evaluator = NULL;
00887       if (input->type == INPUT_TYPE_XML)
00888         {
00889           input->result
00890             = (char *) xmlStrdup ((const xmlChar *)
00891                                   gtk_entry_get_text (window->entry_result));
00892           input->variables
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->
      entry_variables));
00895         }
00896       else
00897         {
00898           input->result = g_strdup (gtk_entry_get_text (window->
      entry_result));
00899           input->variables =
00900             g_strdup (gtk_entry_get_text (window->entry_variables));
00901         }
00902
00903       // Setting the algorithm
00904       switch (window_get_algorithm ())
00905         {
00906         case ALGORITHM_MONTE_CARLO:
00907           input->algorithm = ALGORITHM_MONTE_CARLO;
00908           input->nsimulations
00909             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00910           input->niterations
00911             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00912           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00913           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00914           window_save_direction ();
00915           break;
00916         case ALGORITHM_SWEEP:
00917           input->algorithm = ALGORITHM_SWEEP;
00918           input->niterations
00919             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00920           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00921           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00922           window_save_direction ();
00923           break;
00924         case ALGORITHM_ORTHOGONAL:
00925           input->algorithm = ALGORITHM_ORTHOGONAL;
00926           input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00928           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
```

```
00929           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00930           window_save_direction ();
00931           break;
00932         default:
00933           input->algorithm = ALGORITHM_GENETIC;
00934           input->nsimulations
00935             = gtk_spin_button_get_value_as_int (window->spin_population);
00936           input->niterations
00937             = gtk_spin_button_get_value_as_int (window->spin_generations);
00938           input->mutation_ratio
00939             = gtk_spin_button_get_value (window->spin_mutation);
00940           input->reproduction_ratio
00941             = gtk_spin_button_get_value (window->spin_reproduction);
00942           input->adaptation_ratio
00943             = gtk_spin_button_get_value (window->spin_adaptation);
00944           break;
00945         }
00946       input->norm = window_get_norm ();
00947       input->p = gtk_spin_button_get_value (window->spin_p);
00948       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00949
00950       // Saving the XML file
00951       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00952       input_save (buffer);
00953
00954       // Closing and freeing memory
00955       g_free (buffer);
00956       gtk_widget_destroy (GTK_WIDGET (dlg));
00957 #if DEBUG_INTERFACE
00958       fprintf (stderr, "window_save: end\n");
00959 #endif
00960       return 1;
00961     }
00962
00963   // Closing and freeing memory
00964   gtk_widget_destroy (GTK_WIDGET (dlg));
00965 #if DEBUG_INTERFACE
00966   fprintf (stderr, "window_save: end\n");
00967 #endif
00968   return 0;
00969 }
```

4.11.2.30 window_save_direction()

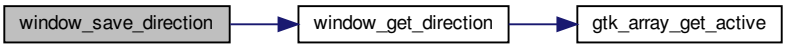
```
void window_save_direction ( )
```

Function to save the direction search method data in the input file.

Definition at line 790 of file interface.c.

```
00791 {
00792 #if DEBUG_INTERFACE
00793   fprintf (stderr, "window_save_direction: start\n");
00794 #endif
00795   if (gtk_toggle_button_get_active
00796       (GTK_TOGGLE_BUTTON (window->check_direction)))
00797     {
00798       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00799       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00800       switch (window_get_direction ())
00801         {
00802         case DIRECTION_METHOD_COORDINATES:
00803           input->direction = DIRECTION_METHOD_COORDINATES;
00804           break;
00805         default:
00806           input->direction = DIRECTION_METHOD_RANDOM;
00807           input->nestimates
00808             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00809         }
00810     }
00811   else
00812     input->nsteps = 0;
00813 #if DEBUG_INTERFACE
00814   fprintf (stderr, "window_save_direction: end\n");
00815 #endif
00816 }
```

Here is the call graph for this function:



4.11.2.31 window_set_algorithm()

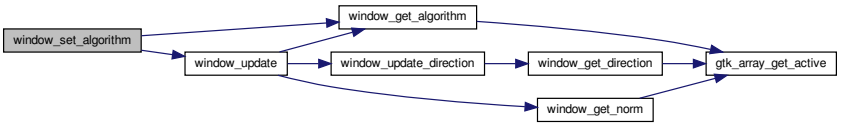
```
void window_set_algorithm ( )
```

Function to avoid memory errors changing the algorithm.

Definition at line 1283 of file interface.c.

```
01284 {
01285   int i;
01286 #if DEBUG_INTERFACE
01287   fprintf (stderr, "window_set_algorithm: start\n");
01288 #endif
01289   i = window_get_algorithm ();
01290   switch (i)
01291     {
01292     case ALGORITHM_SWEEP:
01293     case ALGORITHM_ORTHOGONAL:
01294       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01295       if (i < 0)
01296         i = 0;
01297       gtk_spin_button_set_value (window->spin_sweeps,
01298                                  (gdouble) input->variable[i].
      nsweeps);
01299       break;
01300     case ALGORITHM_GENETIC:
01301       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01302       if (i < 0)
01303         i = 0;
01304       gtk_spin_button_set_value (window->spin_bits,
01305                                  (gdouble) input->variable[i].nbits);
01306     }
01307   window_update ();
01308 #if DEBUG_INTERFACE
01309   fprintf (stderr, "window_set_algorithm: end\n");
01310 #endif
01311 }
```

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 1317 of file interface.c.

```
01318 {
01319   unsigned int i, j;
01320   char *buffer1, *buffer2;
01321 #if DEBUG_INTERFACE
01322   fprintf (stderr, "window_set_experiment: start\n");
01323 #endif
01324   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01325   gtk_spin_button_set_value (window->spin_weight, input->
      experiment[i].weight);
01326   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01327   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01328   g_free (buffer1);
01329   g_signal_handler_block
01330     (window->button_experiment, window->
      id_experiment_name);
01331   gtk_file_chooser_set_filename
01332     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01333   g_signal_handler_unblock
01334     (window->button_experiment, window->
      id_experiment_name);
01335   g_free (buffer2);
01336   for (j = 0; j < input->experiment->ninputs; ++j)
01337     {
01338       g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01339       buffer2 =
01340         g_build_filename (input->directory, input->experiment[i].
      stencil[j],
01341                           NULL);
01342       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01343                                      (window->button_template[j]), buffer2);
01344       g_free (buffer2);
01345       g_signal_handler_unblock
01346         (window->button_template[j], window->id_input[j]);
01347     }
01348 #if DEBUG_INTERFACE
01349   fprintf (stderr, "window_set_experiment: end\n");
01350 #endif
01351 }
```

4.11.2.33 window_set_variable()

```
void window_set_variable ( )
```

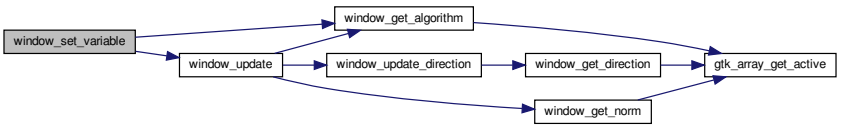
Function to set the variable data in the main window.

Definition at line 1550 of file interface.c.

```
01551 {
01552   unsigned int i;
01553 #if DEBUG_INTERFACE
01554   fprintf (stderr, "window_set_variable: start\n");
01555 #endif
01556   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01557   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01558   gtk_entry_set_text (window->entry_variable, input->
      variable[i].name);
01559   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01560   gtk_spin_button_set_value (window->spin_min, input->
      variable[i].rangemin);
```

```
01561   gtk_spin_button_set_value (window->spin_max, input->
      variable[i].rangemax);
01562   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01563     {
01564       gtk_spin_button_set_value (window->spin_minabs,
01565                                  input->variable[i].rangeminabs);
01566       gtk_toggle_button_set_active
01567         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01568     }
01569   else
01570     {
01571       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01572       gtk_toggle_button_set_active
01573         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01574     }
01575   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01576     {
01577       gtk_spin_button_set_value (window->spin_maxabs,
01578                                  input->variable[i].rangemaxabs);
01579       gtk_toggle_button_set_active
01580         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01581     }
01582   else
01583     {
01584       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01585       gtk_toggle_button_set_active
01586         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01587     }
01588   gtk_spin_button_set_value (window->spin_precision,
01589                              input->variable[i].precision);
01590   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01591   if (input->nsteps)
01592     gtk_spin_button_set_value (window->spin_step, input->
      variable[i].step);
01593 #if DEBUG_INTERFACE
01594   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01595            input->variable[i].precision);
01596 #endif
01597   switch (window_get_algorithm ())
01598     {
01599     case ALGORITHM_SWEEP:
01600     case ALGORITHM_ORTHOGONAL:
01601       gtk_spin_button_set_value (window->spin_sweeps,
01602                                  (gdouble) input->variable[i].
      nsweeps);
01603 #if DEBUG_INTERFACE
01604       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605                input->variable[i].nsweeps);
01606 #endif
01607       break;
01608     case ALGORITHM_GENETIC:
01609       gtk_spin_button_set_value (window->spin_bits,
01610                                  (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613                input->variable[i].nbits);
01614 #endif
01615       break;
01616     }
01617   window_update ();
01618 #if DEBUG_INTERFACE
01619   fprintf (stderr, "window_set_variable: end\n");
01620 #endif
01621 }
```

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 1809 of file interface.c.

```
01810 {
01811   unsigned int i;
01812 #if DEBUG_INTERFACE
01813   fprintf (stderr, "window_step_variable: start\n");
01814 #endif
01815   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01816   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01817 #if DEBUG_INTERFACE
01818   fprintf (stderr, "window_step_variable: end\n");
01819 #endif
01820 }
```

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 1519 of file interface.c.

```
01521 {
01522   unsigned int i, j;
01523   char *buffer;
01524   GFile *file1, *file2;
01525 #if DEBUG_INTERFACE
01526   fprintf (stderr, "window_template_experiment: start\n");
01527 #endif
01528   i = (size_t) data;
01529   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01530   file1
01531     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01532   file2 = g_file_new_for_path (input->directory);
01533   buffer = g_file_get_relative_path (file2, file1);
01534   if (input->type == INPUT_TYPE_XML)
01535     input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01536   else
01537     input->experiment[j].stencil[i] = g_strdup (buffer);
01538   g_free (buffer);
01539   g_object_unref (file2);
01540   g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }
```

4.11.2.36 window_update()

```
void window_update ( )
```

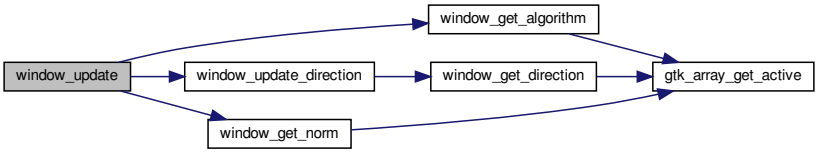
Function to update the main window view.

Definition at line 1126 of file interface.c.

```
01127 {
01128   unsigned int i;
01129 #if DEBUG_INTERFACE
01130   fprintf (stderr, "window_update: start\n");
01131 #endif
01132   gtk_widget_set_sensitive
01133     (GTK_WIDGET (window->button_evaluator),
01134      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01135                                    (window->check_evaluator)));
01136   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01137   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01138   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01139   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01140   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01142   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01144   gtk_widget_hide (GTK_WIDGET (window->label_population));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01146   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01147   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01148   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01150   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01151   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01152   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01153   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01154   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01155   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01156   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01157   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01158   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01159   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01160   gtk_widget_hide (GTK_WIDGET (window->label_step));
01161   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01162   gtk_widget_hide (GTK_WIDGET (window->label_p));
01163   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01164   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01165   switch (window_get_algorithm ())
01166     {
01167     case ALGORITHM_MONTE_CARLO:
01168       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01169       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01170       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01171       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01172       if (i > 1)
01173         {
01174           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01175           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01176           gtk_widget_show (GTK_WIDGET (window->label_bests));
01177           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01178         }
01179       window_update_direction ();
01180       break;
01181     case ALGORITHM_SWEEP:
01182     case ALGORITHM_ORTHOGONAL:
01183       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01184       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01185       if (i > 1)
01186         {
01187           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01188           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01189           gtk_widget_show (GTK_WIDGET (window->label_bests));
01190           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01191         }
01192       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01193       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01194       gtk_widget_show (GTK_WIDGET (window->check_direction));
01195       window_update_direction ();
01196       break;
01197     default:
01198       gtk_widget_show (GTK_WIDGET (window->label_population));
01199       gtk_widget_show (GTK_WIDGET (window->spin_population));
01200       gtk_widget_show (GTK_WIDGET (window->label_generations));
```

```
01201       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01202       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01203       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01204       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01205       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01206       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01207       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01208       gtk_widget_show (GTK_WIDGET (window->label_bits));
01209       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01210     }
01211   gtk_widget_set_sensitive
01212     (GTK_WIDGET (window->button_remove_experiment),
      input->nexperiments > 1);
01213   gtk_widget_set_sensitive
01214     (GTK_WIDGET (window->button_remove_variable),
      input->nvariables > 1);
01215   for (i = 0; i < input->experiment->ninputs; ++i)
01216     {
01217       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01218       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01219       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01220       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01221       g_signal_handler_block
01222         (window->check_template[i], window->
      id_template[i]);
01223       g_signal_handler_block (window->button_template[i],
      window->id_input[i]);
01224       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01225                                     (window->check_template[i]), 1);
01226       g_signal_handler_unblock (window->button_template[i],
01227                                 window->id_input[i]);
01228       g_signal_handler_unblock (window->check_template[i],
01229                                 window->id_template[i]);
01230     }
01231   if (i > 0)
01232     {
01233       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01234       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01235                                 gtk_toggle_button_get_active
01236                                 GTK_TOGGLE_BUTTON (window->check_template
01237                                                    [i - 1]));
01238     }
01239   if (i < MAX_NINPUTS)
01240     {
01241       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01242       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01243       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01244       gtk_widget_set_sensitive
01245         (GTK_WIDGET (window->button_template[i]),
01246          gtk_toggle_button_get_active
01247          GTK_TOGGLE_BUTTON (window->check_template[i]));
01248       g_signal_handler_block
01249         (window->check_template[i], window->
      id_template[i]);
01250       g_signal_handler_block (window->button_template[i],
      window->id_input[i]);
01251       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01252                                     (window->check_template[i]), 0);
01253       g_signal_handler_unblock (window->button_template[i],
01254                                 window->id_input[i]);
01255       g_signal_handler_unblock (window->check_template[i],
01256                                 window->id_template[i]);
01257     }
01258   while (++i < MAX_NINPUTS)
01259     {
01260       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01261       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01262     }
01263   gtk_widget_set_sensitive
01264     (GTK_WIDGET (window->spin_minabs),
01265      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01266   gtk_widget_set_sensitive
01267     (GTK_WIDGET (window->spin_maxabs),
01268      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01269   if (window_get_norm () == ERROR_NORM_P)
01270     {
01271       gtk_widget_show (GTK_WIDGET (window->label_p));
01272       gtk_widget_show (GTK_WIDGET (window->spin_p));
01273     }
01274 #if DEBUG_INTERFACE
01275   fprintf (stderr, "window_update: end\n");
01276 #endif
01277 }
```

Here is the call graph for this function:



4.11.2.37 window_update_direction()

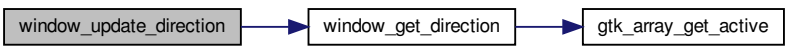
```
void window_update_direction ( )
```

Function to update direction search method widgets view in the main window.

Definition at line 1094 of file interface.c.

```
01095 {
01096 #if DEBUG_INTERFACE
01097   fprintf (stderr, "window_update_direction: start\n");
01098 #endif
01099   gtk_widget_show (GTK_WIDGET (window->check_direction));
01100   if (gtk_toggle_button_get_active
01101       (GTK_TOGGLE_BUTTON (window->check_direction)))
01102     {
01103       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01104       gtk_widget_show (GTK_WIDGET (window->label_step));
01105       gtk_widget_show (GTK_WIDGET (window->spin_step));
01106     }
01107   switch (window_get_direction ())
01108     {
01109     case DIRECTION_METHOD_COORDINATES:
01110       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01111       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01112       break;
01113     default:
01114       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01115       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01116     }
01117 #if DEBUG_INTERFACE
01118   fprintf (stderr, "window_update_direction: end\n");
01119 #endif
01120 }
```

Here is the call graph for this function:



4.11.2.38 window_update_variable()

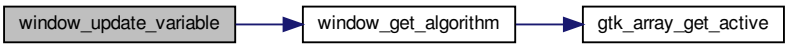
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1826 of file interface.c.

```
01827 {
01828   int i;
01829 #if DEBUG_INTERFACE
01830   fprintf (stderr, "window_update_variable: start\n");
01831 #endif
01832   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01833   if (i < 0)
01834     i = 0;
01835   switch (window_get_algorithm ())
01836     {
01837     case ALGORITHM_SWEEP:
01838     case ALGORITHM_ORTHOGONAL:
01839       input->variable[i].nsweeps
01840         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01841 #if DEBUG_INTERFACE
01842       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01843                input->variable[i].nsweeps);
01844 #endif
01845       break;
01846     case ALGORITHM_GENETIC:
01847       input->variable[i].nbits
01848         = gtk_spin_button_get_value_as_int (window->spin_bits);
01849 #if DEBUG_INTERFACE
01850       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01851                input->variable[i].nbits);
01852 #endif
01853     }
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_update_variable: end\n");
01856 #endif
01857 }
```

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 1477 of file interface.c.

```
01478 {
01479   unsigned int i;
01480 #if DEBUG_INTERFACE
01481   fprintf (stderr, "window_weight_experiment: start\n");
01482 #endif
01483   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01484   input->experiment[i].weight = gtk_spin_button_get_value (
      window->spin_weight);
01485 #if DEBUG_INTERFACE
01486   fprintf (stderr, "window_weight_experiment: end\n");
01487 #endif
01488 }
```

4.12 interface.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #include <gio/gio.h>
00057 #include <gtk/gtk.h>
00058 #include "genetic/genetic.h"
00059 #include "utils.h"
00060 #include "experiment.h"
00061 #include "variable.h"
00062 #include "input.h"
00063 #include "optimize.h"
00064 #include "interface.h"
00065
00066 #define DEBUG_INTERFACE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define INPUT_FILE "test-ga-win.xml"
00074 #else
00075 #define INPUT_FILE "test-ga.xml"
00076 #endif
00077
00078 const char *logo[] = {
00079   "32 32 3 1",
00080   "      c None",
00081   ".     c #0000FF",
00082   "+     c #FF0000",
00083   "                                ",
00084   "                                ",
00085   "                                ",
00086   "      .       .       .       ",
00087   "      .       .       .       ",
00088   "      .       .       .       ",
00089   "      .       .       .       ",
00090   "      .       .      +++      ",
00091   "      .       .     +++++     ",
00092   "      .       .     +++++     ",
00093   "      .       .     +++++     ",
```

```
00094   "    +++      .     +++    +++    ",
00095   "   +++++     .       .    +++++   ",
00096   "   +++++     .       .    +++++   ",
00097   "   +++++     .       .    +++++   ",
00098   "    +++      .       .     +++    ",
00099   "      .      .       .      .     ",
00100   "      .     +++      .      .     ",
00101   "      .    +++++     .      .     ",
00102   "      .    +++++     .      .     ",
00103   "      .    +++++     .      .     ",
00104   "      .     +++      .      .     ",
00105   "      .      .       .      .     ",
00106   "      .      .       .      .     ",
00107   "      .      .       .      .     ",
00108   "      .      .       .      .     ",
00109   "      .      .       .      .     ",
00110   "      .      .       .      .     ",
00111   "      .      .       .      .     ",
00112   "                                  ",
00113   "                                  ",
00114   "                                  "
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119 "32 32 3 1",
00120 "     c #FFFFFFFFFFFF",
00121 ".    c #00000000FFFF",
00122 "X    c #FFFF00000000",
00123 "                                  ",
00124 "                                  ",
00125 "                                  ",
00126 "     .      .      .      .      ",
00127 "     .      .      .      .      ",
00128 "     .      .      .      .      ",
00129 "     .      .      .      .      ",
00130 "     .      .     XXX     .      ",
00131 "     .      .    XXXXX    .      ",
00132 "     .      .    XXXXX    .      ",
00133 "     .      .    XXXXX    .      ",
00134 "    XXX     .     XXX    XXX     ",
00135 "   XXXXX    .      .    XXXXX    ",
00136 "   XXXXX    .      .    XXXXX    ",
00137 "   XXXXX    .      .    XXXXX    ",
00138 "    XXX     .      .     XXX     ",
00139 "     .      .      .      .      ",
00140 "     .     XXX     .      .      ",
00141 "     .    XXXXX    .      .      ",
00142 "     .    XXXXX    .      .      ",
00143 "     .    XXXXX    .      .      ",
00144 "     .     XXX     .      .      ",
00145 "     .      .      .      .      ",
00146 "     .      .      .      .      ",
00147 "     .      .      .      .      ",
00148 "     .      .      .      .      ",
00149 "     .      .      .      .      ",
00150 "     .      .      .      .      ",
00151 "     .      .      .      .      ",
00152 "                                  ",
00153 "                                  ",
00154 "                                  "};
00155 */
00156
00157 Options options[1];
00159 Running running[1];
00161 Window window[1];
00163
00167 void
00168 input_save_direction_xml (xmlNode * node)
00169 {
00170 #if DEBUG_INTERFACE
00171   fprintf (stderr, "input_save_direction_xml: start\n");
00172 #endif
00173   if (input->nsteps)
00174     {
00175       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00176       if (input->relaxation != DEFAULT_RELAXATION)
00177         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00178                             input->relaxation);
00179       switch (input->direction)
00180         {
00181         case DIRECTION_METHOD_COORDINATES:
00182           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00183                       (const xmlChar *) LABEL_COORDINATES);
00184           break;
```

```
00185         default:
00186           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00187                       (const xmlChar *) LABEL_RANDOM);
00188           xml_node_set_uint (node, (const xmlChar *)
    LABEL_NESTIMATES,
00189                              input->nestimates);
00190         }
00191     }
00192 #if DEBUG_INTERFACE
00193   fprintf (stderr, "input_save_direction_xml: end\n");
00194 #endif
00195 }
00196
00200 void
00201 input_save_direction_json (JsonNode * node)
00202 {
00203   JsonObject *object;
00204 #if DEBUG_INTERFACE
00205   fprintf (stderr, "input_save_direction_json: start\n");
00206 #endif
00207   object = json_node_get_object (node);
00208   if (input->nsteps)
00209     {
00210       json_object_set_uint (object, LABEL_NSTEPS,
    input->nsteps);
00211       if (input->relaxation != DEFAULT_RELAXATION)
00212         json_object_set_float (object, LABEL_RELAXATION,
    input->relaxation);
00213       switch (input->direction)
00214         {
00215         case DIRECTION_METHOD_COORDINATES:
00216           json_object_set_string_member (object, LABEL_DIRECTION,
00217                                          LABEL_COORDINATES);
00218           break;
00219         default:
00220           json_object_set_string_member (object, LABEL_DIRECTION,
    LABEL_RANDOM);
00221           json_object_set_uint (object, LABEL_NESTIMATES,
    input->nestimates);
00222         }
00223     }
00224 #if DEBUG_INTERFACE
00225   fprintf (stderr, "input_save_direction_json: end\n");
00226 #endif
00227 }
00228
00232 void
00233 input_save_xml (xmlDoc * doc)
00234 {
00235   unsigned int i, j;
00236   char *buffer;
00237   xmlNode *node, *child;
00238   GFile *file, *file2;
00239
00240 #if DEBUG_INTERFACE
00241   fprintf (stderr, "input_save_xml: start\n");
00242 #endif
00243
00244   // Setting root XML node
00245   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00246   xmlDocSetRootElement (doc, node);
00247
00248   // Adding properties to the root XML node
00249   if (xmlStrcmp
00250       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00251     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00252                 (xmlChar *) input->result);
00253   if (xmlStrcmp
00254       ((const xmlChar *) input->variables, (const xmlChar *)
    variables_name))
00255     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00256                 (xmlChar *) input->variables);
00257   file = g_file_new_for_path (input->directory);
00258   file2 = g_file_new_for_path (input->simulator);
00259   buffer = g_file_get_relative_path (file, file2);
00260   g_object_unref (file2);
00261   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00262   g_free (buffer);
00263   if (input->evaluator)
00264     {
00265       file2 = g_file_new_for_path (input->evaluator);
00266       buffer = g_file_get_relative_path (file, file2);
00267       g_object_unref (file2);
00268       if (xmlStrlen ((xmlChar *) buffer))
00269         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00270                     (xmlChar *) buffer);
00271       g_free (buffer);
```

```
00272     }
00273   if (input->seed != DEFAULT_RANDOM_SEED)
00274     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00275
00276   // Setting the algorithm
00277   buffer = (char *) g_slice_alloc (64);
00278   switch (input->algorithm)
00279     {
00280     case ALGORITHM_MONTE_CARLO:
00281       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00282                   (const xmlChar *) LABEL_MONTE_CARLO);
00283       snprintf (buffer, 64, "%u", input->nsimulations);
00284       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00285                   (xmlChar *) buffer);
00286       snprintf (buffer, 64, "%u", input->niterations);
00287       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00288                   (xmlChar *) buffer);
00289       snprintf (buffer, 64, "%.3lg", input->tolerance);
00290       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00291       snprintf (buffer, 64, "%u", input->nbest);
00292       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00293       input_save_direction_xml (node);
00294       break;
00295     case ALGORITHM_SWEEP:
00296       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00297                   (const xmlChar *) LABEL_SWEEP);
00298       snprintf (buffer, 64, "%u", input->niterations);
00299       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00300                   (xmlChar *) buffer);
00301       snprintf (buffer, 64, "%.3lg", input->tolerance);
00302       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00303       snprintf (buffer, 64, "%u", input->nbest);
00304       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00305       input_save_direction_xml (node);
00306       break;
00307     case ALGORITHM_ORTHOGONAL:
00308       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00309                   (const xmlChar *) LABEL_ORTHOGONAL);
00310       snprintf (buffer, 64, "%u", input->niterations);
00311       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00312                   (xmlChar *) buffer);
00313       snprintf (buffer, 64, "%.3lg", input->tolerance);
00314       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00315       snprintf (buffer, 64, "%u", input->nbest);
00316       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00317       input_save_direction_xml (node);
00318       break;
00319     default:
00320       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00321                   (const xmlChar *) LABEL_GENETIC);
00322       snprintf (buffer, 64, "%u", input->nsimulations);
00323       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00324                   (xmlChar *) buffer);
00325       snprintf (buffer, 64, "%u", input->niterations);
00326       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00327                   (xmlChar *) buffer);
00328       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00329       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00331       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00332                   (xmlChar *) buffer);
00333       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00335       break;
00336     }
00337   g_slice_free1 (64, buffer);
00338   if (input->threshold != 0.)
00339     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00340                         input->threshold);
00341
00342   // Setting the experimental data
00343   for (i = 0; i < input->nexperiments; ++i)
00344     {
00345       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00346       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00347                   (xmlChar *) input->experiment[i].name);
00348       if (input->experiment[i].weight != 1.)
00349         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00350                             input->experiment[i].weight);
00351       for (j = 0; j < input->experiment->ninputs; ++j)
00352         xmlSetProp (child, (const xmlChar *) stencil[j],
00353                     (xmlChar *) input->experiment[i].stencil[j]);
00354     }
00355
```

```
00356   // Setting the variables data
00357   for (i = 0; i < input->nvariables; ++i)
00358     {
00359       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00360       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361                   (xmlChar *) input->variable[i].name);
00362       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00363                           input->variable[i].rangemin);
00364       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00366                             input->variable[i].rangeminabs);
00367       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00368                           input->variable[i].rangemax);
00369       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00371                             input->variable[i].rangemaxabs);
00372       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00373         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00374                            input->variable[i].precision);
00375       if (input->algorithm == ALGORITHM_SWEEP
00376           || input->algorithm == ALGORITHM_ORTHOGONAL)
00377         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00378                            input->variable[i].nsweeps);
00379       else if (input->algorithm == ALGORITHM_GENETIC)
00380         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381                            input->variable[i].nbits);
00382       if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00384                             input->variable[i].step);
00385     }
00386
00387   // Saving the error norm
00388   switch (input->norm)
00389     {
00390     case ERROR_NORM_MAXIMUM:
00391       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                   (const xmlChar *) LABEL_MAXIMUM);
00393       break;
00394     case ERROR_NORM_P:
00395       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                   (const xmlChar *) LABEL_P);
00397       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00398       break;
00399     case ERROR_NORM_TAXICAB:
00400       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                   (const xmlChar *) LABEL_TAXICAB);
00402     }
00403
00404 #if DEBUG_INTERFACE
00405   fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }
00408
00412 void
00413 input_save_json (JsonGenerator * generator)
00414 {
00415   unsigned int i, j;
00416   char *buffer;
00417   JsonNode *node, *child;
00418   JsonObject *object;
00419   JsonArray *array;
00420   GFile *file, *file2;
00421
00422 #if DEBUG_INTERFACE
00423   fprintf (stderr, "input_save_json: start\n");
00424 #endif
00425
00426   // Setting root JSON node
00427   node = json_node_new (JSON_NODE_OBJECT);
00428   object = json_node_get_object (node);
00429   json_generator_set_root (generator, node);
00430
00431   // Adding properties to the root JSON node
00432   if (strcmp (input->result, result_name))
00433     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00434   if (strcmp (input->variables, variables_name))
00435     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
```

```
00436                                   input->variables);
00437   file = g_file_new_for_path (input->directory);
00438   file2 = g_file_new_for_path (input->simulator);
00439   buffer = g_file_get_relative_path (file, file2);
00440   g_object_unref (file2);
00441   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00442   g_free (buffer);
00443   if (input->evaluator)
00444     {
00445       file2 = g_file_new_for_path (input->evaluator);
00446       buffer = g_file_get_relative_path (file, file2);
00447       g_object_unref (file2);
00448       if (strlen (buffer))
00449         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00450       g_free (buffer);
00451     }
00452   if (input->seed != DEFAULT_RANDOM_SEED)
00453     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00454
00455   // Setting the algorithm
00456   buffer = (char *) g_slice_alloc (64);
00457   switch (input->algorithm)
00458     {
00459     case ALGORITHM_MONTE_CARLO:
00460       json_object_set_string_member (object, LABEL_ALGORITHM,
00461                                      LABEL_MONTE_CARLO);
00462       snprintf (buffer, 64, "%u", input->nsimulations);
00463       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00464       snprintf (buffer, 64, "%u", input->niterations);
00465       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00466       snprintf (buffer, 64, "%.3lg", input->tolerance);
00467       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00468       snprintf (buffer, 64, "%u", input->nbest);
00469       json_object_set_string_member (object, LABEL_NBEST, buffer);
00470       input_save_direction_json (node);
00471       break;
00472     case ALGORITHM_SWEEP:
00473       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00474       snprintf (buffer, 64, "%u", input->niterations);
00475       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00476       snprintf (buffer, 64, "%.3lg", input->tolerance);
00477       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00478       snprintf (buffer, 64, "%u", input->nbest);
00479       json_object_set_string_member (object, LABEL_NBEST, buffer);
00480       input_save_direction_json (node);
00481       break;
00482     case ALGORITHM_ORTHOGONAL:
00483       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_ORTHOGONAL);
00484       snprintf (buffer, 64, "%u", input->niterations);
00485       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00486       snprintf (buffer, 64, "%.3lg", input->tolerance);
00487       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00488       snprintf (buffer, 64, "%u", input->nbest);
00489       json_object_set_string_member (object, LABEL_NBEST, buffer);
00490       input_save_direction_json (node);
00491       break;
00492     default:
00493       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_GENETIC);
00494       snprintf (buffer, 64, "%u", input->nsimulations);
00495       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00496       snprintf (buffer, 64, "%u", input->niterations);
00497       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00498       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00499       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00500       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00501       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00502       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00503       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00504       break;
00505     }
00506   g_slice_free1 (64, buffer);
00507   if (input->threshold != 0.)
00508     json_object_set_float (object, LABEL_THRESHOLD,
      input->threshold);
00509
00510   // Setting the experimental data
00511   array = json_array_new ();
00512   for (i = 0; i < input->nexperiments; ++i)
00513     {
00514       child = json_node_new (JSON_NODE_OBJECT);
00515       object = json_node_get_object (child);
00516       json_object_set_string_member (object, LABEL_NAME,
00517                                      input->experiment[i].name);
```

```
00518       if (input->experiment[i].weight != 1.)
00519         json_object_set_float (object, LABEL_WEIGHT,
00520                                input->experiment[i].weight);
00521       for (j = 0; j < input->experiment->ninputs; ++j)
00522         json_object_set_string_member (object, stencil[j],
00523                                        input->experiment[i].
    stencil[j]);
00524       json_array_add_element (array, child);
00525     }
00526   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00527
00528   // Setting the variables data
00529   array = json_array_new ();
00530   for (i = 0; i < input->nvariables; ++i)
00531     {
00532       child = json_node_new (JSON_NODE_OBJECT);
00533       object = json_node_get_object (child);
00534       json_object_set_string_member (object, LABEL_NAME,
00535                                      input->variable[i].name);
00536       json_object_set_float (object, LABEL_MINIMUM,
00537                              input->variable[i].rangemin);
00538       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00539         json_object_set_float (object,
    LABEL_ABSOLUTE_MINIMUM,
00540                                input->variable[i].rangeminabs);
00541       json_object_set_float (object, LABEL_MAXIMUM,
00542                              input->variable[i].rangemax);
00543       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00544         json_object_set_float (object,
    LABEL_ABSOLUTE_MAXIMUM,
00545                                input->variable[i].rangemaxabs);
00546       if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00547         json_object_set_uint (object, LABEL_PRECISION,
00548                               input->variable[i].precision);
00549       if (input->algorithm == ALGORITHM_SWEEP
00550           || input->algorithm == ALGORITHM_ORTHOGONAL)
00551         json_object_set_uint (object, LABEL_NSWEEPS,
00552                               input->variable[i].nsweeps);
00553       else if (input->algorithm == ALGORITHM_GENETIC)
00554         json_object_set_uint (object, LABEL_NBITS,
    input->variable[i].nbits);
00555       if (input->nsteps)
00556         json_object_set_float (object, LABEL_STEP,
    input->variable[i].step);
00557       json_array_add_element (array, child);
00558     }
00559   json_object_set_array_member (object, LABEL_VARIABLES, array);
00560
00561   // Saving the error norm
00562   switch (input->norm)
00563     {
00564     case ERROR_NORM_MAXIMUM:
00565       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00566       break;
00567     case ERROR_NORM_P:
00568       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00569       json_object_set_float (object, LABEL_P, input->
    p);
00570       break;
00571     case ERROR_NORM_TAXICAB:
00572       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00573     }
00574
00575 #if DEBUG_INTERFACE
00576   fprintf (stderr, "input_save_json: end\n");
00577 #endif
00578 }
00579
00583 void
00584 input_save (char *filename)
00585 {
00586   xmlDoc *doc;
00587   JsonGenerator *generator;
00588
00589 #if DEBUG_INTERFACE
00590   fprintf (stderr, "input_save: start\n");
00591 #endif
00592
00593   // Getting the input file directory
00594   input->name = g_path_get_basename (filename);
00595   input->directory = g_path_get_dirname (filename);
00596
00597   if (input->type == INPUT_TYPE_XML)
00598     {
00599       // Opening the input file
00600       doc = xmlNewDoc ((const xmlChar *) "1.0");
```

```
00601       input_save_xml (doc);
00602
00603       // Saving the XML file
00604       xmlSaveFormatFile (filename, doc, 1);
00605
00606       // Freeing memory
00607       xmlFreeDoc (doc);
00608     }
00609   else
00610     {
00611       // Opening the input file
00612       generator = json_generator_new ();
00613       json_generator_set_pretty (generator, TRUE);
00614       input_save_json (generator);
00615
00616       // Saving the JSON file
00617       json_generator_to_file (generator, filename, NULL);
00618
00619       // Freeing memory
00620       g_object_unref (generator);
00621     }
00622
00623 #if DEBUG_INTERFACE
00624   fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }
00627
00631 void
00632 options_new ()
00633 {
00634 #if DEBUG_INTERFACE
00635   fprintf (stderr, "options_new: start\n");
00636 #endif
00637   options->label_seed = (GtkLabel *)
00638     gtk_label_new (_("Pseudo-random numbers generator seed"));
00639   options->spin_seed = (GtkSpinButton *)
00640     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641   gtk_widget_set_tooltip_text
00642     (GTK_WIDGET (options->spin_seed),
00643      _("Seed to init the pseudo-random numbers generator"));
00644   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00645   options->label_threads = (GtkLabel *)
00646     gtk_label_new (_("Threads number for the stochastic algorithm"));
00647   options->spin_threads
00648     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649   gtk_widget_set_tooltip_text
00650     (GTK_WIDGET (options->spin_threads),
00651      _("Number of threads to perform the calibration/optimization for "
00652        "the stochastic algorithm"));
00653   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00654   options->label_direction = (GtkLabel *)
00655     gtk_label_new (_("Threads number for the direction search method"));
00656   options->spin_direction =
00657     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658   gtk_widget_set_tooltip_text
00659     (GTK_WIDGET (options->spin_direction),
00660      _("Number of threads to perform the calibration/optimization for the "
00661        "direction search method"));
00662   gtk_spin_button_set_value (options->spin_direction,
00663                              (gdouble) nthreads_direction);
00664   options->grid = (GtkGrid *) gtk_grid_new ();
00665   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00666   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00667   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00668                    1, 1);
00669   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00670                    1);
00671   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00672                    1, 1);
00673   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00674                    1, 1);
00675   gtk_widget_show_all (GTK_WIDGET (options->grid));
00676   options->dialog = (GtkDialog *)
00677     gtk_dialog_new_with_buttons (_("Options"),
00678                                  window->window,
00679                                  GTK_DIALOG_MODAL,
00680                                  _("_OK"), GTK_RESPONSE_OK,
00681                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682   gtk_container_add
00683     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684      GTK_WIDGET (options->grid));
00685   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687       input->seed
00688         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
```

```
00689       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690       nthreads_direction
00691         = gtk_spin_button_get_value_as_int (options->spin_direction);
00692     }
00693   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694 #if DEBUG_INTERFACE
00695   fprintf (stderr, "options_new: end\n");
00696 #endif
00697 }
00698
00702 void
00703 running_new ()
00704 {
00705 #if DEBUG_INTERFACE
00706   fprintf (stderr, "running_new: start\n");
00707 #endif
00708   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710   running->grid = (GtkGrid *) gtk_grid_new ();
00711   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713   running->dialog = (GtkDialog *)
00714     gtk_dialog_new_with_buttons (_("Calculating"),
00715                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716   gtk_container_add (GTK_CONTAINER
00717                      (gtk_dialog_get_content_area (running->dialog)),
00718                      GTK_WIDGET (running->grid));
00719   gtk_spinner_start (running->spinner);
00720   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721 #if DEBUG_INTERFACE
00722   fprintf (stderr, "running_new: end\n");
00723 #endif
00724 }
00725
00731 unsigned int
00732 window_get_algorithm ()
00733 {
00734   unsigned int i;
00735 #if DEBUG_INTERFACE
00736   fprintf (stderr, "window_get_algorithm: start\n");
00737 #endif
00738   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00739 #if DEBUG_INTERFACE
00740   fprintf (stderr, "window_get_algorithm: %u\n", i);
00741   fprintf (stderr, "window_get_algorithm: end\n");
00742 #endif
00743   return i;
00744 }
00745
00751 unsigned int
00752 window_get_direction ()
00753 {
00754   unsigned int i;
00755 #if DEBUG_INTERFACE
00756   fprintf (stderr, "window_get_direction: start\n");
00757 #endif
00758   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00759 #if DEBUG_INTERFACE
00760   fprintf (stderr, "window_get_direction: %u\n", i);
00761   fprintf (stderr, "window_get_direction: end\n");
00762 #endif
00763   return i;
00764 }
00765
00771 unsigned int
00772 window_get_norm ()
00773 {
00774   unsigned int i;
00775 #if DEBUG_INTERFACE
00776   fprintf (stderr, "window_get_norm: start\n");
00777 #endif
00778   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00779 #if DEBUG_INTERFACE
00780   fprintf (stderr, "window_get_norm: %u\n", i);
00781   fprintf (stderr, "window_get_norm: end\n");
00782 #endif
00783   return i;
00784 }
00785
00789 void
00790 window_save_direction ()
00791 {
00792 #if DEBUG_INTERFACE
00793   fprintf (stderr, "window_save_direction: start\n");
```

```
00794 #endif
00795   if (gtk_toggle_button_get_active
00796       (GTK_TOGGLE_BUTTON (window->check_direction)))
00797     {
00798       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00799       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00800       switch (window_get_direction ())
00801         {
00802         case DIRECTION_METHOD_COORDINATES:
00803           input->direction = DIRECTION_METHOD_COORDINATES;
00804           break;
00805         default:
00806           input->direction = DIRECTION_METHOD_RANDOM;
00807           input->nestimates
00808             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00809         }
00810     }
00811   else
00812     input->nsteps = 0;
00813 #if DEBUG_INTERFACE
00814   fprintf (stderr, "window_save_direction: end\n");
00815 #endif
00816 }
00817
00823 int
00824 window_save ()
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (_("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00840                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00841   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00842   buffer = g_build_filename (input->directory, input->name, NULL);
00843   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00844   g_free (buffer);
00845
00846   // Adding XML filter
00847   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00848   gtk_file_filter_set_name (filter1, "XML");
00849   gtk_file_filter_add_pattern (filter1, "*.xml");
00850   gtk_file_filter_add_pattern (filter1, "*.XML");
00851   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00852
00853   // Adding JSON filter
00854   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00855   gtk_file_filter_set_name (filter2, "JSON");
00856   gtk_file_filter_add_pattern (filter2, "*.json");
00857   gtk_file_filter_add_pattern (filter2, "*.JSON");
00858   gtk_file_filter_add_pattern (filter2, "*.js");
00859   gtk_file_filter_add_pattern (filter2, "*.JS");
00860   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00861
00862   if (input->type == INPUT_TYPE_XML)
00863     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00864   else
00865     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00866
00867   // If OK response then saving
00868   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00869     {
00870       // Setting input file type
00871       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00872       buffer = (char *) gtk_file_filter_get_name (filter1);
00873       if (!strcmp (buffer, "XML"))
00874         input->type = INPUT_TYPE_XML;
00875       else
00876         input->type = INPUT_TYPE_JSON;
00877
00878       // Adding properties to the root XML node
00879       input->simulator = gtk_file_chooser_get_filename
00880         (GTK_FILE_CHOOSER (window->button_simulator));
00881       if (gtk_toggle_button_get_active
00882           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00883         input->evaluator = gtk_file_chooser_get_filename
```

```
00884           (GTK_FILE_CHOOSER (window->button_evaluator));
00885       else
00886         input->evaluator = NULL;
00887       if (input->type == INPUT_TYPE_XML)
00888         {
00889           input->result
00890             = (char *) xmlStrdup ((const xmlChar *)
00891                                   gtk_entry_get_text (window->entry_result));
00892           input->variables
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->entry_variables));
00895         }
00896       else
00897         {
00898           input->result = g_strdup (gtk_entry_get_text (window->
      entry_result));
00899           input->variables =
00900             g_strdup (gtk_entry_get_text (window->entry_variables));
00901         }
00902
00903       // Setting the algorithm
00904       switch (window_get_algorithm ())
00905         {
00906         case ALGORITHM_MONTE_CARLO:
00907           input->algorithm = ALGORITHM_MONTE_CARLO;
00908           input->nsimulations
00909             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00910           input->niterations
00911             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00912           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00913           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00914           window_save_direction ();
00915           break;
00916         case ALGORITHM_SWEEP:
00917           input->algorithm = ALGORITHM_SWEEP;
00918           input->niterations
00919             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00920           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00921           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00922           window_save_direction ();
00923           break;
00924         case ALGORITHM_ORTHOGONAL:
00925           input->algorithm = ALGORITHM_ORTHOGONAL;
00926           input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00928           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00929           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00930           window_save_direction ();
00931           break;
00932         default:
00933           input->algorithm = ALGORITHM_GENETIC;
00934           input->nsimulations
00935             = gtk_spin_button_get_value_as_int (window->spin_population);
00936           input->niterations
00937             = gtk_spin_button_get_value_as_int (window->spin_generations);
00938           input->mutation_ratio
00939             = gtk_spin_button_get_value (window->spin_mutation);
00940           input->reproduction_ratio
00941             = gtk_spin_button_get_value (window->spin_reproduction);
00942           input->adaptation_ratio
00943             = gtk_spin_button_get_value (window->spin_adaptation);
00944           break;
00945         }
00946       input->norm = window_get_norm ();
00947       input->p = gtk_spin_button_get_value (window->spin_p);
00948       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00949
00950       // Saving the XML file
00951       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00952       input_save (buffer);
00953
00954       // Closing and freeing memory
00955       g_free (buffer);
00956       gtk_widget_destroy (GTK_WIDGET (dlg));
00957 #if DEBUG_INTERFACE
00958       fprintf (stderr, "window_save: end\n");
00959 #endif
00960       return 1;
00961     }
00962
```

```
00963   // Closing and freeing memory
00964   gtk_widget_destroy (GTK_WIDGET (dlg));
00965 #if DEBUG_INTERFACE
00966   fprintf (stderr, "window_save: end\n");
00967 #endif
00968   return 0;
00969 }
00970
00974 void
00975 window_run ()
00976 {
00977   unsigned int i;
00978   char *msg, *msg2, buffer[64], buffer2[64];
00979 #if DEBUG_INTERFACE
00980   fprintf (stderr, "window_run: start\n");
00981 #endif
00982   if (!window_save ())
00983     {
00984 #if DEBUG_INTERFACE
00985       fprintf (stderr, "window_run: end\n");
00986 #endif
00987       return;
00988     }
00989   running_new ();
00990   while (gtk_events_pending ())
00991     gtk_main_iteration ();
00992   optimize_open ();
00993 #if DEBUG_INTERFACE
00994   fprintf (stderr, "window_run: closing running dialog\n");
00995 #endif
00996   gtk_spinner_stop (running->spinner);
00997   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00998 #if DEBUG_INTERFACE
00999   fprintf (stderr, "window_run: displaying results\n");
01000 #endif
01001   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01002   msg2 = g_strdup (buffer);
01003   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01004     {
01005       snprintf (buffer, 64, "%s = %s\n",
01006                 input->variable[i].name, format[input->
      variable[i].precision]);
01007       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008       msg = g_strconcat (msg2, buffer2, NULL);
01009       g_free (msg2);
01010     }
01011   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01012             optimize->calculation_time);
01013   msg = g_strconcat (msg2, buffer, NULL);
01014   g_free (msg2);
01015   show_message (_("Best result"), msg, INFO_TYPE);
01016   g_free (msg);
01017 #if DEBUG_INTERFACE
01018   fprintf (stderr, "window_run: freeing memory\n");
01019 #endif
01020   optimize_free ();
01021 #if DEBUG_INTERFACE
01022   fprintf (stderr, "window_run: end\n");
01023 #endif
01024 }
01025
01029 void
01030 window_help ()
01031 {
01032   char *buffer, *buffer2;
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_help: start\n");
01035 #endif
01036   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01037                               _("user-manual.pdf"), NULL);
01038   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01039   g_free (buffer2);
01040 #if GTK_MINOR_VERSION >= 22
01041   gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01042 #else
01043   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01044 #endif
01045 #if DEBUG_INTERFACE
01046   fprintf (stderr, "window_help: uri=%s\n", buffer);
01047 #endif
01048   g_free (buffer);
01049 #if DEBUG_INTERFACE
01050   fprintf (stderr, "window_help: end\n");
01051 #endif
01052 }
01053
01057 void
```

```
01058 window_about ()
01059 {
01060   static const gchar *authors[] = {
01061     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01062     "Borja Latorre Garcés <borja.latorre@csic.es>",
01063     NULL
01064   };
01065 #if DEBUG_INTERFACE
01066   fprintf (stderr, "window_about: start\n");
01067 #endif
01068   gtk_show_about_dialog
01069     (window->window,
01070      "program_name", "MPCOTool",
01071      "comments",
01072      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01073        "A software to perform calibrations or optimizations of empirical "
01074        "parameters"),
01075      "authors", authors,
01076      "translator-credits",
01077      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01078      "(english, french and spanish)\n"
01079      "Uğur Çayoğlu (german)",
01080      "version", "3.4.5",
01081      "copyright", "Copyright 2012-2018 Javier Burguete Tolosa",
01082      "logo", window->logo,
01083      "website", "https://github.com/jburguete/mpcotool",
01084      "license-type", GTK_LICENSE_BSD, NULL);
01085 #if DEBUG_INTERFACE
01086   fprintf (stderr, "window_about: end\n");
01087 #endif
01088 }
01089
01093 void
01094 window_update_direction ()
01095 {
01096 #if DEBUG_INTERFACE
01097   fprintf (stderr, "window_update_direction: start\n");
01098 #endif
01099   gtk_widget_show (GTK_WIDGET (window->check_direction));
01100   if (gtk_toggle_button_get_active
01101       (GTK_TOGGLE_BUTTON (window->check_direction)))
01102     {
01103       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01104       gtk_widget_show (GTK_WIDGET (window->label_step));
01105       gtk_widget_show (GTK_WIDGET (window->spin_step));
01106     }
01107   switch (window_get_direction ())
01108     {
01109     case DIRECTION_METHOD_COORDINATES:
01110       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01111       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01112       break;
01113     default:
01114       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01115       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01116     }
01117 #if DEBUG_INTERFACE
01118   fprintf (stderr, "window_update_direction: end\n");
01119 #endif
01120 }
01121
01125 void
01126 window_update ()
01127 {
01128   unsigned int i;
01129 #if DEBUG_INTERFACE
01130   fprintf (stderr, "window_update: start\n");
01131 #endif
01132   gtk_widget_set_sensitive
01133     (GTK_WIDGET (window->button_evaluator),
01134      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01135                                    (window->check_evaluator)));
01136   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01137   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01138   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01139   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01140   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01142   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01144   gtk_widget_hide (GTK_WIDGET (window->label_population));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01146   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01147   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01148   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01150   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
```

```
01151   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01152   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01153   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01154   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01155   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01156   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01157   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01158   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01159   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01160   gtk_widget_hide (GTK_WIDGET (window->label_step));
01161   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01162   gtk_widget_hide (GTK_WIDGET (window->label_p));
01163   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01164   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01165   switch (window_get_algorithm ())
01166     {
01167     case ALGORITHM_MONTE_CARLO:
01168       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01169       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01170       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01171       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01172       if (i > 1)
01173         {
01174           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01175           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01176           gtk_widget_show (GTK_WIDGET (window->label_bests));
01177           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01178         }
01179       window_update_direction ();
01180       break;
01181     case ALGORITHM_SWEEP:
01182     case ALGORITHM_ORTHOGONAL:
01183       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01184       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01185       if (i > 1)
01186         {
01187           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01188           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01189           gtk_widget_show (GTK_WIDGET (window->label_bests));
01190           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01191         }
01192       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01193       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01194       gtk_widget_show (GTK_WIDGET (window->check_direction));
01195       window_update_direction ();
01196       break;
01197     default:
01198       gtk_widget_show (GTK_WIDGET (window->label_population));
01199       gtk_widget_show (GTK_WIDGET (window->spin_population));
01200       gtk_widget_show (GTK_WIDGET (window->label_generations));
01201       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01202       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01203       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01204       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01205       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01206       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01207       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01208       gtk_widget_show (GTK_WIDGET (window->label_bits));
01209       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01210     }
01211   gtk_widget_set_sensitive
01212     (GTK_WIDGET (window->button_remove_experiment),
      input->nexperiments > 1);
01213   gtk_widget_set_sensitive
01214     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
01215   for (i = 0; i < input->experiment->ninputs; ++i)
01216     {
01217       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01218       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01219       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01220       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01221       g_signal_handler_block
01222         (window->check_template[i], window->id_template[i]);
01223       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01224       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01225                                     (window->check_template[i]), 1);
01226       g_signal_handler_unblock (window->button_template[i],
01227                                 window->id_input[i]);
01228       g_signal_handler_unblock (window->check_template[i],
01229                                 window->id_template[i]);
01230     }
01231   if (i > 0)
01232     {
01233       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01234       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
```

```
01235                                 gtk_toggle_button_get_active
01236                                 GTK_TOGGLE_BUTTON (window->check_template
01237                                                    [i - 1]));
01238     }
01239   if (i < MAX_NINPUTS)
01240     {
01241       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01242       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01243       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01244       gtk_widget_set_sensitive
01245         (GTK_WIDGET (window->button_template[i]),
01246          gtk_toggle_button_get_active
01247          GTK_TOGGLE_BUTTON (window->check_template[i]));
01248       g_signal_handler_block
01249         (window->check_template[i], window->id_template[i]);
01250       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01251       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01252                                     (window->check_template[i]), 0);
01253       g_signal_handler_unblock (window->button_template[i],
01254                                 window->id_input[i]);
01255       g_signal_handler_unblock (window->check_template[i],
01256                                 window->id_template[i]);
01257     }
01258   while (++i < MAX_NINPUTS)
01259     {
01260       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01261       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01262     }
01263   gtk_widget_set_sensitive
01264     (GTK_WIDGET (window->spin_minabs),
01265      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01266   gtk_widget_set_sensitive
01267     (GTK_WIDGET (window->spin_maxabs),
01268      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01269   if (window_get_norm () == ERROR_NORM_P)
01270     {
01271       gtk_widget_show (GTK_WIDGET (window->label_p));
01272       gtk_widget_show (GTK_WIDGET (window->spin_p));
01273     }
01274 #if DEBUG_INTERFACE
01275   fprintf (stderr, "window_update: end\n");
01276 #endif
01277 }
01278
01282 void
01283 window_set_algorithm ()
01284 {
01285   int i;
01286 #if DEBUG_INTERFACE
01287   fprintf (stderr, "window_set_algorithm: start\n");
01288 #endif
01289   i = window_get_algorithm ();
01290   switch (i)
01291     {
01292     case ALGORITHM_SWEEP:
01293     case ALGORITHM_ORTHOGONAL:
01294       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01295       if (i < 0)
01296         i = 0;
01297       gtk_spin_button_set_value (window->spin_sweeps,
01298                                  (gdouble) input->variable[i].
      nsweeps);
01299       break;
01300     case ALGORITHM_GENETIC:
01301       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01302       if (i < 0)
01303         i = 0;
01304       gtk_spin_button_set_value (window->spin_bits,
01305                                  (gdouble) input->variable[i].nbits);
01306     }
01307   window_update ();
01308 #if DEBUG_INTERFACE
01309   fprintf (stderr, "window_set_algorithm: end\n");
01310 #endif
01311 }
01312
01316 void
01317 window_set_experiment ()
01318 {
01319   unsigned int i, j;
01320   char *buffer1, *buffer2;
01321 #if DEBUG_INTERFACE
01322   fprintf (stderr, "window_set_experiment: start\n");
01323 #endif
01324   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01325   gtk_spin_button_set_value (window->spin_weight, input->
```

```
experiment[i].weight);
01326   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01327   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01328   g_free (buffer1);
01329   g_signal_handler_block
01330     (window->button_experiment, window->id_experiment_name);
01331   gtk_file_chooser_set_filename
01332     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01333   g_signal_handler_unblock
01334     (window->button_experiment, window->id_experiment_name);
01335   g_free (buffer2);
01336   for (j = 0; j < input->experiment->ninputs; ++j)
01337     {
01338       g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01339       buffer2 =
01340         g_build_filename (input->directory, input->experiment[i].
stencil[j],
01341                           NULL);
01342       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01343                                      (window->button_template[j]), buffer2);
01344       g_free (buffer2);
01345       g_signal_handler_unblock
01346         (window->button_template[j], window->id_input[j]);
01347     }
01348 #if DEBUG_INTERFACE
01349   fprintf (stderr, "window_set_experiment: end\n");
01350 #endif
01351 }
01352
01356 void
01357 window_remove_experiment ()
01358 {
01359   unsigned int i, j;
01360 #if DEBUG_INTERFACE
01361   fprintf (stderr, "window_remove_experiment: start\n");
01362 #endif
01363   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01364   g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01365   gtk_combo_box_text_remove (window->combo_experiment, i);
01366   g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01367   experiment_free (input->experiment + i, input->
type);
01368   --input->nexperiments;
01369   for (j = i; j < input->nexperiments; ++j)
01370     memcpy (input->experiment + j, input->experiment + j + 1,
01371             sizeof (Experiment));
01372   j = input->nexperiments - 1;
01373   if (i > j)
01374     i = j;
01375   for (j = 0; j < input->experiment->ninputs; ++j)
01376     g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01377   g_signal_handler_block
01378     (window->button_experiment, window->id_experiment_name);
01379   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01380   g_signal_handler_unblock
01381     (window->button_experiment, window->id_experiment_name);
01382   for (j = 0; j < input->experiment->ninputs; ++j)
01383     g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01384   window_update ();
01385 #if DEBUG_INTERFACE
01386   fprintf (stderr, "window_remove_experiment: end\n");
01387 #endif
01388 }
01389
01393 void
01394 window_add_experiment ()
01395 {
01396   unsigned int i, j;
01397 #if DEBUG_INTERFACE
01398   fprintf (stderr, "window_add_experiment: start\n");
01399 #endif
01400   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01401   g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01402   gtk_combo_box_text_insert_text
01403     (window->combo_experiment, i, input->experiment[i].
name);
01404   g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01405   input->experiment = (Experiment *) g_realloc
01406     (input->experiment, (input->nexperiments + 1) * sizeof (
Experiment));
```

```
01407   for (j = input->nexperiments - 1; j > i; --j)
01408     memcpy (input->experiment + j + 1, input->experiment + j,
01409             sizeof (Experiment));
01410   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01411   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01412   if (input->type == INPUT_TYPE_XML)
01413     {
01414       input->experiment[j + 1].name
01415         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01416       for (j = 0; j < input->experiment->ninputs; ++j)
01417         input->experiment[i + 1].stencil[j]
01418           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      stencil[j]);
01419     }
01420   else
01421     {
01422       input->experiment[j + 1].name = g_strdup (input->
      experiment[j].name);
01423       for (j = 0; j < input->experiment->ninputs; ++j)
01424         input->experiment[i + 1].stencil[j]
01425           = g_strdup (input->experiment[i].stencil[j]);
01426     }
01427   ++input->nexperiments;
01428   for (j = 0; j < input->experiment->ninputs; ++j)
01429     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01430   g_signal_handler_block
01431     (window->button_experiment, window->id_experiment_name);
01432   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01433   g_signal_handler_unblock
01434     (window->button_experiment, window->id_experiment_name);
01435   for (j = 0; j < input->experiment->ninputs; ++j)
01436     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01437   window_update ();
01438 #if DEBUG_INTERFACE
01439   fprintf (stderr, "window_add_experiment: end\n");
01440 #endif
01441 }
01442
01446 void
01447 window_name_experiment ()
01448 {
01449   unsigned int i;
01450   char *buffer;
01451   GFile *file1, *file2;
01452 #if DEBUG_INTERFACE
01453   fprintf (stderr, "window_name_experiment: start\n");
01454 #endif
01455   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01456   file1
01457     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01458   file2 = g_file_new_for_path (input->directory);
01459   buffer = g_file_get_relative_path (file2, file1);
01460   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01461   gtk_combo_box_text_remove (window->combo_experiment, i);
01462   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01463   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01464   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01465   g_free (buffer);
01466   g_object_unref (file2);
01467   g_object_unref (file1);
01468 #if DEBUG_INTERFACE
01469   fprintf (stderr, "window_name_experiment: end\n");
01470 #endif
01471 }
01472
01476 void
01477 window_weight_experiment ()
01478 {
01479   unsigned int i;
01480 #if DEBUG_INTERFACE
01481   fprintf (stderr, "window_weight_experiment: start\n");
01482 #endif
01483   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01484   input->experiment[i].weight = gtk_spin_button_get_value (window->
      spin_weight);
01485 #if DEBUG_INTERFACE
01486   fprintf (stderr, "window_weight_experiment: end\n");
01487 #endif
01488 }
01489
```

```
01493 void
01494 window_inputs_experiment ()
01495 {
01496   unsigned int j;
01497 #if DEBUG_INTERFACE
01498   fprintf (stderr, "window_inputs_experiment: start\n");
01499 #endif
01500   j = input->experiment->ninputs - 1;
01501   if (j
01502       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01503                                         (window->check_template[j])))
01504     --input->experiment->ninputs;
01505   if (input->experiment->ninputs < MAX_NINPUTS
01506       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01507                                        (window->check_template[j])))
01508     ++input->experiment->ninputs;
01509   window_update ();
01510 #if DEBUG_INTERFACE
01511   fprintf (stderr, "window_inputs_experiment: end\n");
01512 #endif
01513 }
01514
01518 void
01519 window_template_experiment (void *data)
01521 {
01522   unsigned int i, j;
01523   char *buffer;
01524   GFile *file1, *file2;
01525 #if DEBUG_INTERFACE
01526   fprintf (stderr, "window_template_experiment: start\n");
01527 #endif
01528   i = (size_t) data;
01529   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01530   file1
01531     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01532   file2 = g_file_new_for_path (input->directory);
01533   buffer = g_file_get_relative_path (file2, file1);
01534   if (input->type == INPUT_TYPE_XML)
01535     input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01536   else
01537     input->experiment[j].stencil[i] = g_strdup (buffer);
01538   g_free (buffer);
01539   g_object_unref (file2);
01540   g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }
01545
01549 void
01550 window_set_variable ()
01551 {
01552   unsigned int i;
01553 #if DEBUG_INTERFACE
01554   fprintf (stderr, "window_set_variable: start\n");
01555 #endif
01556   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01557   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01558   gtk_entry_set_text (window->entry_variable, input->variable[i].
      name);
01559   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01560   gtk_spin_button_set_value (window->spin_min, input->variable[i].
      rangemin);
01561   gtk_spin_button_set_value (window->spin_max, input->variable[i].
      rangemax);
01562   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01563     {
01564       gtk_spin_button_set_value (window->spin_minabs,
01565                                  input->variable[i].rangeminabs);
01566       gtk_toggle_button_set_active
01567         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01568     }
01569   else
01570     {
01571       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01572       gtk_toggle_button_set_active
01573         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01574     }
01575   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01576     {
01577       gtk_spin_button_set_value (window->spin_maxabs,
01578                                  input->variable[i].rangemaxabs);
01579       gtk_toggle_button_set_active
01580         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01581     }
```

```
01582   else
01583     {
01584       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01585       gtk_toggle_button_set_active
01586         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01587     }
01588   gtk_spin_button_set_value (window->spin_precision,
01589                              input->variable[i].precision);
01590   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01591   if (input->nsteps)
01592     gtk_spin_button_set_value (window->spin_step, input->variable[i].
      step);
01593 #if DEBUG_INTERFACE
01594   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01595            input->variable[i].precision);
01596 #endif
01597   switch (window_get_algorithm ())
01598     {
01599     case ALGORITHM_SWEEP:
01600     case ALGORITHM_ORTHOGONAL:
01601       gtk_spin_button_set_value (window->spin_sweeps,
01602                                  (gdouble) input->variable[i].
      nsweeps);
01603 #if DEBUG_INTERFACE
01604       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605                input->variable[i].nsweeps);
01606 #endif
01607       break;
01608     case ALGORITHM_GENETIC:
01609       gtk_spin_button_set_value (window->spin_bits,
01610                                  (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613                input->variable[i].nbits);
01614 #endif
01615       break;
01616     }
01617   window_update ();
01618 #if DEBUG_INTERFACE
01619   fprintf (stderr, "window_set_variable: end\n");
01620 #endif
01621 }
01622
01626 void
01627 window_remove_variable ()
01628 {
01629   unsigned int i, j;
01630 #if DEBUG_INTERFACE
01631   fprintf (stderr, "window_remove_variable: start\n");
01632 #endif
01633   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01634   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01635   gtk_combo_box_text_remove (window->combo_variable, i);
01636   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01637   xmlFree (input->variable[i].name);
01638   --input->nvariables;
01639   for (j = i; j < input->nvariables; ++j)
01640     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01641   j = input->nvariables - 1;
01642   if (i > j)
01643     i = j;
01644   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01645   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01646   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01647   window_update ();
01648 #if DEBUG_INTERFACE
01649   fprintf (stderr, "window_remove_variable: end\n");
01650 #endif
01651 }
01652
01656 void
01657 window_add_variable ()
01658 {
01659   unsigned int i, j;
01660 #if DEBUG_INTERFACE
01661   fprintf (stderr, "window_add_variable: start\n");
01662 #endif
01663   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01664   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01665   gtk_combo_box_text_insert_text (window->combo_variable, i,
```

```
01666                                   input->variable[i].name);
01667   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01668   input->variable = (Variable *) g_realloc
01669     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01670   for (j = input->nvariables - 1; j > i; --j)
01671     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01672   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01673   if (input->type == INPUT_TYPE_XML)
01674     input->variable[j + 1].name
01675       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01676   else
01677     input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
01678   ++input->nvariables;
01679   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01680   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01681   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01682   window_update ();
01683 #if DEBUG_INTERFACE
01684   fprintf (stderr, "window_add_variable: end\n");
01685 #endif
01686 }
01687
01691 void
01692 window_label_variable ()
01693 {
01694   unsigned int i;
01695   const char *buffer;
01696 #if DEBUG_INTERFACE
01697   fprintf (stderr, "window_label_variable: start\n");
01698 #endif
01699   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01700   buffer = gtk_entry_get_text (window->entry_variable);
01701   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01702   gtk_combo_box_text_remove (window->combo_variable, i);
01703   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01704   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01705   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01706 #if DEBUG_INTERFACE
01707   fprintf (stderr, "window_label_variable: end\n");
01708 #endif
01709 }
01710
01714 void
01715 window_precision_variable ()
01716 {
01717   unsigned int i;
01718 #if DEBUG_INTERFACE
01719   fprintf (stderr, "window_precision_variable: start\n");
01720 #endif
01721   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01722   input->variable[i].precision
01723     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01724   gtk_spin_button_set_digits (window->spin_min, input->variable[i].
      precision);
01725   gtk_spin_button_set_digits (window->spin_max, input->variable[i].
      precision);
01726   gtk_spin_button_set_digits (window->spin_minabs,
01727                               input->variable[i].precision);
01728   gtk_spin_button_set_digits (window->spin_maxabs,
01729                               input->variable[i].precision);
01730 #if DEBUG_INTERFACE
01731   fprintf (stderr, "window_precision_variable: end\n");
01732 #endif
01733 }
01734
01738 void
01739 window_rangemin_variable ()
01740 {
01741   unsigned int i;
01742 #if DEBUG_INTERFACE
01743   fprintf (stderr, "window_rangemin_variable: start\n");
01744 #endif
01745   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01746   input->variable[i].rangemin = gtk_spin_button_get_value (window->
      spin_min);
01747 #if DEBUG_INTERFACE
01748   fprintf (stderr, "window_rangemin_variable: end\n");
01749 #endif
```

```
01750 }
01751
01755 void
01756 window_rangemax_variable ()
01757 {
01758   unsigned int i;
01759 #if DEBUG_INTERFACE
01760   fprintf (stderr, "window_rangemax_variable: start\n");
01761 #endif
01762   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01763   input->variable[i].rangemax = gtk_spin_button_get_value (window->
      spin_max);
01764 #if DEBUG_INTERFACE
01765   fprintf (stderr, "window_rangemax_variable: end\n");
01766 #endif
01767 }
01768
01772 void
01773 window_rangeminabs_variable ()
01774 {
01775   unsigned int i;
01776 #if DEBUG_INTERFACE
01777   fprintf (stderr, "window_rangeminabs_variable: start\n");
01778 #endif
01779   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01780   input->variable[i].rangeminabs
01781     = gtk_spin_button_get_value (window->spin_minabs);
01782 #if DEBUG_INTERFACE
01783   fprintf (stderr, "window_rangeminabs_variable: end\n");
01784 #endif
01785 }
01786
01790 void
01791 window_rangemaxabs_variable ()
01792 {
01793   unsigned int i;
01794 #if DEBUG_INTERFACE
01795   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01796 #endif
01797   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01798   input->variable[i].rangemaxabs
01799     = gtk_spin_button_get_value (window->spin_maxabs);
01800 #if DEBUG_INTERFACE
01801   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01802 #endif
01803 }
01804
01808 void
01809 window_step_variable ()
01810 {
01811   unsigned int i;
01812 #if DEBUG_INTERFACE
01813   fprintf (stderr, "window_step_variable: start\n");
01814 #endif
01815   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01816   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01817 #if DEBUG_INTERFACE
01818   fprintf (stderr, "window_step_variable: end\n");
01819 #endif
01820 }
01821
01825 void
01826 window_update_variable ()
01827 {
01828   int i;
01829 #if DEBUG_INTERFACE
01830   fprintf (stderr, "window_update_variable: start\n");
01831 #endif
01832   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01833   if (i < 0)
01834     i = 0;
01835   switch (window_get_algorithm ())
01836     {
01837     case ALGORITHM_SWEEP:
01838     case ALGORITHM_ORTHOGONAL:
01839       input->variable[i].nsweeps
01840         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01841 #if DEBUG_INTERFACE
01842       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01843                input->variable[i].nsweeps);
01844 #endif
01845       break;
01846     case ALGORITHM_GENETIC:
01847       input->variable[i].nbits
01848         = gtk_spin_button_get_value_as_int (window->spin_bits);
01849 #if DEBUG_INTERFACE
```

```
01850       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01851                input->variable[i].nbits);
01852 #endif
01853     }
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_update_variable: end\n");
01856 #endif
01857 }
01858
01864 int
01865 window_read (char *filename)
01866 {
01867   unsigned int i;
01868   char *buffer;
01869 #if DEBUG_INTERFACE
01870   fprintf (stderr, "window_read: start\n");
01871 #endif
01872
01873   // Reading new input file
01874   input_free ();
01875   input->result = input->variables = NULL;
01876   if (!input_open (filename))
01877     {
01878 #if DEBUG_INTERFACE
01879       fprintf (stderr, "window_read: end\n");
01880 #endif
01881       return 0;
01882     }
01883
01884   // Setting GTK+ widgets data
01885   gtk_entry_set_text (window->entry_result, input->result);
01886   gtk_entry_set_text (window->entry_variables, input->
      variables);
01887   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01888   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01889                                  (window->button_simulator), buffer);
01890   g_free (buffer);
01891   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01892                                 (size_t) input->evaluator);
01893   if (input->evaluator)
01894     {
01895       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01896       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01897                                      (window->button_evaluator), buffer);
01898       g_free (buffer);
01899     }
01900   gtk_toggle_button_set_active
01901     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01902   switch (input->algorithm)
01903     {
01904     case ALGORITHM_MONTE_CARLO:
01905       gtk_spin_button_set_value (window->spin_simulations,
01906                                  (gdouble) input->nsimulations);
01907       // fallthrough
01908     case ALGORITHM_SWEEP:
01909     case ALGORITHM_ORTHOGONAL:
01910       gtk_spin_button_set_value (window->spin_iterations,
01911                                  (gdouble) input->niterations);
01912       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01913       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01914       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01915                                     (window->check_direction),
      input->nsteps);
01916       if (input->nsteps)
01917         {
01918           gtk_toggle_button_set_active
01919             (GTK_TOGGLE_BUTTON (window->button_direction
01920                                 [input->direction]), TRUE);
01921           gtk_spin_button_set_value (window->spin_steps,
01922                                      (gdouble) input->nsteps);
01923           gtk_spin_button_set_value (window->spin_relaxation,
01924                                      (gdouble) input->relaxation);
01925           switch (input->direction)
01926             {
01927             case DIRECTION_METHOD_RANDOM:
01928               gtk_spin_button_set_value (window->spin_estimates,
01929                                          (gdouble) input->nestimates);
01930             }
01931         }
01932       break;
01933     default:
01934       gtk_spin_button_set_value (window->spin_population,
```

```
01935                                 (gdouble) input->nsimulations);
01936       gtk_spin_button_set_value (window->spin_generations,
01937                                  (gdouble) input->niterations);
01938       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01939       gtk_spin_button_set_value (window->spin_reproduction,
01940                                  input->reproduction_ratio);
01941       gtk_spin_button_set_value (window->spin_adaptation,
01942                                  input->adaptation_ratio);
01943     }
01944   gtk_toggle_button_set_active
01945     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01946   gtk_spin_button_set_value (window->spin_p, input->p);
01947   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01948   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01949   g_signal_handler_block (window->button_experiment,
01950                           window->id_experiment_name);
01951   gtk_combo_box_text_remove_all (window->combo_experiment);
01952   for (i = 0; i < input->nexperiments; ++i)
01953     gtk_combo_box_text_append_text (window->combo_experiment,
01954                                     input->experiment[i].name);
01955   g_signal_handler_unblock
01956     (window->button_experiment, window->id_experiment_name);
01957   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01958   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01959   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01960   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01961   gtk_combo_box_text_remove_all (window->combo_variable);
01962   for (i = 0; i < input->nvariables; ++i)
01963     gtk_combo_box_text_append_text (window->combo_variable,
01964                                     input->variable[i].name);
01965   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01966   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01967   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01968   window_set_variable ();
01969   window_update ();
01970
01971 #if DEBUG_INTERFACE
01972   fprintf (stderr, "window_read: end\n");
01973 #endif
01974   return 1;
01975 }
01976
01980 void
01981 window_open ()
01982 {
01983   GtkFileChooserDialog *dlg;
01984   GtkFileFilter *filter;
01985   char *buffer, *directory, *name;
01986
01987 #if DEBUG_INTERFACE
01988   fprintf (stderr, "window_open: start\n");
01989 #endif
01990
01991   // Saving a backup of the current input file
01992   directory = g_strdup (input->directory);
01993   name = g_strdup (input->name);
01994
01995   // Opening dialog
01996   dlg = (GtkFileChooserDialog *)
01997     gtk_file_chooser_dialog_new (_("Open input file"),
01998                                  window->window,
01999                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02000                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02001                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02002
02003   // Adding XML filter
02004   filter = (GtkFileFilter *) gtk_file_filter_new ();
02005   gtk_file_filter_set_name (filter, "XML");
02006   gtk_file_filter_add_pattern (filter, "*.xml");
02007   gtk_file_filter_add_pattern (filter, "*.XML");
02008   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02009
02010   // Adding JSON filter
02011   filter = (GtkFileFilter *) gtk_file_filter_new ();
02012   gtk_file_filter_set_name (filter, "JSON");
02013   gtk_file_filter_add_pattern (filter, "*.json");
02014   gtk_file_filter_add_pattern (filter, "*.JSON");
02015   gtk_file_filter_add_pattern (filter, "*.js");
02016   gtk_file_filter_add_pattern (filter, "*.JS");
```

```
02017   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019   // If OK saving
02020   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02021     {
02022
02023       // Traying to open the input file
02024       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02025       if (!window_read (buffer))
02026         {
02027 #if DEBUG_INTERFACE
02028           fprintf (stderr, "window_open: error reading input file\n");
02029 #endif
02030           g_free (buffer);
02031
02032           // Reading backup file on error
02033           buffer = g_build_filename (directory, name, NULL);
02034           input->result = input->variables = NULL;
02035           if (!input_open (buffer))
02036             {
02037
02038               // Closing on backup file reading error
02039 #if DEBUG_INTERFACE
02040               fprintf (stderr, "window_read: error reading backup file\n");
02041 #endif
02042               g_free (buffer);
02043               break;
02044             }
02045           g_free (buffer);
02046         }
02047       else
02048         {
02049           g_free (buffer);
02050           break;
02051         }
02052     }
02053
02054   // Freeing and closing
02055   g_free (name);
02056   g_free (directory);
02057   gtk_widget_destroy (GTK_WIDGET (dlg));
02058 #if DEBUG_INTERFACE
02059   fprintf (stderr, "window_open: end\n");
02060 #endif
02061 }
02062
02066 void
02067 window_new (GtkApplication * application)
02068 {
02069   unsigned int i;
02070   char *buffer, *buffer2, buffer3[64];
02071   char *label_algorithm[NALGORITHMS] = {
02072     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02073   };
02074   char *tip_algorithm[NALGORITHMS] = {
02075     _("Monte-Carlo brute force algorithm"),
02076     _("Sweep brute force algorithm"),
02077     _("Genetic algorithm"),
02078     _("Orthogonal sampling brute force algorithm"),
02079   };
02080   char *label_direction[NDIRECTIONS] = {
02081     _("_Coordinates descent"), _("_Random")
02082   };
02083   char *tip_direction[NDIRECTIONS] = {
02084     _("Coordinates direction estimate method"),
02085     _("Random direction estimate method")
02086   };
02087   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02088   char *tip_norm[NNORMS] = {
02089     _("Euclidean error norm (L2)"),
02090     _("Maximum error norm (L)"),
02091     _("P error norm (Lp)"),
02092     _("Taxicab error norm (L1)")
02093   };
02094
02095 #if DEBUG_INTERFACE
02096   fprintf (stderr, "window_new: start\n");
02097 #endif
02098
02099   // Creating the window
02100   window->window = main_window
02101     = (GtkWindow *) gtk_application_window_new (application);
02102
02103   // Finish when closing the window
02104   g_signal_connect_swapped (window->window, "delete-event",
02105                             G_CALLBACK (g_application_quit),
02106                             G_APPLICATION (application));
```

```
02107
02108   // Setting the window title
02109   gtk_window_set_title (window->window, "MPCOTool");
02110
02111   // Creating the open button
02112   window->button_open = (GtkToolButton *) gtk_tool_button_new
02113     (gtk_image_new_from_icon_name ("document-open",
02114                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02115   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02116
02117   // Creating the save button
02118   window->button_save = (GtkToolButton *) gtk_tool_button_new
02119     (gtk_image_new_from_icon_name ("document-save",
02120                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02121   g_signal_connect (window->button_save, "clicked", (GCallback)
      window_save,
02122                     NULL);
02123
02124   // Creating the run button
02125   window->button_run = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("system-run",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02128   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02129
02130   // Creating the options button
02131   window->button_options = (GtkToolButton *) gtk_tool_button_new
02132     (gtk_image_new_from_icon_name ("preferences-system",
02133                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02134   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02135
02136   // Creating the help button
02137   window->button_help = (GtkToolButton *) gtk_tool_button_new
02138     (gtk_image_new_from_icon_name ("help-browser",
02139                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02140   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02141
02142   // Creating the about button
02143   window->button_about = (GtkToolButton *) gtk_tool_button_new
02144     (gtk_image_new_from_icon_name ("help-about",
02145                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02146   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02147
02148   // Creating the exit button
02149   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02150     (gtk_image_new_from_icon_name ("application-exit",
02151                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02152   g_signal_connect_swapped (window->button_exit, "clicked",
02153                             G_CALLBACK (g_application_quit),
02154                             G_APPLICATION (application));
02155
02156   // Creating the buttons bar
02157   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02158   gtk_toolbar_insert
02159     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02160   gtk_toolbar_insert
02161     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02162   gtk_toolbar_insert
02163     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02164   gtk_toolbar_insert
02165     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02166   gtk_toolbar_insert
02167     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02168   gtk_toolbar_insert
02169     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02170   gtk_toolbar_insert
02171     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02172   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02173
02174   // Creating the simulator program label and entry
02175   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02176   window->button_simulator = (GtkFileChooserButton *)
02177     gtk_file_chooser_button_new (_("Simulator program"),
02178                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02179   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02180                                _("Simulator program executable file"));
02181   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02182
02183   // Creating the evaluator program label and entry
02184   window->check_evaluator = (GtkCheckButton *)
02185     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02186   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02187   window->button_evaluator = (GtkFileChooserButton *)
02188     gtk_file_chooser_button_new (_("Evaluator program"),
02189                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02190   gtk_widget_set_tooltip_text
02191     (GTK_WIDGET (window->button_evaluator),
```

```
02192     _("Optional evaluator program executable file"));
02193
02194   // Creating the results files labels and entries
02195   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02196   window->entry_result = (GtkEntry *) gtk_entry_new ();
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->entry_result), _("Best results file"));
02199   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02200   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02201   gtk_widget_set_tooltip_text
02202     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02203
02204   // Creating the files grid and attaching widgets
02205   window->grid_files = (GtkGrid *) gtk_grid_new ();
02206   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02207                    0, 0, 1, 1);
02208   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02209                    1, 0, 1, 1);
02210   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02211                    0, 1, 1, 1);
02212   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02213                    1, 1, 1, 1);
02214   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02215                    0, 2, 1, 1);
02216   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02217                    1, 2, 1, 1);
02218   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02219                    0, 3, 1, 1);
02220   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02221                    1, 3, 1, 1);
02222
02223   // Creating the algorithm properties
02224   window->label_simulations = (GtkLabel *) gtk_label_new
02225     (_("Simulations number"));
02226   window->spin_simulations
02227     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02228   gtk_widget_set_tooltip_text
02229     (GTK_WIDGET (window->spin_simulations),
02230      _("Number of simulations to perform for each iteration"));
02231   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02232   window->label_iterations = (GtkLabel *)
02233     gtk_label_new (_("Iterations number"));
02234   window->spin_iterations
02235     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02236   gtk_widget_set_tooltip_text
02237     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02238   g_signal_connect
02239     (window->spin_iterations, "value-changed", window_update, NULL);
02240   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02241   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02242   window->spin_tolerance =
02243     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02244   gtk_widget_set_tooltip_text
02245     (GTK_WIDGET (window->spin_tolerance),
02246      _("Tolerance to set the variable interval on the next iteration"));
02247   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02248   window->spin_bests
02249     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02250   gtk_widget_set_tooltip_text
02251     (GTK_WIDGET (window->spin_bests),
02252      _("Number of best simulations used to set the variable interval "
02253        "on the next iteration"));
02254   window->label_population
02255     = (GtkLabel *) gtk_label_new (_("Population number"));
02256   window->spin_population
02257     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02258   gtk_widget_set_tooltip_text
02259     (GTK_WIDGET (window->spin_population),
02260      _("Number of population for the genetic algorithm"));
02261   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02262   window->label_generations
02263     = (GtkLabel *) gtk_label_new (_("Generations number"));
02264   window->spin_generations
02265     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02266   gtk_widget_set_tooltip_text
02267     (GTK_WIDGET (window->spin_generations),
02268      _("Number of generations for the genetic algorithm"));
02269   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02270   window->spin_mutation
```

```
02271     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02272   gtk_widget_set_tooltip_text
02273     (GTK_WIDGET (window->spin_mutation),
02274      _("Ratio of mutation for the genetic algorithm"));
02275   window->label_reproduction
02276     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02277   window->spin_reproduction
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279   gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_reproduction),
02281      _("Ratio of reproduction for the genetic algorithm"));
02282   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02283   window->spin_adaptation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285   gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_adaptation),
02287      _("Ratio of adaptation for the genetic algorithm"));
02288   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02289   window->spin_threshold = (GtkSpinButton *)
02290     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02291                                     precision[DEFAULT_PRECISION]);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_threshold),
02294      _("Threshold in the objective function to finish the simulations"));
02295   window->scrolled_threshold =
02296     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02297   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02298                      GTK_WIDGET (window->spin_threshold));
02299 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02300 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02301 //                         GTK_ALIGN_FILL);
02302
02303   // Creating the direction search method properties
02304   window->check_direction = (GtkCheckButton *)
02305     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02306   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02307   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02308   window->button_direction[0] = (GtkRadioButton *)
02309     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02310   gtk_grid_attach (window->grid_direction,
02311                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02312   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02313                     NULL);
02314   for (i = 0; ++i < NDIRECTIONS;)
02315     {
02316       window->button_direction[i] = (GtkRadioButton *)
02317         gtk_radio_button_new_with_mnemonic
02318         (gtk_radio_button_get_group (window->button_direction[0]),
02319          label_direction[i]);
02320       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02321                                    tip_direction[i]);
02322       gtk_grid_attach (window->grid_direction,
02323                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02324       g_signal_connect (window->button_direction[i], "clicked",
02325                         window_update, NULL);
02326     }
02327   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02328   window->spin_steps = (GtkSpinButton *)
02329     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02330   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02331   window->label_estimates
02332     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02333   window->spin_estimates = (GtkSpinButton *)
02334     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02335   window->label_relaxation
02336     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02337   window->spin_relaxation = (GtkSpinButton *)
02338     gtk_spin_button_new_with_range (0., 2., 0.001);
02339   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02340                    0, NDIRECTIONS, 1, 1);
02341   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02342                    1, NDIRECTIONS, 1, 1);
02343   gtk_grid_attach (window->grid_direction,
02344                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02345                    1, 1);
02346   gtk_grid_attach (window->grid_direction,
02347                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02348                    1);
02349   gtk_grid_attach (window->grid_direction,
02350                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02351                    1, 1);
02352   gtk_grid_attach (window->grid_direction,
02353                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
```

```
02354                   1, 1);
02355
02356   // Creating the array of algorithms
02357   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02358   window->button_algorithm[0] = (GtkRadioButton *)
02359     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02360   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02361                                tip_algorithm[0]);
02362   gtk_grid_attach (window->grid_algorithm,
02363                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02364   g_signal_connect (window->button_algorithm[0], "clicked",
02365                     window_set_algorithm, NULL);
02366   for (i = 0; ++i < NALGORITHMS;)
02367     {
02368       window->button_algorithm[i] = (GtkRadioButton *)
02369         gtk_radio_button_new_with_mnemonic
02370         (gtk_radio_button_get_group (window->button_algorithm[0]),
02371          label_algorithm[i]);
02372       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02373                                    tip_algorithm[i]);
02374       gtk_grid_attach (window->grid_algorithm,
02375                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02376       g_signal_connect (window->button_algorithm[i], "clicked",
02377                         window_set_algorithm, NULL);
02378     }
02379   gtk_grid_attach (window->grid_algorithm,
02380                    GTK_WIDGET (window->label_simulations),
02381                    0, NALGORITHMS, 1, 1);
02382   gtk_grid_attach (window->grid_algorithm,
02383                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02384   gtk_grid_attach (window->grid_algorithm,
02385                    GTK_WIDGET (window->label_iterations),
02386                    0, NALGORITHMS + 1, 1, 1);
02387   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_iterations),
02388                    1, NALGORITHMS + 1, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_tolerance),
02390                    0, NALGORITHMS + 2, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_tolerance),
02392                    1, NALGORITHMS + 2, 1, 1);
02393   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_bests), 0,
02394                    NALGORITHMS + 3, 1, 1);
02395   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_bests), 1,
02396                    NALGORITHMS + 3, 1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_population),
02399                    0, NALGORITHMS + 4, 1, 1);
02400   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_population),
02401                    1, NALGORITHMS + 4, 1, 1);
02402   gtk_grid_attach (window->grid_algorithm,
02403                    GTK_WIDGET (window->label_generations),
02404                    0, NALGORITHMS + 5, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->spin_generations),
02407                    1, NALGORITHMS + 5, 1, 1);
02408   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_mutation),
02409                    0, NALGORITHMS + 6, 1, 1);
02410   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_mutation),
02411                    1, NALGORITHMS + 6, 1, 1);
02412   gtk_grid_attach (window->grid_algorithm,
02413                    GTK_WIDGET (window->label_reproduction),
02414                    0, NALGORITHMS + 7, 1, 1);
02415   gtk_grid_attach (window->grid_algorithm,
02416                    GTK_WIDGET (window->spin_reproduction),
02417                    1, NALGORITHMS + 7, 1, 1);
02418   gtk_grid_attach (window->grid_algorithm,
02419                    GTK_WIDGET (window->label_adaptation),
02420                    0, NALGORITHMS + 8, 1, 1);
02421   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_adaptation),
02422                    1, NALGORITHMS + 8, 1, 1);
02423   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      check_direction),
02424                    0, NALGORITHMS + 9, 2, 1);
02425   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      grid_direction),
02426                    0, NALGORITHMS + 10, 2, 1);
02427   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_threshold),
02428                    0, NALGORITHMS + 11, 1, 1);
```

```
02429   gtk_grid_attach (window->grid_algorithm,
02430                    GTK_WIDGET (window->scrolled_threshold),
02431                    1, NALGORITHMS + 11, 1, 1);
02432   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02433   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02434                      GTK_WIDGET (window->grid_algorithm));
02435
02436   // Creating the variable widgets
02437   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02438   gtk_widget_set_tooltip_text
02439     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02440   window->id_variable = g_signal_connect
02441     (window->combo_variable, "changed", window_set_variable, NULL);
02442   window->button_add_variable
02443     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02444                                                    GTK_ICON_SIZE_BUTTON);
02445   g_signal_connect
02446     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02447   gtk_widget_set_tooltip_text
02448     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02449   window->button_remove_variable
02450     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02451                                                    GTK_ICON_SIZE_BUTTON);
02452   g_signal_connect
02453     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02456   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02457   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02458   gtk_widget_set_tooltip_text
02459     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02460   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02461   window->id_variable_label = g_signal_connect
02462     (window->entry_variable, "changed", window_label_variable, NULL);
02463   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02464   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02465     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02466   gtk_widget_set_tooltip_text
02467     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02468   window->scrolled_min
02469     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02470   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02471                      GTK_WIDGET (window->spin_min));
02472   g_signal_connect (window->spin_min, "value-changed",
02473                     window_rangemin_variable, NULL);
02474   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02475   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02476     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02479   window->scrolled_max
02480     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02481   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02482                      GTK_WIDGET (window->spin_max));
02483   g_signal_connect (window->spin_max, "value-changed",
02484                     window_rangemax_variable, NULL);
02485   window->check_minabs = (GtkCheckButton *)
02486     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02487   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02488   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02489     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02490   gtk_widget_set_tooltip_text
02491     (GTK_WIDGET (window->spin_minabs),
02492      _("Minimum allowed value of the variable"));
02493   window->scrolled_minabs
02494     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02495   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02496                      GTK_WIDGET (window->spin_minabs));
02497   g_signal_connect (window->spin_minabs, "value-changed",
02498                     window_rangeminabs_variable, NULL);
02499   window->check_maxabs = (GtkCheckButton *)
02500     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02501   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02502   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02503     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02504   gtk_widget_set_tooltip_text
02505     (GTK_WIDGET (window->spin_maxabs),
02506      _("Maximum allowed value of the variable"));
02507   window->scrolled_maxabs
02508     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02509   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02510                      GTK_WIDGET (window->spin_maxabs));
02511   g_signal_connect (window->spin_maxabs, "value-changed",
02512                     window_rangemaxabs_variable, NULL);
02513   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
```

```
02514   window->spin_precision = (GtkSpinButton *)
02515     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02516   gtk_widget_set_tooltip_text
02517     (GTK_WIDGET (window->spin_precision),
02518      _("Number of precision floating point digits\n"
02519        "0 is for integer numbers"));
02520   g_signal_connect (window->spin_precision, "value-changed",
02521                     window_precision_variable, NULL);
02522   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02523   window->spin_sweeps =
02524     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02525   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02526                                _("Number of steps sweeping the variable"));
02527   g_signal_connect (window->spin_sweeps, "value-changed",
02528                     window_update_variable, NULL);
02529   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02530   window->spin_bits
02531     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_bits),
02534      _("Number of bits to encode the variable"));
02535   g_signal_connect
02536     (window->spin_bits, "value-changed", window_update_variable, NULL);
02537   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02538   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02539     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02540   gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_step),
02542      _("Initial step size for the direction search method"));
02543   window->scrolled_step
02544     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02545   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02546                      GTK_WIDGET (window->spin_step));
02547   g_signal_connect
02548     (window->spin_step, "value-changed", window_step_variable, NULL);
02549   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02550   gtk_grid_attach (window->grid_variable,
02551                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02552   gtk_grid_attach (window->grid_variable,
02553                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02554   gtk_grid_attach (window->grid_variable,
02555                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02556   gtk_grid_attach (window->grid_variable,
02557                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02558   gtk_grid_attach (window->grid_variable,
02559                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02560   gtk_grid_attach (window->grid_variable,
02561                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02562   gtk_grid_attach (window->grid_variable,
02563                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02564   gtk_grid_attach (window->grid_variable,
02565                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02592   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02593   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02594                      GTK_WIDGET (window->grid_variable));
02595
02596   // Creating the experiment widgets
02597   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02598   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02599                                _("Experiment selector"));
02600   window->id_experiment = g_signal_connect
```

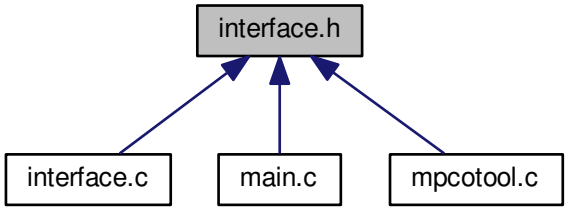
```
02601     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
02602   window->button_add_experiment
02603     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02604                                                    GTK_ICON_SIZE_BUTTON);
02605   g_signal_connect
02606     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02607   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02608                                _("Add experiment"));
02609   window->button_remove_experiment
02610     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02611                                                    GTK_ICON_SIZE_BUTTON);
02612   g_signal_connect (window->button_remove_experiment, "clicked",
02613                     window_remove_experiment, NULL);
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02615                                _("Remove experiment"));
02616   window->label_experiment
02617     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02618   window->button_experiment = (GtkFileChooserButton *)
02619     gtk_file_chooser_button_new (_("Experimental data file"),
02620                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02621   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02622                                _("Experimental data file"));
02623   window->id_experiment_name
02624     = g_signal_connect (window->button_experiment, "selection-changed",
02625                         window_name_experiment, NULL);
02626   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02627   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02628   window->spin_weight
02629     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02630   gtk_widget_set_tooltip_text
02631     (GTK_WIDGET (window->spin_weight),
02632      _("Weight factor to build the objective function"));
02633   g_signal_connect
02634     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
02635   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02636   gtk_grid_attach (window->grid_experiment,
02637                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02638   gtk_grid_attach (window->grid_experiment,
02639                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02640   gtk_grid_attach (window->grid_experiment,
02641                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02642   gtk_grid_attach (window->grid_experiment,
02643                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02644   gtk_grid_attach (window->grid_experiment,
02645                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02646   gtk_grid_attach (window->grid_experiment,
02647                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02648   gtk_grid_attach (window->grid_experiment,
02649                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02650   for (i = 0; i < MAX_NINPUTS; ++i)
02651     {
02652       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02653       window->check_template[i] = (GtkCheckButton *)
02654         gtk_check_button_new_with_label (buffer3);
02655       window->id_template[i]
02656         = g_signal_connect (window->check_template[i], "toggled",
02657                             window_inputs_experiment, NULL);
02658       gtk_grid_attach (window->grid_experiment,
02659                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02660       window->button_template[i] =
02661         (GtkFileChooserButton *)
02662         gtk_file_chooser_button_new (_("Input template"),
02663                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02664       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02665                                    _("Experimental input template file"));
02666       window->id_input[i] =
02667         g_signal_connect_swapped (window->button_template[i],
02668                                   "selection-changed",
02669                                   (GCallback) window_template_experiment,
02670                                   (void *) (size_t) i);
02671       gtk_grid_attach (window->grid_experiment,
02672                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02673     }
02674   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02675   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02676                      GTK_WIDGET (window->grid_experiment));
02677
02678   // Creating the error norm widgets
02679   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02680   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02681   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02682                      GTK_WIDGET (window->grid_norm));
02683   window->button_norm[0] = (GtkRadioButton *)
02684     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
```

```
02685   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02686                                tip_norm[0]);
02687   gtk_grid_attach (window->grid_norm,
02688                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02689   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02690   for (i = 0; ++i < NNORMS;)
02691     {
02692       window->button_norm[i] = (GtkRadioButton *)
02693         gtk_radio_button_new_with_mnemonic
02694         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02695       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02696                                    tip_norm[i]);
02697       gtk_grid_attach (window->grid_norm,
02698                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02699       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02700     }
02701   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02702   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02703   window->spin_p =
02704     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02705                                                       G_MAXDOUBLE, 0.01);
02706   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02707                                _("P parameter for the P error norm"));
02708   window->scrolled_p =
02709     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02710   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02711                      GTK_WIDGET (window->spin_p));
02712   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02713   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02714   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02715                    1, 2, 1, 2);
02716
02717   // Creating the grid and attaching the widgets to the grid
02718   window->grid = (GtkGrid *) gtk_grid_new ();
02719   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02720   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02721   gtk_grid_attach (window->grid,
02722                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02723   gtk_grid_attach (window->grid,
02724                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02725   gtk_grid_attach (window->grid,
02726                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02727   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02728   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
02729
02730   // Setting the window logo
02731   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02732   gtk_window_set_icon (window->window, window->logo);
02733
02734   // Showing the window
02735   gtk_widget_show_all (GTK_WIDGET (window->window));
02736
02737   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02738 #if GTK_MINOR_VERSION >= 16
02739   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02740   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02741   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02742   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02743   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02744   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02745   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02746 #endif
02747
02748   // Reading initial example
02749   input_new ();
02750   buffer2 = g_get_current_dir ();
02751   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02752   g_free (buffer2);
02753   window_read (buffer);
02754   g_free (buffer);
02755
02756 #if DEBUG_INTERFACE
02757   fprintf (stderr, "window_new: start\n");
02758 #endif
02759 }
```

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_direction ()
- unsigned int window_get_norm ()
- void window_save_direction ()
- int window_save ()
- void window_run ()
- void window_help ()
- void window_update_direction ()
- 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.

```
00471 {
00472   unsigned int i;
00473   for (i = 0; i < n; ++i)
00474     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475       break;
00476   return i;
00477 }
```

4.13.2.2 input_save()

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

Function to save the input file.

Parameters

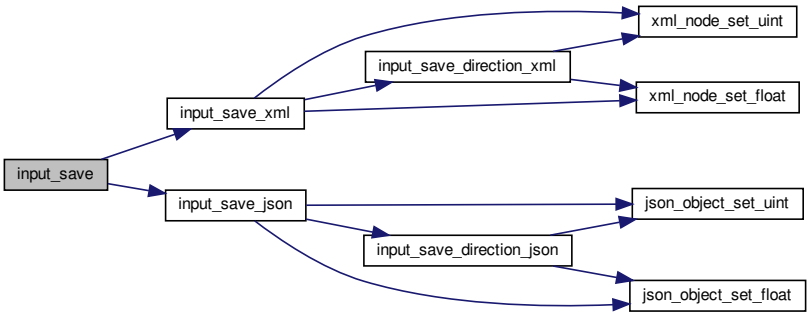
filename	Input file name.

Definition at line 584 of file interface.c.

```
00585 {
00586   xmlDoc *doc;
00587   JsonGenerator *generator;
00588
00589 #if DEBUG_INTERFACE
00590   fprintf (stderr, "input_save: start\n");
00591 #endif
00592
00593   // Getting the input file directory
00594   input->name = g_path_get_basename (filename);
00595   input->directory = g_path_get_dirname (filename);
00596
00597   if (input->type == INPUT_TYPE_XML)
00598     {
00599       // Opening the input file
00600       doc = xmlNewDoc ((const xmlChar *) "1.0");
00601       input_save_xml (doc);
00602
00603       // Saving the XML file
00604       xmlSaveFormatFile (filename, doc, 1);
00605
00606       // Freeing memory
00607       xmlFreeDoc (doc);
00608     }
00609   else
00610     {
00611       // Opening the input file
00612       generator = json_generator_new ();
00613       json_generator_set_pretty (generator, TRUE);
00614       input_save_json (generator);
00615
00616       // Saving the JSON file
00617       json_generator_to_file (generator, filename, NULL);
00618
00619       // Freeing memory
```

```
00620       g_object_unref (generator);
00621     }
00622
00623 #if DEBUG_INTERFACE
00624   fprintf (stderr, "input_save: end\n");
00625 #endif
00626 }
```

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.

```
00633 {
00634 #if DEBUG_INTERFACE
00635   fprintf (stderr, "options_new: start\n");
00636 #endif
00637   options->label_seed = (GtkLabel *)
00638     gtk_label_new (_("Pseudo-random numbers generator seed"));
00639   options->spin_seed = (GtkSpinButton *)
00640     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00641   gtk_widget_set_tooltip_text
00642     (GTK_WIDGET (options->spin_seed),
00643      _("Seed to init the pseudo-random numbers generator"));
00644   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00645   options->label_threads = (GtkLabel *)
00646     gtk_label_new (_("Threads number for the stochastic algorithm"));
00647   options->spin_threads
00648     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00649   gtk_widget_set_tooltip_text
00650     (GTK_WIDGET (options->spin_threads),
00651      _("Number of threads to perform the calibration/optimization for "
00652        "the stochastic algorithm"));
00653   gtk_spin_button_set_value (options->spin_threads, (gdouble) nthreads);
00654   options->label_direction = (GtkLabel *)
00655     gtk_label_new (_("Threads number for the direction search method"));
00656   options->spin_direction =
00657     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00658   gtk_widget_set_tooltip_text
00659     (GTK_WIDGET (options->spin_direction),
00660      _("Number of threads to perform the calibration/optimization for the "
00661        "direction search method"));
00662   gtk_spin_button_set_value (options->spin_direction,
```

```
00663                               (gdouble) nthreads_direction);
00664   options->grid = (GtkGrid *) gtk_grid_new ();
00665   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_seed), 0, 0, 1, 1);
00666   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_seed), 1, 0, 1, 1);
00667   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_threads), 0, 1,
00668                    1, 1);
00669   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_threads), 1, 1, 1,
00670                    1);
00671   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      label_direction), 0, 2,
00672                    1, 1);
00673   gtk_grid_attach (options->grid, GTK_WIDGET (options->
      spin_direction), 1, 2,
00674                    1, 1);
00675   gtk_widget_show_all (GTK_WIDGET (options->grid));
00676   options->dialog = (GtkDialog *)
00677     gtk_dialog_new_with_buttons (_("Options"),
00678                                  window->window,
00679                                  GTK_DIALOG_MODAL,
00680                                  _("_OK"), GTK_RESPONSE_OK,
00681                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00682   gtk_container_add
00683     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00684      GTK_WIDGET (options->grid));
00685   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00686     {
00687       input->seed
00688         = (unsigned long int) gtk_spin_button_get_value (options->
      spin_seed);
00689       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00690       nthreads_direction
00691         = gtk_spin_button_get_value_as_int (options->spin_direction);
00692     }
00693   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00694 #if DEBUG_INTERFACE
00695   fprintf (stderr, "options_new: end\n");
00696 #endif
00697 }
```

4.13.2.4 running_new()

```
void running_new ( )
```

Function to open the running dialog.

Definition at line 703 of file interface.c.

```
00704 {
00705 #if DEBUG_INTERFACE
00706   fprintf (stderr, "running_new: start\n");
00707 #endif
00708   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00709   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00710   running->grid = (GtkGrid *) gtk_grid_new ();
00711   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00712   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00713   running->dialog = (GtkDialog *)
00714     gtk_dialog_new_with_buttons (_("Calculating"),
00715                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00716   gtk_container_add (GTK_CONTAINER
00717                      (gtk_dialog_get_content_area (running->dialog)),
00718                      GTK_WIDGET (running->grid));
00719   gtk_spinner_start (running->spinner);
00720   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721 #if DEBUG_INTERFACE
00722   fprintf (stderr, "running_new: end\n");
00723 #endif
00724 }
```

4.13.2.5 window_add_experiment()

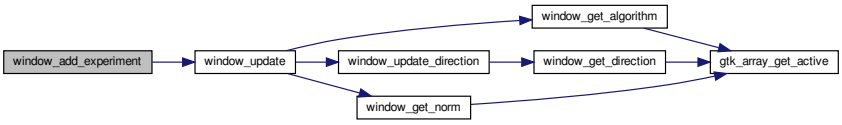
```
void window_add_experiment ( )
```

Function to add an experiment in the main window.

Definition at line 1394 of file interface.c.

```
01395 {
01396   unsigned int i, j;
01397 #if DEBUG_INTERFACE
01398   fprintf (stderr, "window_add_experiment: start\n");
01399 #endif
01400   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01401   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01402   gtk_combo_box_text_insert_text
01403     (window->combo_experiment, i, input->experiment[i].
      name);
01404   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01405   input->experiment = (Experiment *) g_realloc
01406     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01407   for (j = input->nexperiments - 1; j > i; --j)
01408     memcpy (input->experiment + j + 1, input->experiment + j,
01409             sizeof (Experiment));
01410   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01411   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01412   if (input->type == INPUT_TYPE_XML)
01413     {
01414       input->experiment[j + 1].name
01415         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01416       for (j = 0; j < input->experiment->ninputs; ++j)
01417         input->experiment[i + 1].stencil[j]
01418           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      stencil[j]);
01419     }
01420   else
01421     {
01422       input->experiment[j + 1].name = g_strdup (input->
      experiment[j].name);
01423       for (j = 0; j < input->experiment->ninputs; ++j)
01424         input->experiment[i + 1].stencil[j]
01425           = g_strdup (input->experiment[i].stencil[j]);
01426     }
01427   ++input->nexperiments;
01428   for (j = 0; j < input->experiment->ninputs; ++j)
01429     g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01430   g_signal_handler_block
01431     (window->button_experiment, window->
      id_experiment_name);
01432   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01433   g_signal_handler_unblock
01434     (window->button_experiment, window->
      id_experiment_name);
01435   for (j = 0; j < input->experiment->ninputs; ++j)
01436     g_signal_handler_unblock (window->button_template[j],
      window->id_input[j]);
01437   window_update ();
01438 #if DEBUG_INTERFACE
01439   fprintf (stderr, "window_add_experiment: end\n");
01440 #endif
01441 }
```

Here is the call graph for this function:



4.13.2.6 window_add_variable()

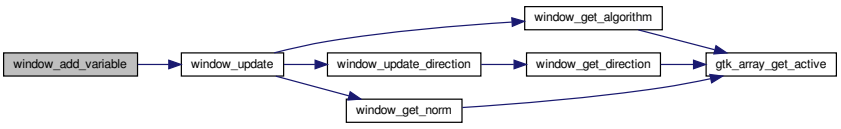
```
void window_add_variable ( )
```

Function to add a variable in the main window.

Definition at line 1657 of file interface.c.

```
01658 {
01659   unsigned int i, j;
01660 #if DEBUG_INTERFACE
01661   fprintf (stderr, "window_add_variable: start\n");
01662 #endif
01663   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01664   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01665   gtk_combo_box_text_insert_text (window->combo_variable, i,
01666                                   input->variable[i].name);
01667   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01668   input->variable = (Variable *) g_realloc
01669     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01670   for (j = input->nvariables - 1; j > i; --j)
01671     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01672   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01673   if (input->type == INPUT_TYPE_XML)
01674     input->variable[j + 1].name
01675       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01676   else
01677     input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
01678   ++input->nvariables;
01679   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01680   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01681   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01682   window_update ();
01683 #if DEBUG_INTERFACE
01684   fprintf (stderr, "window_add_variable: end\n");
01685 #endif
01686 }
```

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.

```
00733 {
00734   unsigned int i;
00735 #if DEBUG_INTERFACE
00736   fprintf (stderr, "window_get_algorithm: start\n");
00737 #endif
00738   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00739 #if DEBUG_INTERFACE
00740   fprintf (stderr, "window_get_algorithm: %u\n", i);
00741   fprintf (stderr, "window_get_algorithm: end\n");
00742 #endif
00743   return i;
00744 }
```

Here is the call graph for this function:

4.13.2.8 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 752 of file interface.c.

```
00753 {
00754   unsigned int i;
00755 #if DEBUG_INTERFACE
00756   fprintf (stderr, "window_get_direction: start\n");
00757 #endif
00758   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00759 #if DEBUG_INTERFACE
00760   fprintf (stderr, "window_get_direction: %u\n", i);
00761   fprintf (stderr, "window_get_direction: end\n");
00762 #endif
00763   return i;
00764 }
```

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.

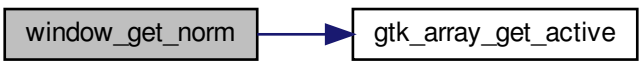
Returns

Norm method number.

Definition at line 772 of file interface.c.

```
00773 {
00774   unsigned int i;
00775 #if DEBUG_INTERFACE
00776   fprintf (stderr, "window_get_norm: start\n");
00777 #endif
00778   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00779 #if DEBUG_INTERFACE
00780   fprintf (stderr, "window_get_norm: %u\n", i);
00781   fprintf (stderr, "window_get_norm: end\n");
00782 #endif
00783   return i;
00784 }
```

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 1030 of file interface.c.

```
01031 {
01032   char *buffer, *buffer2;
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_help: start\n");
01035 #endif
01036   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01037                               _("user-manual.pdf"), NULL);
01038   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01039   g_free (buffer2);
01040 #if GTK_MINOR_VERSION >= 22
01041   gtk_show_uri_on_window (window->window, buffer, GDK_CURRENT_TIME, NULL);
01042 #else
01043   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01044 #endif
01045 #if DEBUG_INTERFACE
01046   fprintf (stderr, "window_help: uri=%s\n", buffer);
01047 #endif
01048   g_free (buffer);
01049 #if DEBUG_INTERFACE
01050   fprintf (stderr, "window_help: end\n");
01051 #endif
01052 }
```

4.13.2.11 window_inputs_experiment()

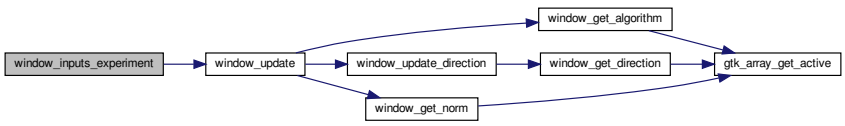
```
void window_inputs_experiment ( )
```

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

Definition at line 1494 of file interface.c.

```
01495 {
01496   unsigned int j;
01497 #if DEBUG_INTERFACE
01498   fprintf (stderr, "window_inputs_experiment: start\n");
01499 #endif
01500   j = input->experiment->ninputs - 1;
01501   if (j
01502       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01503                                         (window->check_template[j])))
01504     --input->experiment->ninputs;
01505   if (input->experiment->ninputs < MAX_NINPUTS
01506       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01507                                        (window->check_template[j])))
01508     ++input->experiment->ninputs;
01509   window_update ();
01510 #if DEBUG_INTERFACE
01511   fprintf (stderr, "window_inputs_experiment: end\n");
01512 #endif
01513 }
```

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 1692 of file interface.c.

```
01693 {
01694   unsigned int i;
01695   const char *buffer;
01696 #if DEBUG_INTERFACE
01697   fprintf (stderr, "window_label_variable: start\n");
01698 #endif
01699   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01700   buffer = gtk_entry_get_text (window->entry_variable);
01701   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01702   gtk_combo_box_text_remove (window->combo_variable, i);
01703   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01704   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01705   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01706 #if DEBUG_INTERFACE
01707   fprintf (stderr, "window_label_variable: end\n");
01708 #endif
01709 }
```

4.13.2.13 window_name_experiment()

```
void window_name_experiment ( )
```

Function to set the experiment name in the main window.

Definition at line 1447 of file interface.c.

```
01448 {
01449   unsigned int i;
01450   char *buffer;
01451   GFile *file1, *file2;
01452 #if DEBUG_INTERFACE
01453   fprintf (stderr, "window_name_experiment: start\n");
01454 #endif
01455   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01456   file1
01457     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->
      button_experiment));
01458   file2 = g_file_new_for_path (input->directory);
01459   buffer = g_file_get_relative_path (file2, file1);
01460   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01461   gtk_combo_box_text_remove (window->combo_experiment, i);
01462   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01463   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01464   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01465   g_free (buffer);
01466   g_object_unref (file2);
01467   g_object_unref (file1);
01468 #if DEBUG_INTERFACE
01469   fprintf (stderr, "window_name_experiment: end\n");
01470 #endif
01471 }
```

4.13.2.14 window_new()

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

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2067 of file interface.c.

```
02068 {
02069   unsigned int i;
02070   char *buffer, *buffer2, buffer3[64];
02071   char *label_algorithm[NALGORITHMS] = {
02072     "_Monte-Carlo", _("_Sweep"), _("_Genetic"), _("_Orthogonal")
02073   };
02074   char *tip_algorithm[NALGORITHMS] = {
02075     _("Monte-Carlo brute force algorithm"),
02076     _("Sweep brute force algorithm"),
02077     _("Genetic algorithm"),
02078     _("Orthogonal sampling brute force algorithm"),
02079   };
02080   char *label_direction[NDIRECTIONS] = {
02081     _("_Coordinates descent"), _("_Random")
02082   };
02083   char *tip_direction[NDIRECTIONS] = {
02084     _("Coordinates direction estimate method"),
02085     _("Random direction estimate method")
02086   };
02087   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02088   char *tip_norm[NNORMS] = {
02089     _("Euclidean error norm (L2)"),
02090     _("Maximum error norm (L)"),
02091     _("P error norm (Lp)"),
02092     _("Taxicab error norm (L1)")
02093   };
02094
02095 #if DEBUG_INTERFACE
02096   fprintf (stderr, "window_new: start\n");
02097 #endif
02098
02099   // Creating the window
02100   window->window = main_window
02101     = (GtkWindow *) gtk_application_window_new (application);
02102
02103   // Finish when closing the window
02104   g_signal_connect_swapped (window->window, "delete-event",
02105                             G_CALLBACK (g_application_quit),
02106                             G_APPLICATION (application));
02107
02108   // Setting the window title
02109   gtk_window_set_title (window->window, "MPCOTool");
02110
02111   // Creating the open button
02112   window->button_open = (GtkToolButton *) gtk_tool_button_new
02113     (gtk_image_new_from_icon_name ("document-open",
02114                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02115   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02116
02117   // Creating the save button
02118   window->button_save = (GtkToolButton *) gtk_tool_button_new
02119     (gtk_image_new_from_icon_name ("document-save",
02120                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02121   g_signal_connect (window->button_save, "clicked", (GCallback)
      window_save,
02122                     NULL);
02123
02124   // Creating the run button
02125   window->button_run = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("system-run",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02128   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02129
02130   // Creating the options button
02131   window->button_options = (GtkToolButton *) gtk_tool_button_new
02132     (gtk_image_new_from_icon_name ("preferences-system",
02133                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02134   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02135
02136   // Creating the help button
02137   window->button_help = (GtkToolButton *) gtk_tool_button_new
02138     (gtk_image_new_from_icon_name ("help-browser",
02139                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02140   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02141
```

```
02142   // Creating the about button
02143   window->button_about = (GtkToolButton *) gtk_tool_button_new
02144     (gtk_image_new_from_icon_name ("help-about",
02145                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02146   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02147
02148   // Creating the exit button
02149   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02150     (gtk_image_new_from_icon_name ("application-exit",
02151                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02152   g_signal_connect_swapped (window->button_exit, "clicked",
02153                             G_CALLBACK (g_application_quit),
02154                             G_APPLICATION (application));
02155
02156   // Creating the buttons bar
02157   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02158   gtk_toolbar_insert
02159     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02160   gtk_toolbar_insert
02161     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02162   gtk_toolbar_insert
02163     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
02164   gtk_toolbar_insert
02165     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02166   gtk_toolbar_insert
02167     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02168   gtk_toolbar_insert
02169     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02170   gtk_toolbar_insert
02171     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02172   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02173
02174   // Creating the simulator program label and entry
02175   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02176   window->button_simulator = (GtkFileChooserButton *)
02177     gtk_file_chooser_button_new (_("Simulator program"),
02178                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02179   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02180                                _("Simulator program executable file"));
02181   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02182
02183   // Creating the evaluator program label and entry
02184   window->check_evaluator = (GtkCheckButton *)
02185     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02186   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02187   window->button_evaluator = (GtkFileChooserButton *)
02188     gtk_file_chooser_button_new (_("Evaluator program"),
02189                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02190   gtk_widget_set_tooltip_text
02191     (GTK_WIDGET (window->button_evaluator),
02192      _("Optional evaluator program executable file"));
02193
02194   // Creating the results files labels and entries
02195   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02196   window->entry_result = (GtkEntry *) gtk_entry_new ();
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->entry_result), _("Best results file"));
02199   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02200   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02201   gtk_widget_set_tooltip_text
02202     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02203
02204   // Creating the files grid and attaching widgets
02205   window->grid_files = (GtkGrid *) gtk_grid_new ();
02206   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02207                    0, 0, 1, 1);
02208   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02209                    1, 0, 1, 1);
02210   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02211                    0, 1, 1, 1);
02212   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02213                    1, 1, 1, 1);
02214   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
```

```
02215                   0, 2, 1, 1);
02216   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02217                   1, 2, 1, 1);
02218   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02219                   0, 3, 1, 1);
02220   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02221                   1, 3, 1, 1);
02222
02223   // Creating the algorithm properties
02224   window->label_simulations = (GtkLabel *) gtk_label_new
02225     (_("Simulations number"));
02226   window->spin_simulations
02227     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02228   gtk_widget_set_tooltip_text
02229     (GTK_WIDGET (window->spin_simulations),
02230      _("Number of simulations to perform for each iteration"));
02231   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02232   window->label_iterations = (GtkLabel *)
02233     gtk_label_new (_("Iterations number"));
02234   window->spin_iterations
02235     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02236   gtk_widget_set_tooltip_text
02237     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02238   g_signal_connect
02239     (window->spin_iterations, "value-changed",
      window_update, NULL);
02240   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02241   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02242   window->spin_tolerance =
02243     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02244   gtk_widget_set_tooltip_text
02245     (GTK_WIDGET (window->spin_tolerance),
02246      _("Tolerance to set the variable interval on the next iteration"));
02247   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02248   window->spin_bests
02249     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02250   gtk_widget_set_tooltip_text
02251     (GTK_WIDGET (window->spin_bests),
02252      _("Number of best simulations used to set the variable interval "
02253        "on the next iteration"));
02254   window->label_population
02255     = (GtkLabel *) gtk_label_new (_("Population number"));
02256   window->spin_population
02257     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02258   gtk_widget_set_tooltip_text
02259     (GTK_WIDGET (window->spin_population),
02260      _("Number of population for the genetic algorithm"));
02261   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02262   window->label_generations
02263     = (GtkLabel *) gtk_label_new (_("Generations number"));
02264   window->spin_generations
02265     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02266   gtk_widget_set_tooltip_text
02267     (GTK_WIDGET (window->spin_generations),
02268      _("Number of generations for the genetic algorithm"));
02269   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02270   window->spin_mutation
02271     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02272   gtk_widget_set_tooltip_text
02273     (GTK_WIDGET (window->spin_mutation),
02274      _("Ratio of mutation for the genetic algorithm"));
02275   window->label_reproduction
02276     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02277   window->spin_reproduction
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279   gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_reproduction),
02281      _("Ratio of reproduction for the genetic algorithm"));
02282   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02283   window->spin_adaptation
02284     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02285   gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_adaptation),
02287      _("Ratio of adaptation for the genetic algorithm"));
02288   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02289   window->spin_threshold = (GtkSpinButton *)
02290     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02291                                     precision[DEFAULT_PRECISION]);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_threshold),
02294      _("Threshold in the objective function to finish the simulations"));
02295   window->scrolled_threshold =
02296     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02297   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
```

```
02298                     GTK_WIDGET (window->spin_threshold));
02299 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02300 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02301 //                         GTK_ALIGN_FILL);
02302
02303   // Creating the direction search method properties
02304   window->check_direction = (GtkCheckButton *)
02305     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02306   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02307   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02308   window->button_direction[0] = (GtkRadioButton *)
02309     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02310   gtk_grid_attach (window->grid_direction,
02311                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02312   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02313                     NULL);
02314   for (i = 0; ++i < NDIRECTIONS;)
02315     {
02316       window->button_direction[i] = (GtkRadioButton *)
02317         gtk_radio_button_new_with_mnemonic
02318         (gtk_radio_button_get_group (window->button_direction[0]),
02319          label_direction[i]);
02320       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02321                                    tip_direction[i]);
02322       gtk_grid_attach (window->grid_direction,
02323                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02324       g_signal_connect (window->button_direction[i], "clicked",
02325                         window_update, NULL);
02326     }
02327   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02328   window->spin_steps = (GtkSpinButton *)
02329     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02330   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02331   window->label_estimates
02332     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02333   window->spin_estimates = (GtkSpinButton *)
02334     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02335   window->label_relaxation
02336     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02337   window->spin_relaxation = (GtkSpinButton *)
02338     gtk_spin_button_new_with_range (0., 2., 0.001);
02339   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->label_steps),
02340                    0, NDIRECTIONS, 1, 1);
02341   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->spin_steps),
02342                    1, NDIRECTIONS, 1, 1);
02343   gtk_grid_attach (window->grid_direction,
02344                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02345                    1, 1);
02346   gtk_grid_attach (window->grid_direction,
02347                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02348                    1);
02349   gtk_grid_attach (window->grid_direction,
02350                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02351                    1, 1);
02352   gtk_grid_attach (window->grid_direction,
02353                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02354                    1, 1);
02355
02356   // Creating the array of algorithms
02357   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02358   window->button_algorithm[0] = (GtkRadioButton *)
02359     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02360   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02361                                tip_algorithm[0]);
02362   gtk_grid_attach (window->grid_algorithm,
02363                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02364   g_signal_connect (window->button_algorithm[0], "clicked",
02365                     window_set_algorithm, NULL);
02366   for (i = 0; ++i < NALGORITHMS;)
02367     {
02368       window->button_algorithm[i] = (GtkRadioButton *)
02369         gtk_radio_button_new_with_mnemonic
02370         (gtk_radio_button_get_group (window->button_algorithm[0]),
02371          label_algorithm[i]);
02372       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02373                                    tip_algorithm[i]);
02374       gtk_grid_attach (window->grid_algorithm,
02375                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02376       g_signal_connect (window->button_algorithm[i], "clicked",
02377                         window_set_algorithm, NULL);
02378     }
02379   gtk_grid_attach (window->grid_algorithm,
02380                    GTK_WIDGET (window->label_simulations),
```

```
02381                     0, NALGORITHMS, 1, 1);
02382   gtk_grid_attach (window->grid_algorithm,
02383                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02384   gtk_grid_attach (window->grid_algorithm,
02385                    GTK_WIDGET (window->label_iterations),
02386                    0, NALGORITHMS + 1, 1, 1);
02387   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_iterations),
02388                    1, NALGORITHMS + 1, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_tolerance),
02390                    0, NALGORITHMS + 2, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_tolerance),
02392                    1, NALGORITHMS + 2, 1, 1);
02393   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests), 0,
02394                    NALGORITHMS + 3, 1, 1);
02395   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02396                    NALGORITHMS + 3, 1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_population),
02399                    0, NALGORITHMS + 4, 1, 1);
02400   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_population),
02401                    1, NALGORITHMS + 4, 1, 1);
02402   gtk_grid_attach (window->grid_algorithm,
02403                    GTK_WIDGET (window->label_generations),
02404                    0, NALGORITHMS + 5, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->spin_generations),
02407                    1, NALGORITHMS + 5, 1, 1);
02408   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_mutation),
02409                    0, NALGORITHMS + 6, 1, 1);
02410   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02411                    1, NALGORITHMS + 6, 1, 1);
02412   gtk_grid_attach (window->grid_algorithm,
02413                    GTK_WIDGET (window->label_reproduction),
02414                    0, NALGORITHMS + 7, 1, 1);
02415   gtk_grid_attach (window->grid_algorithm,
02416                    GTK_WIDGET (window->spin_reproduction),
02417                    1, NALGORITHMS + 7, 1, 1);
02418   gtk_grid_attach (window->grid_algorithm,
02419                    GTK_WIDGET (window->label_adaptation),
02420                    0, NALGORITHMS + 8, 1, 1);
02421   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_adaptation),
02422                    1, NALGORITHMS + 8, 1, 1);
02423   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->check_direction),
02424                    0, NALGORITHMS + 9, 2, 1);
02425   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->grid_direction),
02426                    0, NALGORITHMS + 10, 2, 1);
02427   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_threshold),
02428                    0, NALGORITHMS + 11, 1, 1);
02429   gtk_grid_attach (window->grid_algorithm,
02430                    GTK_WIDGET (window->scrolled_threshold),
02431                    1, NALGORITHMS + 11, 1, 1);
02432   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02433   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02434                      GTK_WIDGET (window->grid_algorithm));
02435
02436   // Creating the variable widgets
02437   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02438   gtk_widget_set_tooltip_text
02439     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02440   window->id_variable = g_signal_connect
02441     (window->combo_variable, "changed", window_set_variable, NULL);
02442   window->button_add_variable
02443     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02444                                                    GTK_ICON_SIZE_BUTTON);
02445   g_signal_connect
02446     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02447   gtk_widget_set_tooltip_text
02448     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02449   window->button_remove_variable
02450     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02451                                                    GTK_ICON_SIZE_BUTTON);
02452   g_signal_connect
02453     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
```

```
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02456   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02457   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02458   gtk_widget_set_tooltip_text
02459     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02460   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02461   window->id_variable_label = g_signal_connect
02462     (window->entry_variable, "changed",
      window_label_variable, NULL);
02463   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02464   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02465     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02466   gtk_widget_set_tooltip_text
02467     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02468   window->scrolled_min
02469     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02470   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02471                      GTK_WIDGET (window->spin_min));
02472   g_signal_connect (window->spin_min, "value-changed",
02473                     window_rangemin_variable, NULL);
02474   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02475   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02476     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02477   gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02479   window->scrolled_max
02480     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02481   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02482                      GTK_WIDGET (window->spin_max));
02483   g_signal_connect (window->spin_max, "value-changed",
02484                     window_rangemax_variable, NULL);
02485   window->check_minabs = (GtkCheckButton *)
02486     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02487   g_signal_connect (window->check_minabs, "toggled",
      window_update, NULL);
02488   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02489     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02490   gtk_widget_set_tooltip_text
02491     (GTK_WIDGET (window->spin_minabs),
02492      _("Minimum allowed value of the variable"));
02493   window->scrolled_minabs
02494     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02495   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02496                      GTK_WIDGET (window->spin_minabs));
02497   g_signal_connect (window->spin_minabs, "value-changed",
02498                     window_rangeminabs_variable, NULL);
02499   window->check_maxabs = (GtkCheckButton *)
02500     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02501   g_signal_connect (window->check_maxabs, "toggled",
      window_update, NULL);
02502   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02503     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02504   gtk_widget_set_tooltip_text
02505     (GTK_WIDGET (window->spin_maxabs),
02506      _("Maximum allowed value of the variable"));
02507   window->scrolled_maxabs
02508     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02509   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02510                      GTK_WIDGET (window->spin_maxabs));
02511   g_signal_connect (window->spin_maxabs, "value-changed",
02512                     window_rangemaxabs_variable, NULL);
02513   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02514   window->spin_precision = (GtkSpinButton *)
02515     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02516   gtk_widget_set_tooltip_text
02517     (GTK_WIDGET (window->spin_precision),
02518      _("Number of precision floating point digits\n"
02519        "0 is for integer numbers"));
02520   g_signal_connect (window->spin_precision, "value-changed",
02521                     window_precision_variable, NULL);
02522   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02523   window->spin_sweeps =
02524     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02525   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02526                                _("Number of steps sweeping the variable"));
02527   g_signal_connect (window->spin_sweeps, "value-changed",
02528                     window_update_variable, NULL);
02529   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02530   window->spin_bits
02531     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_bits),
02534      _("Number of bits to encode the variable"));
02535   g_signal_connect
02536     (window->spin_bits, "value-changed", window_update_variable, NULL)
      ;
```

```
02537   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02538   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02539     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02540   gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_step),
02542      _("Initial step size for the direction search method"));
02543   window->scrolled_step
02544     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02545   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02546                      GTK_WIDGET (window->spin_step));
02547   g_signal_connect
02548     (window->spin_step, "value-changed", window_step_variable, NULL);
02549   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02550   gtk_grid_attach (window->grid_variable,
02551                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02552   gtk_grid_attach (window->grid_variable,
02553                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02554   gtk_grid_attach (window->grid_variable,
02555                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02556   gtk_grid_attach (window->grid_variable,
02557                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02558   gtk_grid_attach (window->grid_variable,
02559                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02560   gtk_grid_attach (window->grid_variable,
02561                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02562   gtk_grid_attach (window->grid_variable,
02563                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02564   gtk_grid_attach (window->grid_variable,
02565                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02592   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02593   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02594                      GTK_WIDGET (window->grid_variable));
02595
02596   // Creating the experiment widgets
02597   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02598   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02599                                _("Experiment selector"));
02600   window->id_experiment = g_signal_connect
02601     (window->combo_experiment, "changed",
      window_set_experiment, NULL);
02602   window->button_add_experiment
02603     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02604                                                    GTK_ICON_SIZE_BUTTON);
02605   g_signal_connect
02606     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02607   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02608                                _("Add experiment"));
02609   window->button_remove_experiment
02610     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02611                                                    GTK_ICON_SIZE_BUTTON);
02612   g_signal_connect (window->button_remove_experiment, "clicked",
02613                     window_remove_experiment, NULL);
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02615                                _("Remove experiment"));
02616   window->label_experiment
02617     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02618   window->button_experiment = (GtkFileChooserButton *)
02619     gtk_file_chooser_button_new (_("Experimental data file"),
02620                                  GTK_FILE_CHOOSER_ACTION_OPEN);
```

```
02621   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02622                                _("Experimental data file"));
02623   window->id_experiment_name
02624     = g_signal_connect (window->button_experiment, "selection-changed",
02625                         window_name_experiment, NULL);
02626   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02627   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02628   window->spin_weight
02629     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02630   gtk_widget_set_tooltip_text
02631     (GTK_WIDGET (window->spin_weight),
02632      _("Weight factor to build the objective function"));
02633   g_signal_connect
02634     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02635   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02636   gtk_grid_attach (window->grid_experiment,
02637                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02638   gtk_grid_attach (window->grid_experiment,
02639                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02640   gtk_grid_attach (window->grid_experiment,
02641                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02642   gtk_grid_attach (window->grid_experiment,
02643                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02644   gtk_grid_attach (window->grid_experiment,
02645                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02646   gtk_grid_attach (window->grid_experiment,
02647                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02648   gtk_grid_attach (window->grid_experiment,
02649                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02650   for (i = 0; i < MAX_NINPUTS; ++i)
02651     {
02652       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02653       window->check_template[i] = (GtkCheckButton *)
02654         gtk_check_button_new_with_label (buffer3);
02655       window->id_template[i]
02656         = g_signal_connect (window->check_template[i], "toggled",
02657                             window_inputs_experiment, NULL);
02658       gtk_grid_attach (window->grid_experiment,
02659                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02660       window->button_template[i] =
02661         (GtkFileChooserButton *)
02662         gtk_file_chooser_button_new (_("Input template"),
02663                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02664       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02665                                    _("Experimental input template file"));
02666       window->id_input[i] =
02667         g_signal_connect_swapped (window->button_template[i],
02668                                   "selection-changed",
02669                                   (GCallback) window_template_experiment,
02670                                   (void *) (size_t) i);
02671       gtk_grid_attach (window->grid_experiment,
02672                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02673     }
02674   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02675   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02676                      GTK_WIDGET (window->grid_experiment));
02677
02678   // Creating the error norm widgets
02679   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02680   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02681   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02682                      GTK_WIDGET (window->grid_norm));
02683   window->button_norm[0] = (GtkRadioButton *)
02684     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02685   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02686                                tip_norm[0]);
02687   gtk_grid_attach (window->grid_norm,
02688                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02689   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02690   for (i = 0; ++i < NNORMS;)
02691     {
02692       window->button_norm[i] = (GtkRadioButton *)
02693         gtk_radio_button_new_with_mnemonic
02694         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02695       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02696                                    tip_norm[i]);
02697       gtk_grid_attach (window->grid_norm,
02698                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02699       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02700     }
02701   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02702   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1, 1);
```

```
02703   window->spin_p =
02704     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02705                                                       G_MAXDOUBLE, 0.01);
02706   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02707                                _("P parameter for the P error norm"));
02708   window->scrolled_p =
02709     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02710   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02711                      GTK_WIDGET (window->spin_p));
02712   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02713   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02714   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
    scrolled_p),
02715                    1, 2, 1, 2);
02716
02717   // Creating the grid and attaching the widgets to the grid
02718   window->grid = (GtkGrid *) gtk_grid_new ();
02719   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02720   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02721   gtk_grid_attach (window->grid,
02722                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02723   gtk_grid_attach (window->grid,
02724                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02725   gtk_grid_attach (window->grid,
02726                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02727   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02728   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
    window->grid));
02729
02730   // Setting the window logo
02731   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02732   gtk_window_set_icon (window->window, window->logo);
02733
02734   // Showing the window
02735   gtk_widget_show_all (GTK_WIDGET (window->window));
02736
02737   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02738 #if GTK_MINOR_VERSION >= 16
02739   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02740   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02741   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02742   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02743   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02744   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02745   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02746 #endif
02747
02748   // Reading initial example
02749   input_new ();
02750   buffer2 = g_get_current_dir ();
02751   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02752   g_free (buffer2);
02753   window_read (buffer);
02754   g_free (buffer);
02755
02756 #if DEBUG_INTERFACE
02757   fprintf (stderr, "window_new: start\n");
02758 #endif
02759 }
```

4.13.2.15 window_open()

```
void window_open ( )
```

Function to open the input data.

Definition at line 1981 of file interface.c.

```
01982 {
01983   GtkFileChooserDialog *dlg;
01984   GtkFileFilter *filter;
01985   char *buffer, *directory, *name;
01986
01987 #if DEBUG_INTERFACE
01988   fprintf (stderr, "window_open: start\n");
```

```
01989 #endif
01990
01991   // Saving a backup of the current input file
01992   directory = g_strdup (input->directory);
01993   name = g_strdup (input->name);
01994
01995   // Opening dialog
01996   dlg = (GtkFileChooserDialog *)
01997     gtk_file_chooser_dialog_new (_("Open input file"),
01998                                  window->window,
01999                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02000                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02001                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02002
02003   // Adding XML filter
02004   filter = (GtkFileFilter *) gtk_file_filter_new ();
02005   gtk_file_filter_set_name (filter, "XML");
02006   gtk_file_filter_add_pattern (filter, "*.xml");
02007   gtk_file_filter_add_pattern (filter, "*.XML");
02008   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02009
02010   // Adding JSON filter
02011   filter = (GtkFileFilter *) gtk_file_filter_new ();
02012   gtk_file_filter_set_name (filter, "JSON");
02013   gtk_file_filter_add_pattern (filter, "*.json");
02014   gtk_file_filter_add_pattern (filter, "*.JSON");
02015   gtk_file_filter_add_pattern (filter, "*.js");
02016   gtk_file_filter_add_pattern (filter, "*.JS");
02017   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019   // If OK saving
02020   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02021     {
02022
02023       // Traying to open the input file
02024       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02025       if (!window_read (buffer))
02026         {
02027 #if DEBUG_INTERFACE
02028           fprintf (stderr, "window_open: error reading input file\n");
02029 #endif
02030           g_free (buffer);
02031
02032           // Reading backup file on error
02033           buffer = g_build_filename (directory, name, NULL);
02034           input->result = input->variables = NULL;
02035           if (!input_open (buffer))
02036             {
02037
02038               // Closing on backup file reading error
02039 #if DEBUG_INTERFACE
02040               fprintf (stderr, "window_read: error reading backup file\n");
02041 #endif
02042               g_free (buffer);
02043               break;
02044             }
02045           g_free (buffer);
02046         }
02047       else
02048         {
02049           g_free (buffer);
02050           break;
02051         }
02052     }
02053
02054   // Freeing and closing
02055   g_free (name);
02056   g_free (directory);
02057   gtk_widget_destroy (GTK_WIDGET (dlg));
02058 #if DEBUG_INTERFACE
02059   fprintf (stderr, "window_open: end\n");
02060 #endif
02061 }
```

4.13.2.16 window_precision_variable()

```
void window_precision_variable ( )
```

Function to update the variable precision in the main window.

Definition at line 1715 of file interface.c.

```
01716 {
01717   unsigned int i;
01718 #if DEBUG_INTERFACE
01719   fprintf (stderr, "window_precision_variable: start\n");
01720 #endif
01721   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01722   input->variable[i].precision
01723     = (unsigned int) gtk_spin_button_get_value_as_int (window->
      spin_precision);
01724   gtk_spin_button_set_digits (window->spin_min, input->
      variable[i].precision);
01725   gtk_spin_button_set_digits (window->spin_max, input->
      variable[i].precision);
01726   gtk_spin_button_set_digits (window->spin_minabs,
01727                               input->variable[i].precision);
01728   gtk_spin_button_set_digits (window->spin_maxabs,
01729                               input->variable[i].precision);
01730 #if DEBUG_INTERFACE
01731   fprintf (stderr, "window_precision_variable: end\n");
01732 #endif
01733 }
```

4.13.2.17 window_rangemax_variable()

```
void window_rangemax_variable ( )
```

Function to update the variable rangemax in the main window.

Definition at line 1756 of file interface.c.

```
01757 {
01758   unsigned int i;
01759 #if DEBUG_INTERFACE
01760   fprintf (stderr, "window_rangemax_variable: start\n");
01761 #endif
01762   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01763   input->variable[i].rangemax = gtk_spin_button_get_value (
      window->spin_max);
01764 #if DEBUG_INTERFACE
01765   fprintf (stderr, "window_rangemax_variable: end\n");
01766 #endif
01767 }
```

4.13.2.18 window_rangemaxabs_variable()

```
void window_rangemaxabs_variable ( )
```

Function to update the variable rangemaxabs in the main window.

Definition at line 1791 of file interface.c.

```
01792 {
01793   unsigned int i;
01794 #if DEBUG_INTERFACE
01795   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01796 #endif
01797   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01798   input->variable[i].rangemaxabs
01799     = gtk_spin_button_get_value (window->spin_maxabs);
01800 #if DEBUG_INTERFACE
01801   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01802 #endif
01803 }
```

4.13.2.19 window_rangemin_variable()

```
void window_rangemin_variable ( )
```

Function to update the variable rangemin in the main window.

Definition at line 1739 of file interface.c.

```
01740 {
01741   unsigned int i;
01742 #if DEBUG_INTERFACE
01743   fprintf (stderr, "window_rangemin_variable: start\n");
01744 #endif
01745   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01746   input->variable[i].rangemin = gtk_spin_button_get_value (
      window->spin_min);
01747 #if DEBUG_INTERFACE
01748   fprintf (stderr, "window_rangemin_variable: end\n");
01749 #endif
01750 }
```

4.13.2.20 window_rangeminabs_variable()

```
void window_rangeminabs_variable ( )
```

Function to update the variable rangeminabs in the main window.

Definition at line 1773 of file interface.c.

```
01774 {
01775   unsigned int i;
01776 #if DEBUG_INTERFACE
01777   fprintf (stderr, "window_rangeminabs_variable: start\n");
01778 #endif
01779   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01780   input->variable[i].rangeminabs
01781     = gtk_spin_button_get_value (window->spin_minabs);
01782 #if DEBUG_INTERFACE
01783   fprintf (stderr, "window_rangeminabs_variable: end\n");
01784 #endif
01785 }
```

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

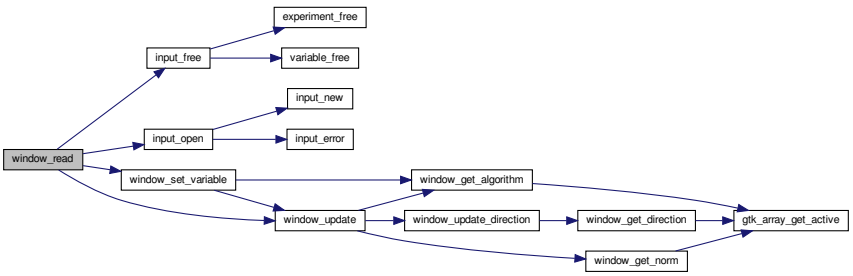
filename	File name.

Definition at line 1865 of file interface.c.

```
01866 {
01867   unsigned int i;
01868   char *buffer;
01869 #if DEBUG_INTERFACE
01870   fprintf (stderr, "window_read: start\n");
01871 #endif
01872
01873   // Reading new input file
01874   input_free ();
01875   input->result = input->variables = NULL;
01876   if (!input_open (filename))
01877     {
01878 #if DEBUG_INTERFACE
01879       fprintf (stderr, "window_read: end\n");
01880 #endif
01881       return 0;
01882     }
01883
01884   // Setting GTK+ widgets data
01885   gtk_entry_set_text (window->entry_result, input->result);
01886   gtk_entry_set_text (window->entry_variables, input->
      variables);
01887   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01888   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01889                                  (window->button_simulator), buffer);
01890   g_free (buffer);
01891   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01892                                 (size_t) input->evaluator);
01893   if (input->evaluator)
01894     {
01895       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01896       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01897                                      (window->button_evaluator), buffer);
01898       g_free (buffer);
01899     }
01900   gtk_toggle_button_set_active
01901     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01902   switch (input->algorithm)
01903     {
01904     case ALGORITHM_MONTE_CARLO:
01905       gtk_spin_button_set_value (window->spin_simulations,
01906                                  (gdouble) input->nsimulations);
01907       // fallthrough
01908     case ALGORITHM_SWEEP:
01909     case ALGORITHM_ORTHOGONAL:
01910       gtk_spin_button_set_value (window->spin_iterations,
01911                                  (gdouble) input->niterations);
01912       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01913       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01914       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01915                                     (window->check_direction),
      input->nsteps);
01916       if (input->nsteps)
01917         {
01918           gtk_toggle_button_set_active
01919             (GTK_TOGGLE_BUTTON (window->button_direction
01920                                 [input->direction]), TRUE);
01921           gtk_spin_button_set_value (window->spin_steps,
01922                                      (gdouble) input->nsteps);
01923           gtk_spin_button_set_value (window->spin_relaxation,
01924                                      (gdouble) input->relaxation);
01925           switch (input->direction)
01926             {
01927             case DIRECTION_METHOD_RANDOM:
01928               gtk_spin_button_set_value (window->spin_estimates,
01929                                          (gdouble) input->nestimates);
01930             }
01931         }
01932       break;
01933     default:
01934       gtk_spin_button_set_value (window->spin_population,
```

```
01935                                (gdouble) input->nsimulations);
01936       gtk_spin_button_set_value (window->spin_generations,
01937                                (gdouble) input->niterations);
01938       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01939       gtk_spin_button_set_value (window->spin_reproduction,
01940                                input->reproduction_ratio);
01941       gtk_spin_button_set_value (window->spin_adaptation,
01942                                input->adaptation_ratio);
01943     }
01944   gtk_toggle_button_set_active
01945     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01946   gtk_spin_button_set_value (window->spin_p, input->p);
01947   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01948   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01949   g_signal_handler_block (window->button_experiment,
01950                           window->id_experiment_name);
01951   gtk_combo_box_text_remove_all (window->combo_experiment);
01952   for (i = 0; i < input->nexperiments; ++i)
01953     gtk_combo_box_text_append_text (window->combo_experiment,
01954                                     input->experiment[i].name);
01955   g_signal_handler_unblock
01956     (window->button_experiment, window->
      id_experiment_name);
01957   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01958   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01959   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01960   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01961   gtk_combo_box_text_remove_all (window->combo_variable);
01962   for (i = 0; i < input->nvariables; ++i)
01963     gtk_combo_box_text_append_text (window->combo_variable,
01964                                     input->variable[i].name);
01965   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01966   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01967   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01968   window_set_variable ();
01969   window_update ();
01970
01971 #if DEBUG_INTERFACE
01972   fprintf (stderr, "window_read: end\n");
01973 #endif
01974   return 1;
01975 }
```

Here is the call graph for this function:



4.13.2.22 window_remove_experiment()

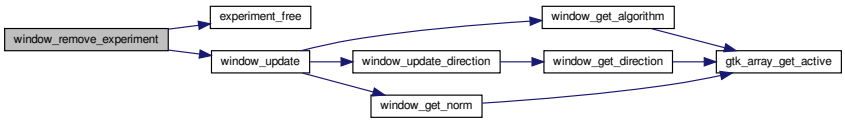
```
void window_remove_experiment ( )
```

Function to remove an experiment in the main window.

Definition at line 1357 of file interface.c.

```
01358 {
01359   unsigned int i, j;
01360 #if DEBUG_INTERFACE
01361   fprintf (stderr, "window_remove_experiment: start\n");
01362 #endif
01363   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01364   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01365   gtk_combo_box_text_remove (window->combo_experiment, i);
01366   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01367   experiment_free (input->experiment + i, input->
      type);
01368   --input->nexperiments;
01369   for (j = i; j < input->nexperiments; ++j)
01370     memcpy (input->experiment + j, input->experiment + j + 1,
01371             sizeof (Experiment));
01372   j = input->nexperiments - 1;
01373   if (i > j)
01374     i = j;
01375   for (j = 0; j < input->experiment->ninputs; ++j)
01376     g_signal_handler_block (window->button_template[j],
      window->id_input[j]);
01377   g_signal_handler_block
01378     (window->button_experiment, window->
      id_experiment_name);
01379   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01380   g_signal_handler_unblock
01381     (window->button_experiment, window->
      id_experiment_name);
01382   for (j = 0; j < input->experiment->ninputs; ++j)
01383     g_signal_handler_unblock (window->button_template[j],
      window->id_input[j]);
01384   window_update ();
01385 #if DEBUG_INTERFACE
01386   fprintf (stderr, "window_remove_experiment: end\n");
01387 #endif
01388 }
```

Here is the call graph for this function:



4.13.2.23 window_remove_variable()

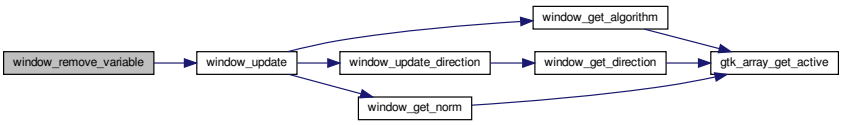
```
void window_remove_variable ( )
```

Function to remove a variable in the main window.

Definition at line 1627 of file interface.c.

```
01628 {
01629   unsigned int i, j;
01630 #if DEBUG_INTERFACE
01631   fprintf (stderr, "window_remove_variable: start\n");
01632 #endif
01633   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01634   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01635   gtk_combo_box_text_remove (window->combo_variable, i);
01636   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01637   xmlFree (input->variable[i].name);
01638   --input->nvariables;
01639   for (j = i; j < input->nvariables; ++j)
01640     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01641   j = input->nvariables - 1;
01642   if (i > j)
01643     i = j;
01644   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01645   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01646   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01647   window_update ();
01648 #if DEBUG_INTERFACE
01649   fprintf (stderr, "window_remove_variable: end\n");
01650 #endif
01651 }
```

Here is the call graph for this function:



4.13.2.24 window_run()

```
void window_run ( )
```

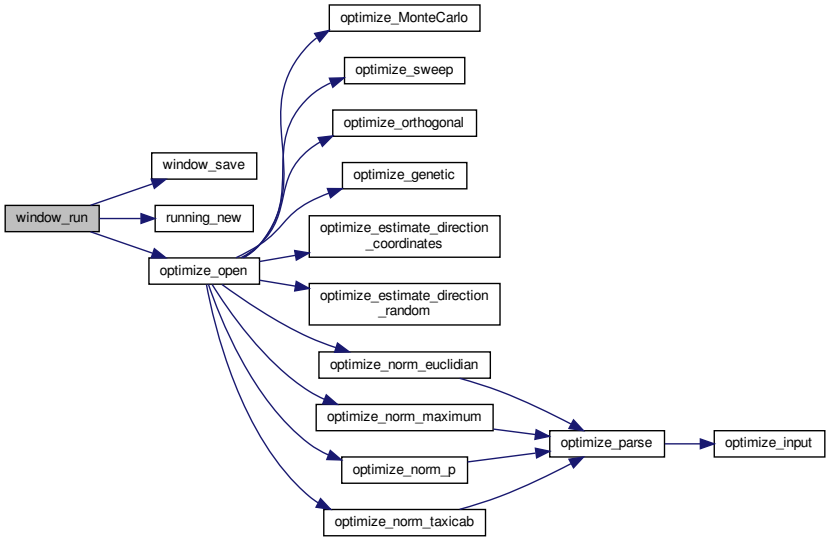
Function to run a optimization.

Definition at line 975 of file interface.c.

```
00976 {
00977   unsigned int i;
00978   char *msg, *msg2, buffer[64], buffer2[64];
00979 #if DEBUG_INTERFACE
00980   fprintf (stderr, "window_run: start\n");
00981 #endif
00982   if (!window_save ())
00983     {
00984 #if DEBUG_INTERFACE
00985       fprintf (stderr, "window_run: end\n");
00986 #endif
00987       return;
00988     }
00989   running_new ();
00990   while (gtk_events_pending ())
00991     gtk_main_iteration ();
00992   optimize_open ();
00993 #if DEBUG_INTERFACE
00994   fprintf (stderr, "window_run: closing running dialog\n");
00995 #endif
```

```
00996   gtk_spinner_stop (running->spinner);
00997   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00998 #if DEBUG_INTERFACE
00999   fprintf (stderr, "window_run: displaying results\n");
01000 #endif
01001   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01002   msg2 = g_strdup (buffer);
01003   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01004     {
01005       snprintf (buffer, 64, "%s = %s\n",
01006                 input->variable[i].name, format[input->
      variable[i].precision]);
01007       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01008       msg = g_strconcat (msg2, buffer2, NULL);
01009       g_free (msg2);
01010     }
01011   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01012             optimize->calculation_time);
01013   msg = g_strconcat (msg2, buffer, NULL);
01014   g_free (msg2);
01015   show_message (_("Best result"), msg, INFO_TYPE);
01016   g_free (msg);
01017 #if DEBUG_INTERFACE
01018   fprintf (stderr, "window_run: freeing memory\n");
01019 #endif
01020   optimize_free ();
01021 #if DEBUG_INTERFACE
01022   fprintf (stderr, "window_run: end\n");
01023 #endif
01024 }
```

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 824 of file interface.c.

```
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (_("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00840                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00841   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00842   buffer = g_build_filename (input->directory, input->name, NULL);
00843   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00844   g_free (buffer);
00845
00846   // Adding XML filter
00847   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00848   gtk_file_filter_set_name (filter1, "XML");
00849   gtk_file_filter_add_pattern (filter1, "*.xml");
00850   gtk_file_filter_add_pattern (filter1, "*.XML");
00851   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00852
00853   // Adding JSON filter
00854   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00855   gtk_file_filter_set_name (filter2, "JSON");
00856   gtk_file_filter_add_pattern (filter2, "*.json");
00857   gtk_file_filter_add_pattern (filter2, "*.JSON");
00858   gtk_file_filter_add_pattern (filter2, "*.js");
00859   gtk_file_filter_add_pattern (filter2, "*.JS");
00860   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00861
00862   if (input->type == INPUT_TYPE_XML)
00863     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00864   else
00865     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00866
00867   // If OK response then saving
00868   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00869     {
00870       // Setting input file type
00871       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00872       buffer = (char *) gtk_file_filter_get_name (filter1);
00873       if (!strcmp (buffer, "XML"))
00874         input->type = INPUT_TYPE_XML;
00875       else
00876         input->type = INPUT_TYPE_JSON;
00877
00878       // Adding properties to the root XML node
00879       input->simulator = gtk_file_chooser_get_filename
00880         (GTK_FILE_CHOOSER (window->button_simulator));
00881       if (gtk_toggle_button_get_active
00882           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00883         input->evaluator = gtk_file_chooser_get_filename
00884           (GTK_FILE_CHOOSER (window->button_evaluator));
00885       else
00886         input->evaluator = NULL;
00887       if (input->type == INPUT_TYPE_XML)
00888         {
00889           input->result
00890             = (char *) xmlStrdup ((const xmlChar *)
00891                                   gtk_entry_get_text (window->entry_result));
00892           input->variables
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->
    entry_variables));
00895         }
00896       else
00897         {
00898           input->result = g_strdup (gtk_entry_get_text (window->
    entry_result));
00899           input->variables =
00900             g_strdup (gtk_entry_get_text (window->entry_variables));
```

```
00901         }
00902
00903       // Setting the algorithm
00904       switch (window_get_algorithm ())
00905         {
00906         case ALGORITHM_MONTE_CARLO:
00907           input->algorithm = ALGORITHM_MONTE_CARLO;
00908           input->nsimulations
00909             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00910           input->niterations
00911             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00912           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00913           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00914           window_save_direction ();
00915           break;
00916         case ALGORITHM_SWEEP:
00917           input->algorithm = ALGORITHM_SWEEP;
00918           input->niterations
00919             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00920           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00921           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00922           window_save_direction ();
00923           break;
00924         case ALGORITHM_ORTHOGONAL:
00925           input->algorithm = ALGORITHM_ORTHOGONAL;
00926           input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00928           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00929           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00930           window_save_direction ();
00931           break;
00932         default:
00933           input->algorithm = ALGORITHM_GENETIC;
00934           input->nsimulations
00935             = gtk_spin_button_get_value_as_int (window->spin_population);
00936           input->niterations
00937             = gtk_spin_button_get_value_as_int (window->spin_generations);
00938           input->mutation_ratio
00939             = gtk_spin_button_get_value (window->spin_mutation);
00940           input->reproduction_ratio
00941             = gtk_spin_button_get_value (window->spin_reproduction);
00942           input->adaptation_ratio
00943             = gtk_spin_button_get_value (window->spin_adaptation);
00944           break;
00945         }
00946       input->norm = window_get_norm ();
00947       input->p = gtk_spin_button_get_value (window->spin_p);
00948       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00949
00950       // Saving the XML file
00951       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00952       input_save (buffer);
00953
00954       // Closing and freeing memory
00955       g_free (buffer);
00956       gtk_widget_destroy (GTK_WIDGET (dlg));
00957 #if DEBUG_INTERFACE
00958       fprintf (stderr, "window_save: end\n");
00959 #endif
00960       return 1;
00961     }
00962
00963   // Closing and freeing memory
00964   gtk_widget_destroy (GTK_WIDGET (dlg));
00965 #if DEBUG_INTERFACE
00966   fprintf (stderr, "window_save: end\n");
00967 #endif
00968   return 0;
00969 }
```

4.13.2.26 window_save_direction()

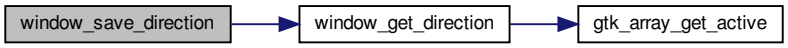
```
void window_save_direction ( )
```

Function to save the direction search method data in the input file.

Definition at line 790 of file interface.c.

```
00791 {
00792 #if DEBUG_INTERFACE
00793   fprintf (stderr, "window_save_direction: start\n");
00794 #endif
00795   if (gtk_toggle_button_get_active
00796       (GTK_TOGGLE_BUTTON (window->check_direction)))
00797     {
00798       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00799       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00800       switch (window_get_direction ())
00801         {
00802         case DIRECTION_METHOD_COORDINATES:
00803           input->direction = DIRECTION_METHOD_COORDINATES;
00804           break;
00805         default:
00806           input->direction = DIRECTION_METHOD_RANDOM;
00807           input->nestimates
00808             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00809         }
00810     }
00811   else
00812     input->nsteps = 0;
00813 #if DEBUG_INTERFACE
00814   fprintf (stderr, "window_save_direction: end\n");
00815 #endif
00816 }
```

Here is the call graph for this function:



4.13.2.27 window_set_algorithm()

```
void window_set_algorithm ( )
```

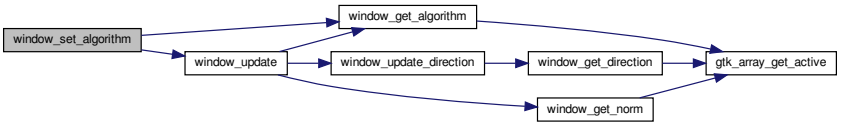
Function to avoid memory errors changing the algorithm.

Definition at line 1283 of file interface.c.

```
01284 {
01285   int i;
01286 #if DEBUG_INTERFACE
01287   fprintf (stderr, "window_set_algorithm: start\n");
01288 #endif
01289   i = window_get_algorithm ();
01290   switch (i)
01291     {
01292     case ALGORITHM_SWEEP:
01293     case ALGORITHM_ORTHOGONAL:
01294       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01295       if (i < 0)
01296         i = 0;
01297       gtk_spin_button_set_value (window->spin_sweeps,
01298                                  (gdouble) input->variable[i].
```

```
    nsweeps);
01299       break;
01300     case ALGORITHM_GENETIC:
01301       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01302       if (i < 0)
01303         i = 0;
01304       gtk_spin_button_set_value (window->spin_bits,
01305                                  (gdouble) input->variable[i].nbits);
01306     }
01307   window_update ();
01308 #if DEBUG_INTERFACE
01309   fprintf (stderr, "window_set_algorithm: end\n");
01310 #endif
01311 }
```

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 1317 of file interface.c.

```
01318 {
01319   unsigned int i, j;
01320   char *buffer1, *buffer2;
01321 #if DEBUG_INTERFACE
01322   fprintf (stderr, "window_set_experiment: start\n");
01323 #endif
01324   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01325   gtk_spin_button_set_value (window->spin_weight, input->
    experiment[i].weight);
01326   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01327   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01328   g_free (buffer1);
01329   g_signal_handler_block
01330     (window->button_experiment, window->
    id_experiment_name);
01331   gtk_file_chooser_set_filename
01332     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01333   g_signal_handler_unblock
01334     (window->button_experiment, window->
    id_experiment_name);
01335   g_free (buffer2);
01336   for (j = 0; j < input->experiment->ninputs; ++j)
01337     {
01338       g_signal_handler_block (window->button_template[j],
    window->id_input[j]);
01339       buffer2 =
01340         g_build_filename (input->directory, input->experiment[i].
    stencil[j],
01341                           NULL);
01342       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01343                                      (window->button_template[j]), buffer2);
01344       g_free (buffer2);
01345       g_signal_handler_unblock
01346         (window->button_template[j], window->id_input[j]);
01347     }
01348 #if DEBUG_INTERFACE
01349   fprintf (stderr, "window_set_experiment: end\n");
01350 #endif
01351 }
```

4.13.2.29 window_set_variable()

```
void window_set_variable ( )
```

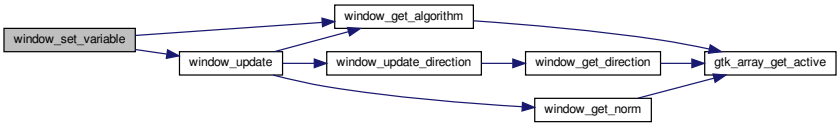
Function to set the variable data in the main window.

Definition at line 1550 of file interface.c.

```
01551 {
01552   unsigned int i;
01553 #if DEBUG_INTERFACE
01554   fprintf (stderr, "window_set_variable: start\n");
01555 #endif
01556   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01557   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01558   gtk_entry_set_text (window->entry_variable, input->
      variable[i].name);
01559   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01560   gtk_spin_button_set_value (window->spin_min, input->
      variable[i].rangemin);
01561   gtk_spin_button_set_value (window->spin_max, input->
      variable[i].rangemax);
01562   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01563     {
01564       gtk_spin_button_set_value (window->spin_minabs,
01565                                  input->variable[i].rangeminabs);
01566       gtk_toggle_button_set_active
01567         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01568     }
01569   else
01570     {
01571       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01572       gtk_toggle_button_set_active
01573         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01574     }
01575   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01576     {
01577       gtk_spin_button_set_value (window->spin_maxabs,
01578                                  input->variable[i].rangemaxabs);
01579       gtk_toggle_button_set_active
01580         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01581     }
01582   else
01583     {
01584       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01585       gtk_toggle_button_set_active
01586         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01587     }
01588   gtk_spin_button_set_value (window->spin_precision,
01589                              input->variable[i].precision);
01590   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01591   if (input->nsteps)
01592     gtk_spin_button_set_value (window->spin_step, input->
      variable[i].step);
01593 #if DEBUG_INTERFACE
01594   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01595            input->variable[i].precision);
01596 #endif
01597   switch (window_get_algorithm ())
01598     {
01599     case ALGORITHM_SWEEP:
01600     case ALGORITHM_ORTHOGONAL:
01601       gtk_spin_button_set_value (window->spin_sweeps,
01602                                  (gdouble) input->variable[i].
      nsweeps);
01603 #if DEBUG_INTERFACE
01604       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605                input->variable[i].nsweeps);
01606 #endif
01607       break;
01608     case ALGORITHM_GENETIC:
01609       gtk_spin_button_set_value (window->spin_bits,
01610                                  (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613                input->variable[i].nbits);
01614 #endif
01615       break;
01616     }
```

```
01617   window_update ();
01618 #if DEBUG_INTERFACE
01619   fprintf (stderr, "window_set_variable: end\n");
01620 #endif
01621 }
```

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 1519 of file interface.c.

```
01521 {
01522   unsigned int i, j;
01523   char *buffer;
01524   GFile *file1, *file2;
01525 #if DEBUG_INTERFACE
01526   fprintf (stderr, "window_template_experiment: start\n");
01527 #endif
01528   i = (size_t) data;
01529   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01530   file1
01531     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01532   file2 = g_file_new_for_path (input->directory);
01533   buffer = g_file_get_relative_path (file2, file1);
01534   if (input->type == INPUT_TYPE_XML)
01535     input->experiment[j].stencil[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01536   else
01537     input->experiment[j].stencil[i] = g_strdup (buffer);
01538   g_free (buffer);
01539   g_object_unref (file2);
01540   g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }
```

4.13.2.31 window_update()

```
void window_update ( )
```

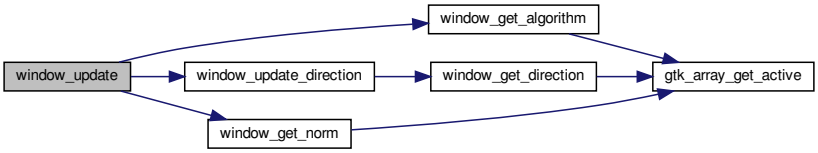
Function to update the main window view.

Definition at line 1126 of file interface.c.

```
01127 {
01128   unsigned int i;
01129 #if DEBUG_INTERFACE
01130   fprintf (stderr, "window_update: start\n");
01131 #endif
01132   gtk_widget_set_sensitive
01133     (GTK_WIDGET (window->button_evaluator),
01134      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01135                                    (window->check_evaluator)));
01136   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01137   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01138   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01139   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01140   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01142   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01144   gtk_widget_hide (GTK_WIDGET (window->label_population));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01146   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01147   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01148   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01150   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01151   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01152   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01153   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01154   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01155   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01156   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01157   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01158   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01159   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01160   gtk_widget_hide (GTK_WIDGET (window->label_step));
01161   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01162   gtk_widget_hide (GTK_WIDGET (window->label_p));
01163   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01164   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01165   switch (window_get_algorithm ())
01166     {
01167     case ALGORITHM_MONTE_CARLO:
01168       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01169       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01170       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01171       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01172       if (i > 1)
01173         {
01174           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01175           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01176           gtk_widget_show (GTK_WIDGET (window->label_bests));
01177           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01178         }
01179       window_update_direction ();
01180       break;
01181     case ALGORITHM_SWEEP:
01182     case ALGORITHM_ORTHOGONAL:
01183       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01184       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01185       if (i > 1)
01186         {
01187           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01188           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01189           gtk_widget_show (GTK_WIDGET (window->label_bests));
01190           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01191         }
01192       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01193       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01194       gtk_widget_show (GTK_WIDGET (window->check_direction));
01195       window_update_direction ();
01196       break;
01197     default:
01198       gtk_widget_show (GTK_WIDGET (window->label_population));
01199       gtk_widget_show (GTK_WIDGET (window->spin_population));
01200       gtk_widget_show (GTK_WIDGET (window->label_generations));
```

```
01201       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01202       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01203       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01204       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01205       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01206       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01207       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01208       gtk_widget_show (GTK_WIDGET (window->label_bits));
01209       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01210     }
01211   gtk_widget_set_sensitive
01212     (GTK_WIDGET (window->button_remove_experiment),
      input->nexperiments > 1);
01213   gtk_widget_set_sensitive
01214     (GTK_WIDGET (window->button_remove_variable),
      input->nvariables > 1);
01215   for (i = 0; i < input->experiment->ninputs; ++i)
01216     {
01217       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01218       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01219       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01220       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01221       g_signal_handler_block
01222         (window->check_template[i], window->
      id_template[i]);
01223       g_signal_handler_block (window->button_template[i],
      window->id_input[i]);
01224       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01225                                     (window->check_template[i]), 1);
01226       g_signal_handler_unblock (window->button_template[i],
01227                                 window->id_input[i]);
01228       g_signal_handler_unblock (window->check_template[i],
01229                                 window->id_template[i]);
01230     }
01231   if (i > 0)
01232     {
01233       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01234       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01235                                 gtk_toggle_button_get_active
01236                                 GTK_TOGGLE_BUTTON (window->check_template
01237                                                    [i - 1]));
01238     }
01239   if (i < MAX_NINPUTS)
01240     {
01241       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01242       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01243       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01244       gtk_widget_set_sensitive
01245         (GTK_WIDGET (window->button_template[i]),
01246          gtk_toggle_button_get_active
01247          GTK_TOGGLE_BUTTON (window->check_template[i]));
01248       g_signal_handler_block
01249         (window->check_template[i], window->
      id_template[i]);
01250       g_signal_handler_block (window->button_template[i],
      window->id_input[i]);
01251       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01252                                     (window->check_template[i]), 0);
01253       g_signal_handler_unblock (window->button_template[i],
01254                                 window->id_input[i]);
01255       g_signal_handler_unblock (window->check_template[i],
01256                                 window->id_template[i]);
01257     }
01258   while (++i < MAX_NINPUTS)
01259     {
01260       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01261       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01262     }
01263   gtk_widget_set_sensitive
01264     (GTK_WIDGET (window->spin_minabs),
01265      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01266   gtk_widget_set_sensitive
01267     (GTK_WIDGET (window->spin_maxabs),
01268      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01269   if (window_get_norm () == ERROR_NORM_P)
01270     {
01271       gtk_widget_show (GTK_WIDGET (window->label_p));
01272       gtk_widget_show (GTK_WIDGET (window->spin_p));
01273     }
01274 #if DEBUG_INTERFACE
01275   fprintf (stderr, "window_update: end\n");
01276 #endif
01277 }
```

Here is the call graph for this function:



4.13.2.32 window_update_direction()

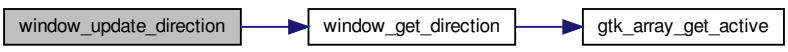
```
void window_update_direction ( )
```

Function to update direction search method widgets view in the main window.

Definition at line 1094 of file interface.c.

```
01095 {
01096 #if DEBUG_INTERFACE
01097   fprintf (stderr, "window_update_direction: start\n");
01098 #endif
01099   gtk_widget_show (GTK_WIDGET (window->check_direction));
01100   if (gtk_toggle_button_get_active
01101       (GTK_TOGGLE_BUTTON (window->check_direction)))
01102     {
01103       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01104       gtk_widget_show (GTK_WIDGET (window->label_step));
01105       gtk_widget_show (GTK_WIDGET (window->spin_step));
01106     }
01107   switch (window_get_direction ())
01108     {
01109     case DIRECTION_METHOD_COORDINATES:
01110       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01111       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01112       break;
01113     default:
01114       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01115       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01116     }
01117 #if DEBUG_INTERFACE
01118   fprintf (stderr, "window_update_direction: end\n");
01119 #endif
01120 }
```

Here is the call graph for this function:



4.13.2.33 window_update_variable()

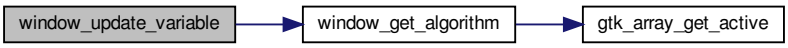
```
void window_update_variable ( )
```

Function to update the variable data in the main window.

Definition at line 1826 of file interface.c.

```
01827 {
01828   int i;
01829 #if DEBUG_INTERFACE
01830   fprintf (stderr, "window_update_variable: start\n");
01831 #endif
01832   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01833   if (i < 0)
01834     i = 0;
01835   switch (window_get_algorithm ())
01836     {
01837     case ALGORITHM_SWEEP:
01838     case ALGORITHM_ORTHOGONAL:
01839       input->variable[i].nsweeps
01840         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01841 #if DEBUG_INTERFACE
01842       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01843                input->variable[i].nsweeps);
01844 #endif
01845       break;
01846     case ALGORITHM_GENETIC:
01847       input->variable[i].nbits
01848         = gtk_spin_button_get_value_as_int (window->spin_bits);
01849 #if DEBUG_INTERFACE
01850       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01851                input->variable[i].nbits);
01852 #endif
01853     }
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_update_variable: end\n");
01856 #endif
01857 }
```

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 1477 of file interface.c.

```
01478 {
01479   unsigned int i;
01480 #if DEBUG_INTERFACE
01481   fprintf (stderr, "window_weight_experiment: start\n");
01482 #endif
01483   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01484   input->experiment[i].weight = gtk_spin_button_get_value (
      window->spin_weight);
01485 #if DEBUG_INTERFACE
01486   fprintf (stderr, "window_weight_experiment: end\n");
01487 #endif
01488 }
```

4.14 interface.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
00049 {
00050   GtkDialog *dialog;
00051   GtkGrid *grid;
00052   GtkLabel *label_seed;
00054   GtkSpinButton *spin_seed;
00056   GtkLabel *label_threads;
00057   GtkSpinButton *spin_threads;
00058   GtkLabel *label_direction;
00059   GtkSpinButton *spin_direction;
00061 } Options;
00062
00067 typedef struct
00068 {
00069   GtkDialog *dialog;
00070   GtkLabel *label;
00071   GtkSpinner *spinner;
00072   GtkGrid *grid;
00073 } Running;
00074
00079 typedef struct
00080 {
00081   GtkWindow *window;
00082   GtkGrid *grid;
00083   GtkToolbar *bar_buttons;
00084   GtkToolButton *button_open;
00085   GtkToolButton *button_save;
00086   GtkToolButton *button_run;
00087   GtkToolButton *button_options;
00088   GtkToolButton *button_help;
00089   GtkToolButton *button_about;
00090   GtkToolButton *button_exit;
00091   GtkGrid *grid_files;
00092   GtkLabel *label_simulator;
00093   GtkFileChooserButton *button_simulator;
00095   GtkCheckButton *check_evaluator;
00096   GtkFileChooserButton *button_evaluator;
00098   GtkLabel *label_result;
00099   GtkEntry *entry_result;
00100   GtkLabel *label_variables;
00101   GtkEntry *entry_variables;
00102   GtkFrame *frame_norm;
00103   GtkGrid *grid_norm;
00104   GtkRadioButton *button_norm[NNORMS];
00106   GtkLabel *label_p;
00107   GtkSpinButton *spin_p;
00108   GtkScrolledWindow *scrolled_p;
00110   GtkFrame *frame_algorithm;
```

```
00111   GtkGrid *grid_algorithm;
00112   GtkRadioButton *button_algorithm[NALGORITHMS];
00114   GtkLabel *label_simulations;
00115   GtkSpinButton *spin_simulations;
00117   GtkLabel *label_iterations;
00118   GtkSpinButton *spin_iterations;
00120   GtkLabel *label_tolerance;
00121   GtkSpinButton *spin_tolerance;
00122   GtkLabel *label_bests;
00123   GtkSpinButton *spin_bests;
00124   GtkLabel *label_population;
00125   GtkSpinButton *spin_population;
00127   GtkLabel *label_generations;
00128   GtkSpinButton *spin_generations;
00130   GtkLabel *label_mutation;
00131   GtkSpinButton *spin_mutation;
00132   GtkLabel *label_reproduction;
00133   GtkSpinButton *spin_reproduction;
00135   GtkLabel *label_adaptation;
00136   GtkSpinButton *spin_adaptation;
00138   GtkCheckButton *check_direction;
00140   GtkGrid *grid_direction;
00142   GtkRadioButton *button_direction[NDIRECTIONS];
00144   GtkLabel *label_steps;
00145   GtkSpinButton *spin_steps;
00146   GtkLabel *label_estimates;
00147   GtkSpinButton *spin_estimates;
00149   GtkLabel *label_relaxation;
00151   GtkSpinButton *spin_relaxation;
00153   GtkLabel *label_threshold;
00154   GtkSpinButton *spin_threshold;
00155   GtkScrolledWindow *scrolled_threshold;
00157   GtkFrame *frame_variable;
00158   GtkGrid *grid_variable;
00159   GtkComboBoxText *combo_variable;
00161   GtkButton *button_add_variable;
00162   GtkButton *button_remove_variable;
00163   GtkLabel *label_variable;
00164   GtkEntry *entry_variable;
00165   GtkLabel *label_min;
00166   GtkSpinButton *spin_min;
00167   GtkScrolledWindow *scrolled_min;
00168   GtkLabel *label_max;
00169   GtkSpinButton *spin_max;
00170   GtkScrolledWindow *scrolled_max;
00171   GtkCheckButton *check_minabs;
00172   GtkSpinButton *spin_minabs;
00173   GtkScrolledWindow *scrolled_minabs;
00174   GtkCheckButton *check_maxabs;
00175   GtkSpinButton *spin_maxabs;
00176   GtkScrolledWindow *scrolled_maxabs;
00177   GtkLabel *label_precision;
00178   GtkSpinButton *spin_precision;
00179   GtkLabel *label_sweeps;
00180   GtkSpinButton *spin_sweeps;
00181   GtkLabel *label_bits;
00182   GtkSpinButton *spin_bits;
00183   GtkLabel *label_step;
00184   GtkSpinButton *spin_step;
00185   GtkScrolledWindow *scrolled_step;
00186   GtkFrame *frame_experiment;
00187   GtkGrid *grid_experiment;
00188   GtkComboBoxText *combo_experiment;
00189   GtkButton *button_add_experiment;
00190   GtkButton *button_remove_experiment;
00191   GtkLabel *label_experiment;
00192   GtkFileChooserButton *button_experiment;
00194   GtkLabel *label_weight;
00195   GtkSpinButton *spin_weight;
00196   GtkCheckButton *check_template[MAX_NINPUTS];
00198   GtkFileChooserButton *button_template[MAX_NINPUTS];
00200   GdkPixbuf *logo;
00201   Experiment *experiment;
00202   Variable *variable;
00203   char *application_directory;
00204   gulong id_experiment;
00205   gulong id_experiment_name;
00206   gulong id_variable;
00207   gulong id_variable_label;
00208   gulong id_template[MAX_NINPUTS];
00210   gulong id_input[MAX_NINPUTS];
00212   unsigned int nexperiments;
00213   unsigned int nvariables;
00214 } Window;
00215
00216 // Global variables
00217 extern const char *logo[];
```

```
00218 extern Options options[1];
00219 extern Running running[1];
00220 extern Window window[1];
00221
00222 // Inline functions
00223 #if GTK_MINOR_VERSION < 10
00224 static inline GtkButton *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227   GtkButton *button;
00228   GtkImage *image;
00229   button = (GtkButton *) gtk_button_new ();
00230   image = (GtkImage *) gtk_image_new_from_icon_name (name, size);
00231   gtk_button_set_image (button, GTK_WIDGET (image));
00232   return button;
00233 }
00234 #endif
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);
00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_direction ();
00243 unsigned int window_get_norm ();
00244 void window_save_direction ();
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_direction ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new (GtkApplication * application);
00271
00272 #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

Definition in file main.c.

4.16 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
```

```
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #if HAVE_GTK
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #endif
00060 #include "genetic/genetic.h"
00061 #include "utils.h"
00062 #include "experiment.h"
00063 #include "variable.h"
00064 #include "input.h"
00065 #include "optimize.h"
00066 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069 #include "mpcotool.h"
00070
00071 int
00072 main (int argn, char **argc)
00073 {
00074 #if HAVE_GTK
00075   show_pending = process_pending;
00076 #endif
00077   return mpcotool (argn, argc);
00078 }
```

4.17 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_direction (unsigned int simulation, double value)
- void optimize_direction_sequential (unsigned int simulation)
- void * optimize_direction_thread (ParallelData *data)
- double optimize_estimate_direction_random (unsigned int variable, unsigned int estimate __attribute__↩ ((unused)))
- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)
- void optimize_step_direction (unsigned int simulation)
- void optimize_direction ()
- 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_direction

  Number of threads for the direction search method.
- void(∗ optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(∗ optimize_estimate_direction)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the direction.
- double(∗ optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Optimize optimize [1]

  Optimization data.

4.17.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file optimize.c.

4.17.2 Function Documentation

4.17.2.1 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.

```
00446 {
```

```
00447   unsigned int i, j;
00448   double e;
00449 #if DEBUG_OPTIMIZE
00450   fprintf (stderr, "optimize_best: start\n");
00451   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00452            optimize->nsaveds, optimize->nbest);
00453 #endif
00454   if (optimize->nsaveds < optimize->nbest
00455       || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457       if (optimize->nsaveds < optimize->nbest)
00458         ++optimize->nsaveds;
00459       optimize->error_best[optimize->nsaveds - 1] = value;
00460       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461       for (i = optimize->nsaveds; --i;)
00462         {
00463           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00464             {
00465               j = optimize->simulation_best[i];
00466               e = optimize->error_best[i];
00467               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00468               optimize->error_best[i] = optimize->
      error_best[i - 1];
00469               optimize->simulation_best[i - 1] = j;
00470               optimize->error_best[i - 1] = e;
00471             }
00472           else
00473             break;
00474         }
00475     }
00476 #if DEBUG_OPTIMIZE
00477   fprintf (stderr, "optimize_best: end\n");
00478 #endif
00479 }
```

4.17.2.2 optimize_best_direction()

```
void optimize_best_direction (
             unsigned int simulation,
             double value )
```

Function to save the best simulation in a direction search method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 806 of file optimize.c.

```
00810 {
00811 #if DEBUG_OPTIMIZE
00812   fprintf (stderr, "optimize_best_direction: start\n");
00813   fprintf (stderr,
00814            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00815            simulation, value, optimize->error_best[0]);
00816 #endif
00817   if (value < optimize->error_best[0])
00818     {
00819       optimize->error_best[0] = value;
00820       optimize->simulation_best[0] = simulation;
00821 #if DEBUG_OPTIMIZE
00822       fprintf (stderr,
00823                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00824                simulation, value);
00825 #endif
00826     }
```

```
00827 #if DEBUG_OPTIMIZE
00828   fprintf (stderr, "optimize_best_direction: end\n");
00829 #endif
00830 }
```

4.17.2.3 optimize_direction()

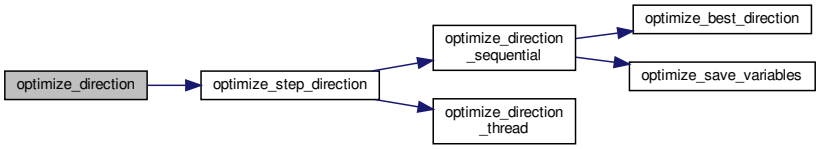
```
void optimize_direction ( )
```

Function to optimize with a direction search method.

Definition at line 1036 of file optimize.c.

```
01037 {
01038   unsigned int i, j, k, b, s, adjust;
01039 #if DEBUG_OPTIMIZE
01040   fprintf (stderr, "optimize_direction: start\n");
01041 #endif
01042   for (i = 0; i < optimize->nvariables; ++i)
01043     optimize->direction[i] = 0.;
01044   b = optimize->simulation_best[0] * optimize->
      nvariables;
01045   s = optimize->nsimulations;
01046   adjust = 1;
01047   for (i = 0; i < optimize->nsteps; ++i, s += optimize->
      nestimates, b = k)
01048     {
01049 #if DEBUG_OPTIMIZE
01050       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01051                i, optimize->simulation_best[0]);
01052 #endif
01053       optimize_step_direction (s);
01054       k = optimize->simulation_best[0] * optimize->
      nvariables;
01055 #if DEBUG_OPTIMIZE
01056       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01057                i, optimize->simulation_best[0]);
01058 #endif
01059       if (k == b)
01060         {
01061           if (adjust)
01062             for (j = 0; j < optimize->nvariables; ++j)
01063               optimize->step[j] *= 0.5;
01064           for (j = 0; j < optimize->nvariables; ++j)
01065             optimize->direction[j] = 0.;
01066           adjust = 1;
01067         }
01068       else
01069         {
01070           for (j = 0; j < optimize->nvariables; ++j)
01071             {
01072 #if DEBUG_OPTIMIZE
01073               fprintf (stderr,
01074                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01075                        j, optimize->value[k + j], j, optimize->
      value[b + j]);
01076 #endif
01077               optimize->direction[j]
01078                 = (1. - optimize->relaxation) * optimize->
      direction[j]
01079                 + optimize->relaxation
01080                 * (optimize->value[k + j] - optimize->value[b + j]);
01081 #if DEBUG_OPTIMIZE
01082               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01083                        j, optimize->direction[j]);
01084 #endif
01085             }
01086           adjust = 0;
01087         }
01088     }
01089 #if DEBUG_OPTIMIZE
01090   fprintf (stderr, "optimize_direction: end\n");
01091 #endif
01092 }
```

Here is the call graph for this function:



4.17.2.4 optimize_direction_sequential()

```
void optimize_direction_sequential (
             unsigned int simulation )
```

Function to estimate the direction search sequentially.

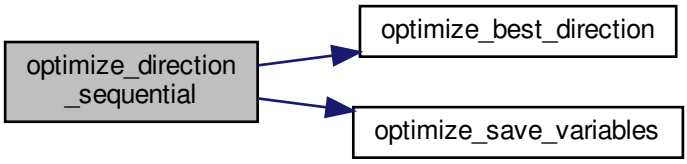
Parameters

simulation	Simulation number.

Definition at line 836 of file optimize.c.

```
00837 {
00838   unsigned int i, j;
00839   double e;
00840 #if DEBUG_OPTIMIZE
00841   fprintf (stderr, "optimize_direction_sequential: start\n");
00842   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00843            "nend_direction=%u\n",
00844            optimize->nstart_direction, optimize->
      nend_direction);
00845 #endif
00846   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00847     {
00848       j = simulation + i;
00849       e = optimize_norm (j);
00850       optimize_best_direction (j, e);
00851       optimize_save_variables (j, e);
00852       if (e < optimize->threshold)
00853         {
00854           optimize->stop = 1;
00855           break;
00856         }
00857 #if DEBUG_OPTIMIZE
00858       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00859 #endif
00860     }
00861 #if DEBUG_OPTIMIZE
00862   fprintf (stderr, "optimize_direction_sequential: end\n");
00863 #endif
00864 }
```

Here is the call graph for this function:



4.17.2.5 optimize_direction_thread()

```
void* optimize_direction_thread (
             ParallelData * data )
```

Function to estimate the direction search on a thread.

Returns

NULL

Parameters

data	Function data.

Definition at line 872 of file optimize.c.

```
00873 {
00874   unsigned int i, thread;
00875   double e;
00876 #if DEBUG_OPTIMIZE
00877   fprintf (stderr, "optimize_direction_thread: start\n");
00878 #endif
00879   thread = data->thread;
00880 #if DEBUG_OPTIMIZE
00881   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00882            thread,
00883            optimize->thread_direction[thread],
00884            optimize->thread_direction[thread + 1]);
00885 #endif
00886   for (i = optimize->thread_direction[thread];
00887        i < optimize->thread_direction[thread + 1]; ++i)
00888     {
00889       e = optimize_norm (i);
00890       g_mutex_lock (mutex);
00891       optimize_best_direction (i, e);
00892       optimize_save_variables (i, e);
00893       if (e < optimize->threshold)
00894         optimize->stop = 1;
00895       g_mutex_unlock (mutex);
00896       if (optimize->stop)
00897         break;
00898 #if DEBUG_OPTIMIZE
00899       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
```

```
00900 #endif
00901     }
00902 #if DEBUG_OPTIMIZE
00903   fprintf (stderr, "optimize_direction_thread: end\n");
00904 #endif
00905   g_thread_exit (NULL);
00906   return NULL;
00907 }
```

4.17.2.6 optimize_estimate_direction_coordinates()

```
double optimize_estimate_direction_coordinates (
             unsigned int variable,
             unsigned int estimate )
```

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 937 of file optimize.c.

```
00941 {
00942   double x;
00943 #if DEBUG_OPTIMIZE
00944   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945 #endif
00946   x = optimize->direction[variable];
00947   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949       if (estimate & 1)
00950         x += optimize->step[variable];
00951       else
00952         x -= optimize->step[variable];
00953     }
00954 #if DEBUG_OPTIMIZE
00955   fprintf (stderr,
00956            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00957            variable, x);
00958   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959 #endif
00960   return x;
00961 }
```

4.17.2.7 optimize_estimate_direction_random()

```
double optimize_estimate_direction_random (
             unsigned int variable,
             unsigned int estimate  __attribute__(unused) )
```

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
__attribute__	Estimate number.

Definition at line 913 of file optimize.c.

```
00918 {
00919   double x;
00920 #if DEBUG_OPTIMIZE
00921   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00922 #endif
00923   x = optimize->direction[variable]
00924     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00925 #if DEBUG_OPTIMIZE
00926   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00927            variable, x);
00928   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00929 #endif
00930   return x;
00931 }
```

4.17.2.8 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the Optimize struct.

Definition at line 1393 of file optimize.c.

```
01394 {
01395   unsigned int i, j;
01396 #if DEBUG_OPTIMIZE
01397   fprintf (stderr, "optimize_free: start\n");
01398 #endif
01399   for (j = 0; j < optimize->ninputs; ++j)
01400     {
01401       for (i = 0; i < optimize->nexperiments; ++i)
01402         g_mapped_file_unref (optimize->file[j][i]);
01403       g_free (optimize->file[j]);
01404     }
01405   g_free (optimize->error_old);
01406   g_free (optimize->value_old);
01407   g_free (optimize->value);
01408   g_free (optimize->genetic_variable);
01409 #if DEBUG_OPTIMIZE
01410   fprintf (stderr, "optimize_free: end\n");
01411 #endif
01412 }
```

4.17.2.9 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1133 of file optimize.c.

```
01134 {
01135   double *best_variable = NULL;
01136   char *best_genome = NULL;
01137   double best_objective = 0.;
01138 #if DEBUG_OPTIMIZE
01139   fprintf (stderr, "optimize_genetic: start\n");
01140   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01141            nthreads);
01142   fprintf (stderr,
01143            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01144            optimize->nvariables, optimize->
      nsimulations, optimize->niterations);
01145   fprintf (stderr,
01146            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01147            optimize->mutation_ratio, optimize->
      reproduction_ratio,
01148            optimize->adaptation_ratio);
01149 #endif
01150   genetic_algorithm_default (optimize->nvariables,
01151                              optimize->genetic_variable,
01152                              optimize->nsimulations,
01153                              optimize->niterations,
01154                              optimize->mutation_ratio,
01155                              optimize->reproduction_ratio,
01156                              optimize->adaptation_ratio,
01157                              optimize->seed,
01158                              optimize->threshold,
01159                              &optimize_genetic_objective,
01160                              &best_genome, &best_variable, &best_objective);
01161 #if DEBUG_OPTIMIZE
01162   fprintf (stderr, "optimize_genetic: the best\n");
01163 #endif
01164   optimize->error_old = (double *) g_malloc (sizeof (double));
01165   optimize->value_old
01166     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01167   optimize->error_old[0] = best_objective;
01168   memcpy (optimize->value_old, best_variable,
01169           optimize->nvariables * sizeof (double));
01170   g_free (best_genome);
01171   g_free (best_variable);
01172   optimize_print ();
01173 #if DEBUG_OPTIMIZE
01174   fprintf (stderr, "optimize_genetic: end\n");
01175 #endif
01176 }
```

4.17.2.10 optimize_genetic_objective()

```
double optimize_genetic_objective (
               Entity * entity )
```

Function to calculate the objective function of an entity.

Returns

objective function value.

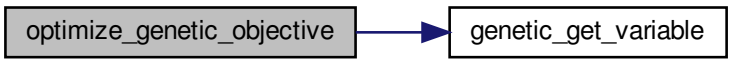
Parameters

entity	entity data.

Definition at line 1100 of file optimize.c.

```
01101 {
01102   unsigned int j;
01103   double objective;
01104   char buffer[64];
01105 #if DEBUG_OPTIMIZE
01106   fprintf (stderr, "optimize_genetic_objective: start\n");
01107 #endif
01108   for (j = 0; j < optimize->nvariables; ++j)
01109     {
01110       optimize->value[entity->id * optimize->nvariables + j]
01111         = genetic_get_variable (entity, optimize->genetic_variable + j);
01112     }
01113   objective = optimize_norm (entity->id);
01114   g_mutex_lock (mutex);
01115   for (j = 0; j < optimize->nvariables; ++j)
01116     {
01117       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01118       fprintf (optimize->file_variables, buffer,
01119                genetic_get_variable (entity, optimize->genetic_variable + j));
01120     }
01121   fprintf (optimize->file_variables, "%.14le\n", objective);
01122   g_mutex_unlock (mutex);
01123 #if DEBUG_OPTIMIZE
01124   fprintf (stderr, "optimize_genetic_objective: end\n");
01125 #endif
01126   return objective;
01127 }
```

Here is the call graph for this function:



4.17.2.11 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.

```
00096 {
00097   char buffer[32], value[32];
00098   GRegex *regex;
00099   FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101   gsize length;
00102   unsigned int i;
00103
00104 #if DEBUG_OPTIMIZE
00105   fprintf (stderr, "optimize_input: start\n");
00106 #endif
00107
00108   // Checking the file
00109   if (!stencil)
00110     goto optimize_input_end;
00111
00112   // Opening stencil
00113   content = g_mapped_file_get_contents (stencil);
00114   length = g_mapped_file_get_length (stencil);
00115 #if DEBUG_OPTIMIZE
00116   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117 #endif
00118   file = g_fopen (input, "w");
00119
00120   // Parsing stencil
00121   for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123 #if DEBUG_OPTIMIZE
00124       fprintf (stderr, "optimize_input: variable=%u\n", i);
00125 #endif
00126       snprintf (buffer, 32, "@variable%u@", i + 1);
00127       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00128                            NULL);
00129       if (i == 0)
00130         {
00131           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                              optimize->label[i],
00133                                              (GRegexMatchFlags) 0, NULL);
00134 #if DEBUG_OPTIMIZE
00135           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136 #endif
00137         }
00138       else
00139         {
00140           length = strlen (buffer3);
00141           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                              optimize->label[i],
00143                                              (GRegexMatchFlags) 0, NULL);
00144           g_free (buffer3);
00145         }
00146       g_regex_unref (regex);
00147       length = strlen (buffer2);
00148       snprintf (buffer, 32, "@value%u@", i + 1);
00149       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                            NULL);
00151       snprintf (value, 32, format[optimize->precision[i]],
00152                 optimize->value[simulation * optimize->
      nvariables + i]);
00153
00154 #if DEBUG_OPTIMIZE
00155       fprintf (stderr, "optimize_input: value=%s\n", value);
00156 #endif
00157       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                          (GRegexMatchFlags) 0, NULL);
00159       g_free (buffer2);
00160       g_regex_unref (regex);
00161     }
00162
00163   // Saving input file
00164   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165   g_free (buffer3);
00166   fclose (file);
00167
00168 optimize_input_end:
00169 #if DEBUG_OPTIMIZE
00170   fprintf (stderr, "optimize_input: end\n");
00171 #endif
00172   return;
00173 }
```

4.17.2.12 optimize_iterate()

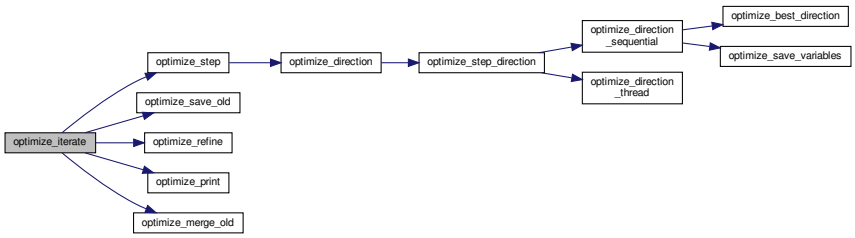
```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1363 of file optimize.c.

```
01364 {
01365   unsigned int i;
01366 #if DEBUG_OPTIMIZE
01367   fprintf (stderr, "optimize_iterate: start\n");
01368 #endif
01369   optimize->error_old = (double *) g_malloc (optimize->
      nbest * sizeof (double));
01370   optimize->value_old =
01371     (double *) g_malloc (optimize->nbest * optimize->
      nvariables *
01372                          sizeof (double));
01373   optimize_step ();
01374   optimize_save_old ();
01375   optimize_refine ();
01376   optimize_print ();
01377   for (i = 1; i < optimize->niterations && !optimize->
      stop; ++i)
01378     {
01379       optimize_step ();
01380       optimize_merge_old ();
01381       optimize_refine ();
01382       optimize_print ();
01383     }
01384 #if DEBUG_OPTIMIZE
01385   fprintf (stderr, "optimize_iterate: end\n");
01386 #endif
01387 }
```

Here is the call graph for this function:



4.17.2.13 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.

```
00562 {
00563   unsigned int i, j, k, s[optimize->nbest];
00564   double e[optimize->nbest];
00565 #if DEBUG_OPTIMIZE
00566   fprintf (stderr, "optimize_merge: start\n");
00567 #endif
00568   i = j = k = 0;
00569   do
00570     {
00571       if (i == optimize->nsaveds)
00572         {
00573           s[k] = simulation_best[j];
00574           e[k] = error_best[j];
00575           ++j;
00576           ++k;
00577           if (j == nsaveds)
00578             break;
00579         }
00580       else if (j == nsaveds)
00581         {
00582           s[k] = optimize->simulation_best[i];
00583           e[k] = optimize->error_best[i];
00584           ++i;
00585           ++k;
00586           if (i == optimize->nsaveds)
00587             break;
00588         }
00589       else if (optimize->error_best[i] > error_best[j])
00590         {
00591           s[k] = simulation_best[j];
00592           e[k] = error_best[j];
00593           ++j;
00594           ++k;
00595         }
00596       else
00597         {
00598           s[k] = optimize->simulation_best[i];
00599           e[k] = optimize->error_best[i];
00600           ++i;
00601           ++k;
00602         }
00603     }
00604   while (k < optimize->nbest);
00605   optimize->nsaveds = k;
00606   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00607   memcpy (optimize->error_best, e, k * sizeof (double));
00608 #if DEBUG_OPTIMIZE
00609   fprintf (stderr, "optimize_merge: end\n");
00610 #endif
00611 }
```

4.17.2.14 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 1214 of file optimize.c.

```
01215 {
01216   unsigned int i, j, k;
01217   double v[optimize->nbest * optimize->nvariables], e[
      optimize->nbest],
01218     *enew, *eold;
01219 #if DEBUG_OPTIMIZE
01220   fprintf (stderr, "optimize_merge_old: start\n");
01221 #endif
01222   enew = optimize->error_best;
01223   eold = optimize->error_old;
01224   i = j = k = 0;
01225   do
01226     {
01227       if (*enew < *eold)
01228         {
01229           memcpy (v + k * optimize->nvariables,
01230                   optimize->value
01231                   + optimize->simulation_best[i] *
      optimize->nvariables,
01232                   optimize->nvariables * sizeof (double));
01233           e[k] = *enew;
01234           ++k;
01235           ++enew;
01236           ++i;
01237         }
01238       else
01239         {
01240           memcpy (v + k * optimize->nvariables,
01241                   optimize->value_old + j * optimize->
      nvariables,
01242                   optimize->nvariables * sizeof (double));
01243           e[k] = *eold;
01244           ++k;
01245           ++eold;
01246           ++j;
01247         }
01248     }
01249   while (k < optimize->nbest);
01250   memcpy (optimize->value_old, v, k * optimize->
      nvariables * sizeof (double));
01251   memcpy (optimize->error_old, e, k * sizeof (double));
01252 #if DEBUG_OPTIMIZE
01253   fprintf (stderr, "optimize_merge_old: end\n");
01254 #endif
01255 }
```

4.17.2.15 optimize_MonteCarlo()

```
void optimize_MonteCarlo ( )
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file optimize.c.

```
00716 {
00717   unsigned int i, j;
00718   GThread *thread[nthreads];
00719   ParallelData data[nthreads];
00720 #if DEBUG_OPTIMIZE
00721   fprintf (stderr, "optimize_MonteCarlo: start\n");
00722 #endif
00723   for (i = 0; i < optimize->nsimulations; ++i)
00724     for (j = 0; j < optimize->nvariables; ++j)
00725       optimize->value[i * optimize->nvariables + j]
00726         = optimize->rangemin[j] + gsl_rng_uniform (optimize->
      rng)
00727         * (optimize->rangemax[j] - optimize->rangemin[j]);
00728   optimize->nsaveds = 0;
00729   if (nthreads <= 1)
00730     optimize_sequential ();
00731   else
00732     {
00733       for (i = 0; i < nthreads; ++i)
00734         {
00735           data[i].thread = i;
```

```
00736           thread[i]
00737             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00738         }
00739       for (i = 0; i < nthreads; ++i)
00740         g_thread_join (thread[i]);
00741     }
00742 #if HAVE_MPI
00743   // Communicating tasks results
00744   optimize_synchronise ();
00745 #endif
00746 #if DEBUG_OPTIMIZE
00747   fprintf (stderr, "optimize_MonteCarlo: end\n");
00748 #endif
00749 }
```

4.17.2.16 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
             unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

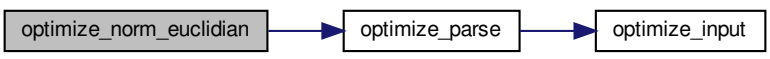
Parameters

simulation	simulation number.

Definition at line 292 of file optimize.c.

```
00293 {
00294   double e, ei;
00295   unsigned int i;
00296 #if DEBUG_OPTIMIZE
00297   fprintf (stderr, "optimize_norm_euclidian: start\n");
00298 #endif
00299   e = 0.;
00300   for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302       ei = optimize_parse (simulation, i);
00303       e += ei * ei;
00304     }
00305   e = sqrt (e);
00306 #if DEBUG_OPTIMIZE
00307   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308   fprintf (stderr, "optimize_norm_euclidian: end\n");
00309 #endif
00310   return e;
00311 }
```

Here is the call graph for this function:



4.17.2.17 optimize_norm_maximum()

```
double optimize_norm_maximum (
             unsigned int simulation )
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

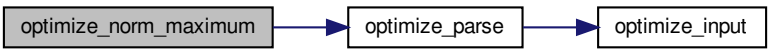
Parameters

simulation	simulation number.

Definition at line 319 of file optimize.c.

```
00320 {
00321   double e, ei;
00322   unsigned int i;
00323 #if DEBUG_OPTIMIZE
00324   fprintf (stderr, "optimize_norm_maximum: start\n");
00325 #endif
00326   e = 0.;
00327   for (i = 0; i < optimize->nexperiments; ++i)
00328     {
00329       ei = fabs (optimize_parse (simulation, i));
00330       e = fmax (e, ei);
00331     }
00332 #if DEBUG_OPTIMIZE
00333   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334   fprintf (stderr, "optimize_norm_maximum: end\n");
00335 #endif
00336   return e;
00337 }
```

Here is the call graph for this function:



4.17.2.18 optimize_norm_p()

```
double optimize_norm_p (
             unsigned int simulation )
```

Function to calculate the P error norm.

Returns

P error norm.

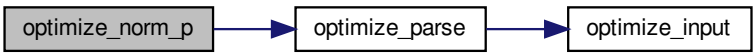
Parameters

simulation	simulation number.

Definition at line 345 of file optimize.c.

```
00346 {
00347   double e, ei;
00348   unsigned int i;
00349 #if DEBUG_OPTIMIZE
00350   fprintf (stderr, "optimize_norm_p: start\n");
00351 #endif
00352   e = 0.;
00353   for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355       ei = fabs (optimize_parse (simulation, i));
00356       e += pow (ei, optimize->p);
00357     }
00358   e = pow (e, 1. / optimize->p);
00359 #if DEBUG_OPTIMIZE
00360   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361   fprintf (stderr, "optimize_norm_p: end\n");
00362 #endif
00363   return e;
00364 }
```

Here is the call graph for this function:



4.17.2.19 optimize_norm_taxicab()

```
double optimize_norm_taxicab (
             unsigned int simulation )
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

Parameters

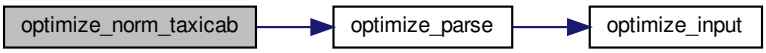
simulation	simulation number.

Definition at line 372 of file optimize.c.

```
00373 {
```

```
00374   double e;
00375   unsigned int i;
00376 #if DEBUG_OPTIMIZE
00377   fprintf (stderr, "optimize_norm_taxicab: start\n");
00378 #endif
00379   e = 0.;
00380   for (i = 0; i < optimize->nexperiments; ++i)
00381     e += fabs (optimize_parse (simulation, i));
00382 #if DEBUG_OPTIMIZE
00383   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384   fprintf (stderr, "optimize_norm_taxicab: end\n");
00385 #endif
00386   return e;
00387 }
```

Here is the call graph for this function:



4.17.2.20 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1418 of file optimize.c.

```
01419 {
01420   GTimeZone *tz;
01421   GDateTime *t0, *t;
01422   unsigned int i, j;
01423
01424 #if DEBUG_OPTIMIZE
01425   char *buffer;
01426   fprintf (stderr, "optimize_open: start\n");
01427 #endif
01428
01429   // Getting initial time
01430 #if DEBUG_OPTIMIZE
01431   fprintf (stderr, "optimize_open: getting initial time\n");
01432 #endif
01433   tz = g_time_zone_new_utc ();
01434   t0 = g_date_time_new_now (tz);
01435
01436   // Obtaining and initing the pseudo-random numbers generator seed
01437 #if DEBUG_OPTIMIZE
01438   fprintf (stderr, "optimize_open: getting initial seed\n");
01439 #endif
01440   if (optimize->seed == DEFAULT_RANDOM_SEED)
01441     optimize->seed = input->seed;
01442   gsl_rng_set (optimize->rng, optimize->seed);
01443
01444   // Replacing the working directory
01445 #if DEBUG_OPTIMIZE
01446   fprintf (stderr, "optimize_open: replacing the working directory\n");
01447 #endif
01448   g_chdir (input->directory);
01449
01450   // Getting results file names
01451   optimize->result = input->result;
01452   optimize->variables = input->variables;
```

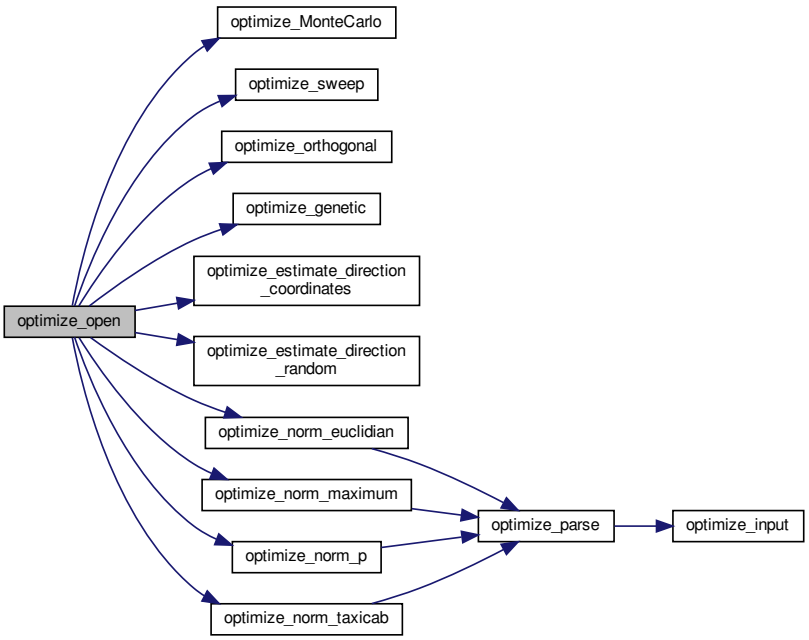
```
01453
01454   // Obtaining the simulator file
01455   optimize->simulator = input->simulator;
01456
01457   // Obtaining the evaluator file
01458   optimize->evaluator = input->evaluator;
01459
01460   // Reading the algorithm
01461   optimize->algorithm = input->algorithm;
01462   switch (optimize->algorithm)
01463     {
01464     case ALGORITHM_MONTE_CARLO:
01465       optimize_algorithm = optimize_MonteCarlo;
01466       break;
01467     case ALGORITHM_SWEEP:
01468       optimize_algorithm = optimize_sweep;
01469       break;
01470     case ALGORITHM_ORTHOGONAL:
01471       optimize_algorithm = optimize_orthogonal;
01472       break;
01473     default:
01474       optimize_algorithm = optimize_genetic;
01475       optimize->mutation_ratio = input->
    mutation_ratio;
01476       optimize->reproduction_ratio = input->
    reproduction_ratio;
01477       optimize->adaptation_ratio = input->
    adaptation_ratio;
01478     }
01479   optimize->nvariables = input->nvariables;
01480   optimize->nsimulations = input->nsimulations;
01481   optimize->niterations = input->niterations;
01482   optimize->nbest = input->nbest;
01483   optimize->tolerance = input->tolerance;
01484   optimize->nsteps = input->nsteps;
01485   optimize->nestimates = 0;
01486   optimize->threshold = input->threshold;
01487   optimize->stop = 0;
01488   if (input->nsteps)
01489     {
01490       optimize->relaxation = input->relaxation;
01491       switch (input->direction)
01492         {
01493         case DIRECTION_METHOD_COORDINATES:
01494           optimize->nestimates = 2 * optimize->
    nvariables;
01495           optimize_estimate_direction =
    optimize_estimate_direction_coordinates;
01496           break;
01497         default:
01498           optimize->nestimates = input->nestimates;
01499           optimize_estimate_direction =
    optimize_estimate_direction_random;
01500         }
01501     }
01502
01503 #if DEBUG_OPTIMIZE
01504   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01505 #endif
01506   optimize->simulation_best
01507     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01508   optimize->error_best = (double *) alloca (optimize->
    nbest * sizeof (double));
01509
01510   // Reading the experimental data
01511 #if DEBUG_OPTIMIZE
01512   buffer = g_get_current_dir ();
01513   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01514   g_free (buffer);
01515 #endif
01516   optimize->nexperiments = input->nexperiments;
01517   optimize->ninputs = input->experiment->ninputs;
01518   optimize->experiment
01519     = (char **) alloca (input->nexperiments * sizeof (char *));
01520   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double
    ));
01521   for (i = 0; i < input->experiment->ninputs; ++i)
01522     optimize->file[i] = (GMappedFile **)
01523       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01524   for (i = 0; i < input->nexperiments; ++i)
01525     {
01526 #if DEBUG_OPTIMIZE
01527       fprintf (stderr, "optimize_open: i=%u\n", i);
01528 #endif
01529       optimize->experiment[i] = input->experiment[i].
    name;
01530       optimize->weight[i] = input->experiment[i].
```

```
      weight;
01531 #if DEBUG_OPTIMIZE
01532       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01533                optimize->experiment[i], optimize->
      weight[i]);
01534 #endif
01535       for (j = 0; j < input->experiment->ninputs; ++j)
01536         {
01537 #if DEBUG_OPTIMIZE
01538           fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01539 #endif
01540           optimize->file[j][i]
01541             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01542         }
01543     }
01544
01545   // Reading the variables data
01546 #if DEBUG_OPTIMIZE
01547   fprintf (stderr, "optimize_open: reading variables\n");
01548 #endif
01549   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01550   j = input->nvariables * sizeof (double);
01551   optimize->rangemin = (double *) alloca (j);
01552   optimize->rangeminabs = (double *) alloca (j);
01553   optimize->rangemax = (double *) alloca (j);
01554   optimize->rangemaxabs = (double *) alloca (j);
01555   optimize->step = (double *) alloca (j);
01556   j = input->nvariables * sizeof (unsigned int);
01557   optimize->precision = (unsigned int *) alloca (j);
01558   optimize->nsweeps = (unsigned int *) alloca (j);
01559   optimize->nbits = (unsigned int *) alloca (j);
01560   for (i = 0; i < input->nvariables; ++i)
01561     {
01562       optimize->label[i] = input->variable[i].name;
01563       optimize->rangemin[i] = input->variable[i].
      rangemin;
01564       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01565       optimize->rangemax[i] = input->variable[i].
      rangemax;
01566       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01567       optimize->precision[i] = input->variable[i].
      precision;
01568       optimize->step[i] = input->variable[i].step;
01569       optimize->nsweeps[i] = input->variable[i].
      nsweeps;
01570       optimize->nbits[i] = input->variable[i].nbits;
01571     }
01572   if (input->algorithm == ALGORITHM_SWEEP
01573               || input->algorithm == ALGORITHM_ORTHOGONAL)
01574     {
01575       optimize->nsimulations = 1;
01576       for (i = 0; i < input->nvariables; ++i)
01577         {
01578           optimize->nsimulations *= optimize->
      nsweeps[i];
01579 #if DEBUG_OPTIMIZE
01580           fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01581                    optimize->nsweeps[i], optimize->
      nsimulations);
01582 #endif
01583         }
01584     }
01585   if (optimize->nsteps)
01586     optimize->direction
01587       = (double *) alloca (optimize->nvariables * sizeof (double));
01588
01589   // Setting error norm
01590   switch (input->norm)
01591     {
01592     case ERROR_NORM_EUCLIDIAN:
01593       optimize_norm = optimize_norm_euclidian;
01594       break;
01595     case ERROR_NORM_MAXIMUM:
01596       optimize_norm = optimize_norm_maximum;
01597       break;
01598     case ERROR_NORM_P:
01599       optimize_norm = optimize_norm_p;
01600       optimize->p = input->p;
01601       break;
01602     default:
01603       optimize_norm = optimize_norm_taxicab;
01604     }
01605
01606   // Allocating values
01607 #if DEBUG_OPTIMIZE
```

```
01608   fprintf (stderr, "optimize_open: allocating variables\n");
01609   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01610            optimize->nvariables, optimize->algorithm);
01611 #endif
01612   optimize->genetic_variable = NULL;
01613   if (optimize->algorithm == ALGORITHM_GENETIC)
01614     {
01615       optimize->genetic_variable = (GeneticVariable *)
01616         g_malloc (optimize->nvariables * sizeof (
      GeneticVariable));
01617       for (i = 0; i < optimize->nvariables; ++i)
01618         {
01619 #if DEBUG_OPTIMIZE
01620           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01621                    i, optimize->rangemin[i], optimize->
      rangemax[i],
01622                    optimize->nbits[i]);
01623 #endif
01624           optimize->genetic_variable[i].minimum =
      optimize->rangemin[i];
01625           optimize->genetic_variable[i].maximum =
      optimize->rangemax[i];
01626           optimize->genetic_variable[i].nbits = optimize->
      nbits[i];
01627         }
01628     }
01629 #if DEBUG_OPTIMIZE
01630   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01631            optimize->nvariables, optimize->
      nsimulations);
01632 #endif
01633   optimize->value = (double *)
01634     g_malloc ((optimize->nsimulations
01635                + optimize->nestimates * optimize->
      nsteps)
01636               * optimize->nvariables * sizeof (double));
01637
01638   // Calculating simulations to perform for each task
01639 #if HAVE_MPI
01640 #if DEBUG_OPTIMIZE
01641   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01642            optimize->mpi_rank, ntasks);
01643 #endif
01644   optimize->nstart = optimize->mpi_rank * optimize->
      nsimulations / ntasks;
01645   optimize->nend = (1 + optimize->mpi_rank) *
      optimize->nsimulations / ntasks;
01646   if (optimize->nsteps)
01647     {
01648       optimize->nstart_direction
01649         = optimize->mpi_rank * optimize->nestimates /
      ntasks;
01650       optimize->nend_direction
01651         = (1 + optimize->mpi_rank) * optimize->
      nestimates / ntasks;
01652     }
01653 #else
01654   optimize->nstart = 0;
01655   optimize->nend = optimize->nsimulations;
01656   if (optimize->nsteps)
01657     {
01658       optimize->nstart_direction = 0;
01659       optimize->nend_direction = optimize->
      nestimates;
01660     }
01661 #endif
01662 #if DEBUG_OPTIMIZE
01663   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
      nstart,
01664            optimize->nend);
01665 #endif
01666
01667   // Calculating simulations to perform for each thread
01668   optimize->thread
01669     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01670   for (i = 0; i <= nthreads; ++i)
01671     {
01672       optimize->thread[i] = optimize->nstart
01673         + i * (optimize->nend - optimize->nstart) / nthreads;
01674 #if DEBUG_OPTIMIZE
01675       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01676                optimize->thread[i]);
01677 #endif
01678     }
01679   if (optimize->nsteps)
01680     optimize->thread_direction = (unsigned int *)
01681       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
```

```
01682
01683   // Opening result files
01684   optimize->file_result = g_fopen (optimize->result, "w");
01685   optimize->file_variables = g_fopen (optimize->
      variables, "w");
01686
01687   // Performing the algorithm
01688   switch (optimize->algorithm)
01689     {
01690       // Genetic algorithm
01691     case ALGORITHM_GENETIC:
01692       optimize_genetic ();
01693       break;
01694
01695       // Iterative algorithm
01696     default:
01697       optimize_iterate ();
01698     }
01699
01700   // Getting calculation time
01701   t = g_date_time_new_now (tz);
01702   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01703   g_date_time_unref (t);
01704   g_date_time_unref (t0);
01705   g_time_zone_unref (tz);
01706   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->
      calculation_time);
01707   fprintf (optimize->file_result, "%s = %.6lg s\n",
01708            _("Calculation time"), optimize->calculation_time);
01709
01710   // Closing result files
01711   fclose (optimize->file_variables);
01712   fclose (optimize->file_result);
01713
01714 #if DEBUG_OPTIMIZE
01715   fprintf (stderr, "optimize_open: end\n");
01716 #endif
01717 }
```

Here is the call graph for this function:



4.17.2.21 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file optimize.c.

```
00756 {
00757   unsigned int i, j, k, l;
00758   double e;
00759   GThread *thread[nthreads];
00760   ParallelData data[nthreads];
00761 #if DEBUG_OPTIMIZE
00762   fprintf (stderr, "optimize_orthogonal: start\n");
00763 #endif
00764   for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766       k = i;
00767       for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769           l = k % optimize->nsweeps[j];
00770           k /= optimize->nsweeps[j];
00771           e = optimize->rangemin[j];
00772           if (optimize->nsweeps[j] > 1)
00773             e += (l + gsl_rng_uniform (optimize->rng))
00774                                 * (optimize->rangemax[j] - optimize->
      rangemin[j])
00775                 / optimize->nsweeps[j];
00776           optimize->value[i * optimize->nvariables + j] = e;
00777         }
00778     }
00779   optimize->nsaveds = 0;
00780   if (nthreads <= 1)
00781     optimize_sequential ();
00782   else
00783     {
00784       for (i = 0; i < nthreads; ++i)
00785         {
00786           data[i].thread = i;
00787           thread[i]
00788             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789         }
00790       for (i = 0; i < nthreads; ++i)
00791         g_thread_join (thread[i]);
00792     }
00793 #if HAVE_MPI
00794   // Communicating tasks results
00795   optimize_synchronise ();
00796 #endif
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_orthogonal: end\n");
00799 #endif
00800 }
```

4.17.2.22 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.

```
00184 {
00185   unsigned int i;
00186   double e;
00187   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188     *buffer3, *buffer4;
00189   FILE *file_result;
00190
00191 #if DEBUG_OPTIMIZE
00192   fprintf (stderr, "optimize_parse: start\n");
00193   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194            simulation, experiment);
00195 #endif
00196
00197   // Opening input files
00198   for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201 #if DEBUG_OPTIMIZE
00202       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203 #endif
00204       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00205     }
00206   for (; i < MAX_NINPUTS; ++i)
00207     strcpy (&input[i][0], "");
00208 #if DEBUG_OPTIMIZE
00209   fprintf (stderr, "optimize_parse: parsing end\n");
00210 #endif
00211
00212   // Performing the simulation
00213   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00214   buffer2 = g_path_get_dirname (optimize->simulator);
00215   buffer3 = g_path_get_basename (optimize->simulator);
00216   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00217   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00218             buffer4, input[0], input[1], input[2], input[3], input[4],
00219             input[5], input[6], input[7], output);
00220   g_free (buffer4);
00221   g_free (buffer3);
00222   g_free (buffer2);
00223 #if DEBUG_OPTIMIZE
00224   fprintf (stderr, "optimize_parse: %s\n", buffer);
00225 #endif
00226   system (buffer);
00227
00228   // Checking the objective value function
00229   if (optimize->evaluator)
00230     {
00231       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00232       buffer2 = g_path_get_dirname (optimize->evaluator);
00233       buffer3 = g_path_get_basename (optimize->evaluator);
00234       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235       snprintf (buffer, 512, "\"%s\" %s %s %s",
00236                 buffer4, output, optimize->experiment[experiment], result);
00237       g_free (buffer4);
00238       g_free (buffer3);
00239       g_free (buffer2);
00240 #if DEBUG_OPTIMIZE
00241       fprintf (stderr, "optimize_parse: %s\n", buffer);
00242       fprintf (stderr, "optimize_parse: result=%s\n", result);
00243 #endif
00244       system (buffer);
00245       file_result = g_fopen (result, "r");
00246       e = atof (fgets (buffer, 512, file_result));
00247       fclose (file_result);
00248     }
00249   else
00250     {
00251 #if DEBUG_OPTIMIZE
00252       fprintf (stderr, "optimize_parse: output=%s\n", output);
00253 #endif
00254       strcpy (result, "");
00255       file_result = g_fopen (output, "r");
00256       e = atof (fgets (buffer, 512, file_result));
```

```
00257       fclose (file_result);
00258     }
00259
00260   // Removing files
00261 #if !DEBUG_OPTIMIZE
00262   for (i = 0; i < optimize->ninputs; ++i)
00263     {
00264       if (optimize->file[i][0])
00265         {
00266           snprintf (buffer, 512, RM " %s", &input[i][0]);
00267           system (buffer);
00268         }
00269     }
00270   snprintf (buffer, 512, RM " %s %s", output, result);
00271   system (buffer);
00272 #endif
00273
00274   // Processing pending events
00275   if (show_pending)
00276     show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
```

Here is the call graph for this function:

4.17.2.23 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file optimize.c.

```
00394 {
00395   unsigned int i;
00396   char buffer[512];
00397 #if HAVE_MPI
00398   if (optimize->mpi_rank)
00399     return;
00400 #endif
00401   printf ("%s\n", _("Best result"));
00402   fprintf (optimize->file_result, "%s\n", _("Best result"));
00403   printf ("error = %.15le\n", optimize->error_old[0]);
00404   fprintf (optimize->file_result, "error = %.15le\n",
      optimize->error_old[0]);
00405   for (i = 0; i < optimize->nvariables; ++i)
00406     {
00407       snprintf (buffer, 512, "%s = %s\n",
00408                 optimize->label[i], format[optimize->
      precision[i]]);
00409       printf (buffer, optimize->value_old[i]);
00410       fprintf (optimize->file_result, buffer, optimize->
      value_old[i]);
00411     }
00412   fflush (optimize->file_result);
00413 }
```

4.17.2.24 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1262 of file optimize.c.

```
01263 {
01264   unsigned int i, j;
01265   double d;
01266 #if HAVE_MPI
01267   MPI_Status mpi_stat;
01268 #endif
01269 #if DEBUG_OPTIMIZE
01270   fprintf (stderr, "optimize_refine: start\n");
01271 #endif
01272 #if HAVE_MPI
01273   if (!optimize->mpi_rank)
01274     {
01275 #endif
01276       for (j = 0; j < optimize->nvariables; ++j)
01277         {
01278           optimize->rangemin[j] = optimize->rangemax[j]
01279             = optimize->value_old[j];
01280         }
01281       for (i = 0; ++i < optimize->nbest;)
01282         {
01283           for (j = 0; j < optimize->nvariables; ++j)
01284             {
01285               optimize->rangemin[j]
01286                 = fmin (optimize->rangemin[j],
01287                         optimize->value_old[i * optimize->
      nvariables + j]);
01288               optimize->rangemax[j]
01289                 = fmax (optimize->rangemax[j],
01290                         optimize->value_old[i * optimize->
      nvariables + j]);
01291             }
01292         }
01293       for (j = 0; j < optimize->nvariables; ++j)
01294         {
01295           d = optimize->tolerance
01296             * (optimize->rangemax[j] - optimize->
      rangemin[j]);
01297           switch (optimize->algorithm)
01298             {
01299             case ALGORITHM_MONTE_CARLO:
01300               d *= 0.5;
01301               break;
01302             default:
01303               if (optimize->nsweeps[j] > 1)
01304                 d /= optimize->nsweeps[j] - 1;
01305               else
01306                 d = 0.;
01307             }
01308           optimize->rangemin[j] -= d;
01309           optimize->rangemin[j]
01310             = fmax (optimize->rangemin[j], optimize->
      rangeminabs[j]);
01311           optimize->rangemax[j] += d;
01312           optimize->rangemax[j]
01313             = fmin (optimize->rangemax[j], optimize->
      rangemaxabs[j]);
01314           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01315                   optimize->rangemin[j], optimize->
      rangemax[j]);
01316           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01317                    optimize->label[j], optimize->rangemin[j],
01318                    optimize->rangemax[j]);
01319         }
01320 #if HAVE_MPI
01321       for (i = 1; i < ntasks; ++i)
01322         {
01323           MPI_Send (optimize->rangemin, optimize->
      nvariables, MPI_DOUBLE, i,
01324                     1, MPI_COMM_WORLD);
01325           MPI_Send (optimize->rangemax, optimize->
      nvariables, MPI_DOUBLE, i,
01326                     1, MPI_COMM_WORLD);
01327         }
01328     }
```

```
01329   else
01330     {
01331       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
      1,
01332                 MPI_COMM_WORLD, &mpi_stat);
01333       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
      1,
01334                 MPI_COMM_WORLD, &mpi_stat);
01335     }
01336 #endif
01337 #if DEBUG_OPTIMIZE
01338   fprintf (stderr, "optimize_refine: end\n");
01339 #endif
01340 }
```

4.17.2.25 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1182 of file optimize.c.

```
01183 {
01184   unsigned int i, j;
01185 #if DEBUG_OPTIMIZE
01186   fprintf (stderr, "optimize_save_old: start\n");
01187   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01188 #endif
01189   memcpy (optimize->error_old, optimize->error_best,
01190           optimize->nbest * sizeof (double));
01191   for (i = 0; i < optimize->nbest; ++i)
01192     {
01193       j = optimize->simulation_best[i];
01194 #if DEBUG_OPTIMIZE
01195       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01196 #endif
01197       memcpy (optimize->value_old + i * optimize->
      nvariables,
01198               optimize->value + j * optimize->nvariables,
01199               optimize->nvariables * sizeof (double));
01200     }
01201 #if DEBUG_OPTIMIZE
01202   for (i = 0; i < optimize->nvariables; ++i)
01203     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01204              i, optimize->value_old[i]);
01205   fprintf (stderr, "optimize_save_old: end\n");
01206 #endif
01207 }
```

4.17.2.26 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.

```
00421 {
00422   unsigned int i;
00423   char buffer[64];
00424 #if DEBUG_OPTIMIZE
00425   fprintf (stderr, "optimize_save_variables: start\n");
00426 #endif
00427   for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430       fprintf (optimize->file_variables, buffer,
00431                optimize->value[simulation * optimize->
      nvariables + i]);
00432     }
00433   fprintf (optimize->file_variables, "%.14le\n", error);
00434   fflush (optimize->file_variables);
00435 #if DEBUG_OPTIMIZE
00436   fprintf (stderr, "optimize_save_variables: end\n");
00437 #endif
00438 }
```

4.17.2.27 optimize_sequential()

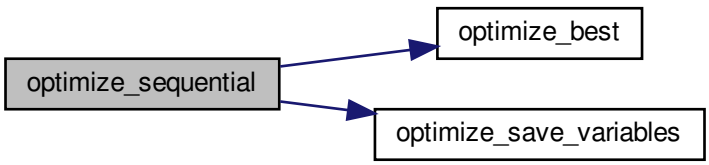
```
void optimize_sequential ( )
```

Function to optimize sequentially.

Definition at line 485 of file optimize.c.

```
00486 {
00487   unsigned int i;
00488   double e;
00489 #if DEBUG_OPTIMIZE
00490   fprintf (stderr, "optimize_sequential: start\n");
00491   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492            optimize->nstart, optimize->nend);
00493 #endif
00494   for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496       e = optimize_norm (i);
00497       optimize_best (i, e);
00498       optimize_save_variables (i, e);
00499       if (e < optimize->threshold)
00500         {
00501           optimize->stop = 1;
00502           break;
00503         }
00504 #if DEBUG_OPTIMIZE
00505       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506 #endif
00507     }
00508 #if DEBUG_OPTIMIZE
00509   fprintf (stderr, "optimize_sequential: end\n");
00510 #endif
00511 }
```

Here is the call graph for this function:



4.17.2.28 optimize_step()

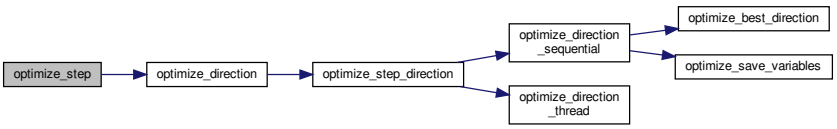
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1346 of file optimize.c.

```
01347 {
01348 #if DEBUG_OPTIMIZE
01349   fprintf (stderr, "optimize_step: start\n");
01350 #endif
01351   optimize_algorithm ();
01352   if (optimize->nsteps)
01353     optimize_direction ();
01354 #if DEBUG_OPTIMIZE
01355   fprintf (stderr, "optimize_step: end\n");
01356 #endif
01357 }
```

Here is the call graph for this function:



4.17.2.29 optimize_step_direction()

```
void optimize_step_direction (
             unsigned int simulation )
```

Function to do a step of the direction search method.

Parameters

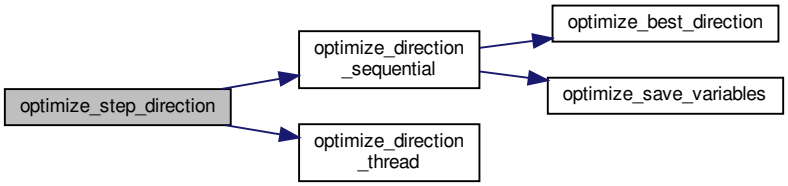
simulation	Simulation number.

Definition at line 967 of file optimize.c.

```
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
```

```
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->
      nvariables;
00979 #if DEBUG_OPTIMIZE
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011               / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
01022             (NULL, (GThreadFunc) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
```

Here is the call graph for this function:



4.17.2.30 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file optimize.c.

```
00666 {
00667   unsigned int i, j, k, l;
00668   double e;
00669   GThread *thread[nthreads];
00670   ParallelData data[nthreads];
00671 #if DEBUG_OPTIMIZE
00672   fprintf (stderr, "optimize_sweep: start\n");
00673 #endif
00674   for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676       k = i;
00677       for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679           l = k % optimize->nsweeps[j];
00680           k /= optimize->nsweeps[j];
00681           e = optimize->rangemin[j];
00682           if (optimize->nsweeps[j] > 1)
00683             e += l * (optimize->rangemax[j] - optimize->
      rangemin[j])
00684                 / (optimize->nsweeps[j] - 1);
00685           optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688   optimize->nsaveds = 0;
00689   if (nthreads <= 1)
00690     optimize_sequential ();
00691   else
00692     {
00693       for (i = 0; i < nthreads; ++i)
00694         {
00695           data[i].thread = i;
00696           thread[i]
00697             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699       for (i = 0; i < nthreads; ++i)
00700         g_thread_join (thread[i]);
00701     }
00702 #if HAVE_MPI
00703   // Communicating tasks results
00704   optimize_synchronise ();
00705 #endif
00706 #if DEBUG_OPTIMIZE
00707   fprintf (stderr, "optimize_sweep: end\n");
00708 #endif
00709 }
```

4.17.2.31 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file optimize.c.

```
00619 {
00620   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621   double error_best[optimize->nbest];
00622   MPI_Status mpi_stat;
00623 #if DEBUG_OPTIMIZE
00624   fprintf (stderr, "optimize_synchronise: start\n");
00625 #endif
00626   if (optimize->mpi_rank == 0)
00627     {
00628       for (i = 1; i < ntasks; ++i)
00629         {
00630           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00632                     MPI_COMM_WORLD, &mpi_stat);
00633           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634                     MPI_COMM_WORLD, &mpi_stat);
00635           optimize_merge (nsaveds, simulation_best, error_best);
00636           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637           if (stop)
00638             optimize->stop = 1;
00639         }
00640       for (i = 1; i < ntasks; ++i)
00641         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642     }
00643   else
00644     {
00645       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646       MPI_Send (optimize->simulation_best, optimize->
      nsaveds, MPI_INT, 0, 1,
00647                 MPI_COMM_WORLD);
00648       MPI_Send (optimize->error_best, optimize->
      nsaveds, MPI_DOUBLE, 0, 1,
00649                 MPI_COMM_WORLD);
00650       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00651       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00652       if (stop)
00653         optimize->stop = 1;
00654     }
00655 #if DEBUG_OPTIMIZE
00656   fprintf (stderr, "optimize_synchronise: end\n");
00657 #endif
00658 }
```

4.17.2.32 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.

```
00520 {
00521   unsigned int i, thread;
00522   double e;
00523 #if DEBUG_OPTIMIZE
00524   fprintf (stderr, "optimize_thread: start\n");
00525 #endif
```

```
00526   thread = data->thread;
00527 #if DEBUG_OPTIMIZE
00528   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529            optimize->thread[thread], optimize->thread[thread + 1]);
00530 #endif
00531   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533       e = optimize_norm (i);
00534       g_mutex_lock (mutex);
00535       optimize_best (i, e);
00536       optimize_save_variables (i, e);
00537       if (e < optimize->threshold)
00538         optimize->stop = 1;
00539       g_mutex_unlock (mutex);
00540       if (optimize->stop)
00541         break;
00542 #if DEBUG_OPTIMIZE
00543       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544 #endif
00545     }
00546 #if DEBUG_OPTIMIZE
00547   fprintf (stderr, "optimize_thread: end\n");
00548 #endif
00549   g_thread_exit (NULL);
00550   return NULL;
00551 }
```

4.18 optimize.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <sys/param.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <glib/gstdio.h>
00050 #include <json-glib/json-glib.h>
00051 #ifdef G_OS_WIN32
00052 #include <windows.h>
00053 #elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
00054 #include <alloca.h>
00055 #endif
00056 #if HAVE_MPI
00057 #include <mpi.h>
00058 #endif
00059 #include "genetic/genetic.h"
```

```
00060 #include "utils.h"
00061 #include "experiment.h"
00062 #include "variable.h"
00063 #include "input.h"
00064 #include "optimize.h"
00065
00066 #define DEBUG_OPTIMIZE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define RM "del"
00074 #else
00075 #define RM "rm"
00076 #endif
00077
00078 unsigned int nthreads_direction;
00080 void (*optimize_algorithm) ();
00082 double (*optimize_estimate_direction) (unsigned int variable,
00083                                        unsigned int estimate);
00085 double (*optimize_norm) (unsigned int simulation);
00087 Optimize optimize[1];
00088
00092 void
00093 optimize_input (unsigned int simulation,
00094                 char *input,
00095                 GMappedFile * stencil)
00096 {
00097   char buffer[32], value[32];
00098   GRegex *regex;
00099   FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101   gsize length;
00102   unsigned int i;
00103
00104 #if DEBUG_OPTIMIZE
00105   fprintf (stderr, "optimize_input: start\n");
00106 #endif
00107
00108   // Checking the file
00109   if (!stencil)
00110     goto optimize_input_end;
00111
00112   // Opening stencil
00113   content = g_mapped_file_get_contents (stencil);
00114   length = g_mapped_file_get_length (stencil);
00115 #if DEBUG_OPTIMIZE
00116   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117 #endif
00118   file = g_fopen (input, "w");
00119
00120   // Parsing stencil
00121   for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123 #if DEBUG_OPTIMIZE
00124       fprintf (stderr, "optimize_input: variable=%u\n", i);
00125 #endif
00126       snprintf (buffer, 32, "@variable%u@", i + 1);
00127       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00128                            NULL);
00129       if (i == 0)
00130         {
00131           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                              optimize->label[i],
00133                                              (GRegexMatchFlags) 0, NULL);
00134 #if DEBUG_OPTIMIZE
00135           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136 #endif
00137         }
00138       else
00139         {
00140           length = strlen (buffer3);
00141           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                              optimize->label[i],
00143                                              (GRegexMatchFlags) 0, NULL);
00144           g_free (buffer3);
00145         }
00146       g_regex_unref (regex);
00147       length = strlen (buffer2);
00148       snprintf (buffer, 32, "@value%u@", i + 1);
00149       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                            NULL);
00151       snprintf (value, 32, format[optimize->precision[i]],
00152                 optimize->value[simulation * optimize->nvariables + i]);
00153
00154 #if DEBUG_OPTIMIZE
00155       fprintf (stderr, "optimize_input: value=%s\n", value);
00156 #endif
```

```
00157       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                          (GRegexMatchFlags) 0, NULL);
00159       g_free (buffer2);
00160       g_regex_unref (regex);
00161     }
00162
00163   // Saving input file
00164   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165   g_free (buffer3);
00166   fclose (file);
00167
00168 optimize_input_end:
00169 #if DEBUG_OPTIMIZE
00170   fprintf (stderr, "optimize_input: end\n");
00171 #endif
00172   return;
00173 }
00174
00181 double
00182 optimize_parse (unsigned int simulation,
00183                 unsigned int experiment)
00184 {
00185   unsigned int i;
00186   double e;
00187   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188     *buffer3, *buffer4;
00189   FILE *file_result;
00190
00191 #if DEBUG_OPTIMIZE
00192   fprintf (stderr, "optimize_parse: start\n");
00193   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194            simulation, experiment);
00195 #endif
00196
00197   // Opening input files
00198   for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201 #if DEBUG_OPTIMIZE
00202       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203 #endif
00204       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00205     }
00206   for (; i < MAX_NINPUTS; ++i)
00207     strcpy (&input[i][0], "");
00208 #if DEBUG_OPTIMIZE
00209   fprintf (stderr, "optimize_parse: parsing end\n");
00210 #endif
00211
00212   // Performing the simulation
00213   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00214   buffer2 = g_path_get_dirname (optimize->simulator);
00215   buffer3 = g_path_get_basename (optimize->simulator);
00216   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00217   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00218             buffer4, input[0], input[1], input[2], input[3], input[4],
00219             input[5], input[6], input[7], output);
00220   g_free (buffer4);
00221   g_free (buffer3);
00222   g_free (buffer2);
00223 #if DEBUG_OPTIMIZE
00224   fprintf (stderr, "optimize_parse: %s\n", buffer);
00225 #endif
00226   system (buffer);
00227
00228   // Checking the objective value function
00229   if (optimize->evaluator)
00230     {
00231       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00232       buffer2 = g_path_get_dirname (optimize->evaluator);
00233       buffer3 = g_path_get_basename (optimize->evaluator);
00234       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235       snprintf (buffer, 512, "\"%s\" %s %s %s",
00236                 buffer4, output, optimize->experiment[experiment], result);
00237       g_free (buffer4);
00238       g_free (buffer3);
00239       g_free (buffer2);
00240 #if DEBUG_OPTIMIZE
00241       fprintf (stderr, "optimize_parse: %s\n", buffer);
00242       fprintf (stderr, "optimize_parse: result=%s\n", result);
00243 #endif
00244       system (buffer);
00245       file_result = g_fopen (result, "r");
00246       e = atof (fgets (buffer, 512, file_result));
00247       fclose (file_result);
00248     }
00249   else
```

```
00250     {
00251 #if DEBUG_OPTIMIZE
00252       fprintf (stderr, "optimize_parse: output=%s\n", output);
00253 #endif
00254       strcpy (result, "");
00255       file_result = g_fopen (output, "r");
00256       e = atof (fgets (buffer, 512, file_result));
00257       fclose (file_result);
00258     }
00259
00260   // Removing files
00261 #if !DEBUG_OPTIMIZE
00262   for (i = 0; i < optimize->ninputs; ++i)
00263     {
00264       if (optimize->file[i][0])
00265         {
00266           snprintf (buffer, 512, RM " %s", &input[i][0]);
00267           system (buffer);
00268         }
00269     }
00270   snprintf (buffer, 512, RM " %s %s", output, result);
00271   system (buffer);
00272 #endif
00273
00274   // Processing pending events
00275   if (show_pending)
00276     show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
00285
00291 double
00292 optimize_norm_euclidian (unsigned int simulation)
00293 {
00294   double e, ei;
00295   unsigned int i;
00296 #if DEBUG_OPTIMIZE
00297   fprintf (stderr, "optimize_norm_euclidian: start\n");
00298 #endif
00299   e = 0.;
00300   for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302       ei = optimize_parse (simulation, i);
00303       e += ei * ei;
00304     }
00305   e = sqrt (e);
00306 #if DEBUG_OPTIMIZE
00307   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308   fprintf (stderr, "optimize_norm_euclidian: end\n");
00309 #endif
00310   return e;
00311 }
00312
00318 double
00319 optimize_norm_maximum (unsigned int simulation)
00320 {
00321   double e, ei;
00322   unsigned int i;
00323 #if DEBUG_OPTIMIZE
00324   fprintf (stderr, "optimize_norm_maximum: start\n");
00325 #endif
00326   e = 0.;
00327   for (i = 0; i < optimize->nexperiments; ++i)
00328     {
00329       ei = fabs (optimize_parse (simulation, i));
00330       e = fmax (e, ei);
00331     }
00332 #if DEBUG_OPTIMIZE
00333   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334   fprintf (stderr, "optimize_norm_maximum: end\n");
00335 #endif
00336   return e;
00337 }
00338
00344 double
00345 optimize_norm_p (unsigned int simulation)
00346 {
00347   double e, ei;
00348   unsigned int i;
00349 #if DEBUG_OPTIMIZE
00350   fprintf (stderr, "optimize_norm_p: start\n");
00351 #endif
```

```
00352   e = 0.;
00353   for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355       ei = fabs (optimize_parse (simulation, i));
00356       e += pow (ei, optimize->p);
00357     }
00358   e = pow (e, 1. / optimize->p);
00359 #if DEBUG_OPTIMIZE
00360   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361   fprintf (stderr, "optimize_norm_p: end\n");
00362 #endif
00363   return e;
00364 }
00365
00371 double
00372 optimize_norm_taxicab (unsigned int simulation)
00373 {
00374   double e;
00375   unsigned int i;
00376 #if DEBUG_OPTIMIZE
00377   fprintf (stderr, "optimize_norm_taxicab: start\n");
00378 #endif
00379   e = 0.;
00380   for (i = 0; i < optimize->nexperiments; ++i)
00381     e += fabs (optimize_parse (simulation, i));
00382 #if DEBUG_OPTIMIZE
00383   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384   fprintf (stderr, "optimize_norm_taxicab: end\n");
00385 #endif
00386   return e;
00387 }
00388
00392 void
00393 optimize_print ()
00394 {
00395   unsigned int i;
00396   char buffer[512];
00397 #if HAVE_MPI
00398   if (optimize->mpi_rank)
00399     return;
00400 #endif
00401   printf ("%s\n", _("Best result"));
00402   fprintf (optimize->file_result, "%s\n", _("Best result"));
00403   printf ("error = %.15le\n", optimize->error_old[0]);
00404   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
00405   for (i = 0; i < optimize->nvariables; ++i)
00406     {
00407       snprintf (buffer, 512, "%s = %s\n",
00408                 optimize->label[i], format[optimize->precision[i]]);
00409       printf (buffer, optimize->value_old[i]);
00410       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00411     }
00412   fflush (optimize->file_result);
00413 }
00414
00418 void
00419 optimize_save_variables (unsigned int simulation,
00420                          double error)
00421 {
00422   unsigned int i;
00423   char buffer[64];
00424 #if DEBUG_OPTIMIZE
00425   fprintf (stderr, "optimize_save_variables: start\n");
00426 #endif
00427   for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430       fprintf (optimize->file_variables, buffer,
00431                optimize->value[simulation * optimize->nvariables + i]);
00432     }
00433   fprintf (optimize->file_variables, "%.14le\n", error);
00434   fflush (optimize->file_variables);
00435 #if DEBUG_OPTIMIZE
00436   fprintf (stderr, "optimize_save_variables: end\n");
00437 #endif
00438 }
00439
00443 void
00444 optimize_best (unsigned int simulation,
00445                double value)
00446 {
00447   unsigned int i, j;
00448   double e;
00449 #if DEBUG_OPTIMIZE
00450   fprintf (stderr, "optimize_best: start\n");
00451   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
```

```
00452           optimize->nsaveds, optimize->nbest);
00453 #endif
00454   if (optimize->nsaveds < optimize->nbest
00455       || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457       if (optimize->nsaveds < optimize->nbest)
00458         ++optimize->nsaveds;
00459       optimize->error_best[optimize->nsaveds - 1] = value;
00460       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461       for (i = optimize->nsaveds; --i;)
00462         {
00463           if (optimize->error_best[i] < optimize->error_best[i - 1])
00464             {
00465               j = optimize->simulation_best[i];
00466               e = optimize->error_best[i];
00467               optimize->simulation_best[i] = optimize->
    simulation_best[i - 1];
00468               optimize->error_best[i] = optimize->error_best[i - 1];
00469               optimize->simulation_best[i - 1] = j;
00470               optimize->error_best[i - 1] = e;
00471             }
00472           else
00473             break;
00474         }
00475     }
00476 #if DEBUG_OPTIMIZE
00477   fprintf (stderr, "optimize_best: end\n");
00478 #endif
00479 }
00480
00484 void
00485 optimize_sequential ()
00486 {
00487   unsigned int i;
00488   double e;
00489 #if DEBUG_OPTIMIZE
00490   fprintf (stderr, "optimize_sequential: start\n");
00491   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492            optimize->nstart, optimize->nend);
00493 #endif
00494   for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496       e = optimize_norm (i);
00497       optimize_best (i, e);
00498       optimize_save_variables (i, e);
00499       if (e < optimize->threshold)
00500         {
00501           optimize->stop = 1;
00502           break;
00503         }
00504 #if DEBUG_OPTIMIZE
00505       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506 #endif
00507     }
00508 #if DEBUG_OPTIMIZE
00509   fprintf (stderr, "optimize_sequential: end\n");
00510 #endif
00511 }
00512
00518 void *
00519 optimize_thread (ParallelData * data)
00520 {
00521   unsigned int i, thread;
00522   double e;
00523 #if DEBUG_OPTIMIZE
00524   fprintf (stderr, "optimize_thread: start\n");
00525 #endif
00526   thread = data->thread;
00527 #if DEBUG_OPTIMIZE
00528   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529            optimize->thread[thread], optimize->thread[thread + 1]);
00530 #endif
00531   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533       e = optimize_norm (i);
00534       g_mutex_lock (mutex);
00535       optimize_best (i, e);
00536       optimize_save_variables (i, e);
00537       if (e < optimize->threshold)
00538         optimize->stop = 1;
00539       g_mutex_unlock (mutex);
00540       if (optimize->stop)
00541         break;
00542 #if DEBUG_OPTIMIZE
00543       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544 #endif
00545     }
```

```
00546 #if DEBUG_OPTIMIZE
00547   fprintf (stderr, "optimize_thread: end\n");
00548 #endif
00549   g_thread_exit (NULL);
00550   return NULL;
00551 }
00552
00556 void
00557 optimize_merge (unsigned int nsaveds,
00558                 unsigned int *simulation_best,
00560                 double *error_best)
00562 {
00563   unsigned int i, j, k, s[optimize->nbest];
00564   double e[optimize->nbest];
00565 #if DEBUG_OPTIMIZE
00566   fprintf (stderr, "optimize_merge: start\n");
00567 #endif
00568   i = j = k = 0;
00569   do
00570     {
00571       if (i == optimize->nsaveds)
00572         {
00573           s[k] = simulation_best[j];
00574           e[k] = error_best[j];
00575           ++j;
00576           ++k;
00577           if (j == nsaveds)
00578             break;
00579         }
00580       else if (j == nsaveds)
00581         {
00582           s[k] = optimize->simulation_best[i];
00583           e[k] = optimize->error_best[i];
00584           ++i;
00585           ++k;
00586           if (i == optimize->nsaveds)
00587             break;
00588         }
00589       else if (optimize->error_best[i] > error_best[j])
00590         {
00591           s[k] = simulation_best[j];
00592           e[k] = error_best[j];
00593           ++j;
00594           ++k;
00595         }
00596       else
00597         {
00598           s[k] = optimize->simulation_best[i];
00599           e[k] = optimize->error_best[i];
00600           ++i;
00601           ++k;
00602         }
00603     }
00604   while (k < optimize->nbest);
00605   optimize->nsaveds = k;
00606   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00607   memcpy (optimize->error_best, e, k * sizeof (double));
00608 #if DEBUG_OPTIMIZE
00609   fprintf (stderr, "optimize_merge: end\n");
00610 #endif
00611 }
00612
00616 #if HAVE_MPI
00617 void
00618 optimize_synchronise ()
00619 {
00620   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621   double error_best[optimize->nbest];
00622   MPI_Status mpi_stat;
00623 #if DEBUG_OPTIMIZE
00624   fprintf (stderr, "optimize_synchronise: start\n");
00625 #endif
00626   if (optimize->mpi_rank == 0)
00627     {
00628       for (i = 1; i < ntasks; ++i)
00629         {
00630           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00632                     MPI_COMM_WORLD, &mpi_stat);
00633           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634                     MPI_COMM_WORLD, &mpi_stat);
00635           optimize_merge (nsaveds, simulation_best, error_best);
00636           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637           if (stop)
00638             optimize->stop = 1;
00639         }
00640       for (i = 1; i < ntasks; ++i)
```

```
00641         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642     }
00643   else
00644     {
00645       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00647                 MPI_COMM_WORLD);
00648       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00649                 MPI_COMM_WORLD);
00650       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00651       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00652       if (stop)
00653         optimize->stop = 1;
00654     }
00655 #if DEBUG_OPTIMIZE
00656   fprintf (stderr, "optimize_synchronise: end\n");
00657 #endif
00658 }
00659 #endif
00660
00664 void
00665 optimize_sweep ()
00666 {
00667   unsigned int i, j, k, l;
00668   double e;
00669   GThread *thread[nthreads];
00670   ParallelData data[nthreads];
00671 #if DEBUG_OPTIMIZE
00672   fprintf (stderr, "optimize_sweep: start\n");
00673 #endif
00674   for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676       k = i;
00677       for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679           l = k % optimize->nsweeps[j];
00680           k /= optimize->nsweeps[j];
00681           e = optimize->rangemin[j];
00682           if (optimize->nsweeps[j] > 1)
00683             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00684               / (optimize->nsweeps[j] - 1);
00685           optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688   optimize->nsaveds = 0;
00689   if (nthreads <= 1)
00690     optimize_sequential ();
00691   else
00692     {
00693       for (i = 0; i < nthreads; ++i)
00694         {
00695           data[i].thread = i;
00696           thread[i]
00697             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699       for (i = 0; i < nthreads; ++i)
00700         g_thread_join (thread[i]);
00701     }
00702 #if HAVE_MPI
00703   // Communicating tasks results
00704   optimize_synchronise ();
00705 #endif
00706 #if DEBUG_OPTIMIZE
00707   fprintf (stderr, "optimize_sweep: end\n");
00708 #endif
00709 }
00710
00714 void
00715 optimize_MonteCarlo ()
00716 {
00717   unsigned int i, j;
00718   GThread *thread[nthreads];
00719   ParallelData data[nthreads];
00720 #if DEBUG_OPTIMIZE
00721   fprintf (stderr, "optimize_MonteCarlo: start\n");
00722 #endif
00723   for (i = 0; i < optimize->nsimulations; ++i)
00724     for (j = 0; j < optimize->nvariables; ++j)
00725       optimize->value[i * optimize->nvariables + j]
00726         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00727         * (optimize->rangemax[j] - optimize->rangemin[j]);
00728   optimize->nsaveds = 0;
00729   if (nthreads <= 1)
00730     optimize_sequential ();
00731   else
00732     {
00733       for (i = 0; i < nthreads; ++i)
```

```
00734         {
00735           data[i].thread = i;
00736           thread[i]
00737             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00738         }
00739       for (i = 0; i < nthreads; ++i)
00740         g_thread_join (thread[i]);
00741     }
00742 #if HAVE_MPI
00743   // Communicating tasks results
00744   optimize_synchronise ();
00745 #endif
00746 #if DEBUG_OPTIMIZE
00747   fprintf (stderr, "optimize_MonteCarlo: end\n");
00748 #endif
00749 }
00750
00754 void
00755 optimize_orthogonal ()
00756 {
00757   unsigned int i, j, k, l;
00758   double e;
00759   GThread *thread[nthreads];
00760   ParallelData data[nthreads];
00761 #if DEBUG_OPTIMIZE
00762   fprintf (stderr, "optimize_orthogonal: start\n");
00763 #endif
00764   for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766       k = i;
00767       for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769           l = k % optimize->nsweeps[j];
00770           k /= optimize->nsweeps[j];
00771           e = optimize->rangemin[j];
00772           if (optimize->nsweeps[j] > 1)
00773             e += (l + gsl_rng_uniform (optimize->rng))
00774                               * (optimize->rangemax[j] - optimize->
      rangemin[j])
00775                 / optimize->nsweeps[j];
00776           optimize->value[i * optimize->nvariables + j] = e;
00777         }
00778     }
00779   optimize->nsaveds = 0;
00780   if (nthreads <= 1)
00781     optimize_sequential ();
00782   else
00783     {
00784       for (i = 0; i < nthreads; ++i)
00785         {
00786           data[i].thread = i;
00787           thread[i]
00788             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789         }
00790       for (i = 0; i < nthreads; ++i)
00791         g_thread_join (thread[i]);
00792     }
00793 #if HAVE_MPI
00794   // Communicating tasks results
00795   optimize_synchronise ();
00796 #endif
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_orthogonal: end\n");
00799 #endif
00800 }
00801
00805 void
00806 optimize_best_direction (unsigned int simulation,
00808                          double value)
00810 {
00811 #if DEBUG_OPTIMIZE
00812   fprintf (stderr, "optimize_best_direction: start\n");
00813   fprintf (stderr,
00814            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00815            simulation, value, optimize->error_best[0]);
00816 #endif
00817   if (value < optimize->error_best[0])
00818     {
00819       optimize->error_best[0] = value;
00820       optimize->simulation_best[0] = simulation;
00821 #if DEBUG_OPTIMIZE
00822       fprintf (stderr,
00823                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00824                simulation, value);
00825 #endif
00826     }
00827 #if DEBUG_OPTIMIZE
```

```
00828   fprintf (stderr, "optimize_best_direction: end\n");
00829 #endif
00830 }
00831
00835 void
00836 optimize_direction_sequential (unsigned int simulation)
00837 {
00838   unsigned int i, j;
00839   double e;
00840 #if DEBUG_OPTIMIZE
00841   fprintf (stderr, "optimize_direction_sequential: start\n");
00842   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00843            "nend_direction=%u\n",
00844            optimize->nstart_direction, optimize->nend_direction);
00845 #endif
00846   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00847     {
00848       j = simulation + i;
00849       e = optimize_norm (j);
00850       optimize_best_direction (j, e);
00851       optimize_save_variables (j, e);
00852       if (e < optimize->threshold)
00853         {
00854           optimize->stop = 1;
00855           break;
00856         }
00857 #if DEBUG_OPTIMIZE
00858       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00859 #endif
00860     }
00861 #if DEBUG_OPTIMIZE
00862   fprintf (stderr, "optimize_direction_sequential: end\n");
00863 #endif
00864 }
00865
00871 void *
00872 optimize_direction_thread (ParallelData * data)
00873 {
00874   unsigned int i, thread;
00875   double e;
00876 #if DEBUG_OPTIMIZE
00877   fprintf (stderr, "optimize_direction_thread: start\n");
00878 #endif
00879   thread = data->thread;
00880 #if DEBUG_OPTIMIZE
00881   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00882            thread,
00883            optimize->thread_direction[thread],
00884            optimize->thread_direction[thread + 1]);
00885 #endif
00886   for (i = optimize->thread_direction[thread];
00887        i < optimize->thread_direction[thread + 1]; ++i)
00888     {
00889       e = optimize_norm (i);
00890       g_mutex_lock (mutex);
00891       optimize_best_direction (i, e);
00892       optimize_save_variables (i, e);
00893       if (e < optimize->threshold)
00894         optimize->stop = 1;
00895       g_mutex_unlock (mutex);
00896       if (optimize->stop)
00897         break;
00898 #if DEBUG_OPTIMIZE
00899       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00900 #endif
00901     }
00902 #if DEBUG_OPTIMIZE
00903   fprintf (stderr, "optimize_direction_thread: end\n");
00904 #endif
00905   g_thread_exit (NULL);
00906   return NULL;
00907 }
00908
00912 double
00913 optimize_estimate_direction_random (unsigned int variable,
00915                                     unsigned int estimate
00916                                     __attribute__ ((unused)))
00918 {
00919   double x;
00920 #if DEBUG_OPTIMIZE
00921   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00922 #endif
00923   x = optimize->direction[variable]
00924     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00925 #if DEBUG_OPTIMIZE
00926   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00927            variable, x);
```

```
00928   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00929 #endif
00930   return x;
00931 }
00932
00936 double
00937 optimize_estimate_direction_coordinates (unsigned int variable,
00939                                          unsigned int estimate)
00941 {
00942   double x;
00943 #if DEBUG_OPTIMIZE
00944   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945 #endif
00946   x = optimize->direction[variable];
00947   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949       if (estimate & 1)
00950         x += optimize->step[variable];
00951       else
00952         x -= optimize->step[variable];
00953     }
00954 #if DEBUG_OPTIMIZE
00955   fprintf (stderr,
00956            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00957            variable, x);
00958   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959 #endif
00960   return x;
00961 }
00962
00966 void
00967 optimize_step_direction (unsigned int simulation)
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->nvariables;
00979 #if DEBUG_OPTIMIZE
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j, i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011             / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
```

```
01022             (NULL, (GThreadFunc) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
01031
01035 void
01036 optimize_direction ()
01037 {
01038   unsigned int i, j, k, b, s, adjust;
01039 #if DEBUG_OPTIMIZE
01040   fprintf (stderr, "optimize_direction: start\n");
01041 #endif
01042   for (i = 0; i < optimize->nvariables; ++i)
01043     optimize->direction[i] = 0.;
01044   b = optimize->simulation_best[0] * optimize->nvariables;
01045   s = optimize->nsimulations;
01046   adjust = 1;
01047   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01048     {
01049 #if DEBUG_OPTIMIZE
01050       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01051                i, optimize->simulation_best[0]);
01052 #endif
01053       optimize_step_direction (s);
01054       k = optimize->simulation_best[0] * optimize->nvariables;
01055 #if DEBUG_OPTIMIZE
01056       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01057                i, optimize->simulation_best[0]);
01058 #endif
01059       if (k == b)
01060         {
01061           if (adjust)
01062             for (j = 0; j < optimize->nvariables; ++j)
01063               optimize->step[j] *= 0.5;
01064           for (j = 0; j < optimize->nvariables; ++j)
01065             optimize->direction[j] = 0.;
01066           adjust = 1;
01067         }
01068       else
01069         {
01070           for (j = 0; j < optimize->nvariables; ++j)
01071             {
01072 #if DEBUG_OPTIMIZE
01073               fprintf (stderr,
01074                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01075                        j, optimize->value[k + j], j, optimize->value[b + j]);
01076 #endif
01077               optimize->direction[j]
01078                 = (1. - optimize->relaxation) * optimize->direction[j]
01079                 + optimize->relaxation
01080                 * (optimize->value[k + j] - optimize->value[b + j]);
01081 #if DEBUG_OPTIMIZE
01082               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01083                        j, optimize->direction[j]);
01084 #endif
01085             }
01086           adjust = 0;
01087         }
01088     }
01089 #if DEBUG_OPTIMIZE
01090   fprintf (stderr, "optimize_direction: end\n");
01091 #endif
01092 }
01093
01099 double
01100 optimize_genetic_objective (Entity * entity)
01101 {
01102   unsigned int j;
01103   double objective;
01104   char buffer[64];
01105 #if DEBUG_OPTIMIZE
01106   fprintf (stderr, "optimize_genetic_objective: start\n");
01107 #endif
01108   for (j = 0; j < optimize->nvariables; ++j)
01109     {
01110       optimize->value[entity->id * optimize->nvariables + j]
01111         = genetic_get_variable (entity, optimize->genetic_variable + j);
01112     }
01113   objective = optimize_norm (entity->id);
01114   g_mutex_lock (mutex);
01115   for (j = 0; j < optimize->nvariables; ++j)
01116     {
```

```
01117       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01118       fprintf (optimize->file_variables, buffer,
01119                genetic_get_variable (entity, optimize->genetic_variable + j));
01120     }
01121   fprintf (optimize->file_variables, "%.14le\n", objective);
01122   g_mutex_unlock (mutex);
01123 #if DEBUG_OPTIMIZE
01124   fprintf (stderr, "optimize_genetic_objective: end\n");
01125 #endif
01126   return objective;
01127 }
01128
01132 void
01133 optimize_genetic ()
01134 {
01135   double *best_variable = NULL;
01136   char *best_genome = NULL;
01137   double best_objective = 0.;
01138 #if DEBUG_OPTIMIZE
01139   fprintf (stderr, "optimize_genetic: start\n");
01140   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01141            nthreads);
01142   fprintf (stderr,
01143            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01144            optimize->nvariables, optimize->nsimulations, optimize->
      niterations);
01145   fprintf (stderr,
01146            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01147            optimize->mutation_ratio, optimize->reproduction_ratio,
01148            optimize->adaptation_ratio);
01149 #endif
01150   genetic_algorithm_default (optimize->nvariables,
01151                              optimize->genetic_variable,
01152                              optimize->nsimulations,
01153                              optimize->niterations,
01154                              optimize->mutation_ratio,
01155                              optimize->reproduction_ratio,
01156                              optimize->adaptation_ratio,
01157                              optimize->seed,
01158                              optimize->threshold,
01159                              &optimize_genetic_objective,
01160                              &best_genome, &best_variable, &best_objective);
01161 #if DEBUG_OPTIMIZE
01162   fprintf (stderr, "optimize_genetic: the best\n");
01163 #endif
01164   optimize->error_old = (double *) g_malloc (sizeof (double));
01165   optimize->value_old
01166     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01167   optimize->error_old[0] = best_objective;
01168   memcpy (optimize->value_old, best_variable,
01169           optimize->nvariables * sizeof (double));
01170   g_free (best_genome);
01171   g_free (best_variable);
01172   optimize_print ();
01173 #if DEBUG_OPTIMIZE
01174   fprintf (stderr, "optimize_genetic: end\n");
01175 #endif
01176 }
01177
01181 void
01182 optimize_save_old ()
01183 {
01184   unsigned int i, j;
01185 #if DEBUG_OPTIMIZE
01186   fprintf (stderr, "optimize_save_old: start\n");
01187   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01188 #endif
01189   memcpy (optimize->error_old, optimize->error_best,
01190           optimize->nbest * sizeof (double));
01191   for (i = 0; i < optimize->nbest; ++i)
01192     {
01193       j = optimize->simulation_best[i];
01194 #if DEBUG_OPTIMIZE
01195       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01196 #endif
01197       memcpy (optimize->value_old + i * optimize->nvariables,
01198               optimize->value + j * optimize->nvariables,
01199               optimize->nvariables * sizeof (double));
01200     }
01201 #if DEBUG_OPTIMIZE
01202   for (i = 0; i < optimize->nvariables; ++i)
01203     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01204              i, optimize->value_old[i]);
01205   fprintf (stderr, "optimize_save_old: end\n");
01206 #endif
01207 }
01208
```

```
01213 void
01214 optimize_merge_old ()
01215 {
01216   unsigned int i, j, k;
01217   double v[optimize->nbest * optimize->nvariables], e[optimize->
      nbest],
01218     *enew, *eold;
01219 #if DEBUG_OPTIMIZE
01220   fprintf (stderr, "optimize_merge_old: start\n");
01221 #endif
01222   enew = optimize->error_best;
01223   eold = optimize->error_old;
01224   i = j = k = 0;
01225   do
01226     {
01227       if (*enew < *eold)
01228         {
01229           memcpy (v + k * optimize->nvariables,
01230                   optimize->value
01231                   + optimize->simulation_best[i] * optimize->
      nvariables,
01232                   optimize->nvariables * sizeof (double));
01233           e[k] = *enew;
01234           ++k;
01235           ++enew;
01236           ++i;
01237         }
01238       else
01239         {
01240           memcpy (v + k * optimize->nvariables,
01241                   optimize->value_old + j * optimize->nvariables,
01242                   optimize->nvariables * sizeof (double));
01243           e[k] = *eold;
01244           ++k;
01245           ++eold;
01246           ++j;
01247         }
01248     }
01249   while (k < optimize->nbest);
01250   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01251   memcpy (optimize->error_old, e, k * sizeof (double));
01252 #if DEBUG_OPTIMIZE
01253   fprintf (stderr, "optimize_merge_old: end\n");
01254 #endif
01255 }
01256
01261 void
01262 optimize_refine ()
01263 {
01264   unsigned int i, j;
01265   double d;
01266 #if HAVE_MPI
01267   MPI_Status mpi_stat;
01268 #endif
01269 #if DEBUG_OPTIMIZE
01270   fprintf (stderr, "optimize_refine: start\n");
01271 #endif
01272 #if HAVE_MPI
01273   if (!optimize->mpi_rank)
01274     {
01275 #endif
01276       for (j = 0; j < optimize->nvariables; ++j)
01277         {
01278           optimize->rangemin[j] = optimize->rangemax[j]
01279             = optimize->value_old[j];
01280         }
01281       for (i = 0; ++i < optimize->nbest;)
01282         {
01283           for (j = 0; j < optimize->nvariables; ++j)
01284             {
01285               optimize->rangemin[j]
01286                 = fmin (optimize->rangemin[j],
01287                         optimize->value_old[i * optimize->nvariables + j]);
01288               optimize->rangemax[j]
01289                 = fmax (optimize->rangemax[j],
01290                         optimize->value_old[i * optimize->nvariables + j]);
01291             }
01292         }
01293       for (j = 0; j < optimize->nvariables; ++j)
01294         {
01295           d = optimize->tolerance
01296             * (optimize->rangemax[j] - optimize->rangemin[j]);
01297           switch (optimize->algorithm)
01298             {
01299             case ALGORITHM_MONTE_CARLO:
01300               d *= 0.5;
01301               break;
```

```
01302             default:
01303               if (optimize->nsweeps[j] > 1)
01304                 d /= optimize->nsweeps[j] - 1;
01305               else
01306                 d = 0.;
01307             }
01308           optimize->rangemin[j] -= d;
01309           optimize->rangemin[j]
01310             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01311           optimize->rangemax[j] += d;
01312           optimize->rangemax[j]
01313             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01314           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01315                   optimize->rangemin[j], optimize->rangemax[j]);
01316           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01317                    optimize->label[j], optimize->rangemin[j],
01318                    optimize->rangemax[j]);
01319         }
01320 #if HAVE_MPI
01321       for (i = 1; i < ntasks; ++i)
01322         {
01323           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01324                     1, MPI_COMM_WORLD);
01325           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01326                     1, MPI_COMM_WORLD);
01327         }
01328     }
01329   else
01330     {
01331       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01332                 MPI_COMM_WORLD, &mpi_stat);
01333       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01334                 MPI_COMM_WORLD, &mpi_stat);
01335     }
01336 #endif
01337 #if DEBUG_OPTIMIZE
01338   fprintf (stderr, "optimize_refine: end\n");
01339 #endif
01340 }
01341
01345 void
01346 optimize_step ()
01347 {
01348 #if DEBUG_OPTIMIZE
01349   fprintf (stderr, "optimize_step: start\n");
01350 #endif
01351   optimize_algorithm ();
01352   if (optimize->nsteps)
01353     optimize_direction ();
01354 #if DEBUG_OPTIMIZE
01355   fprintf (stderr, "optimize_step: end\n");
01356 #endif
01357 }
01358
01362 void
01363 optimize_iterate ()
01364 {
01365   unsigned int i;
01366 #if DEBUG_OPTIMIZE
01367   fprintf (stderr, "optimize_iterate: start\n");
01368 #endif
01369   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01370   optimize->value_old =
01371     (double *) g_malloc (optimize->nbest * optimize->nvariables *
01372                          sizeof (double));
01373   optimize_step ();
01374   optimize_save_old ();
01375   optimize_refine ();
01376   optimize_print ();
01377   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01378     {
01379       optimize_step ();
01380       optimize_merge_old ();
01381       optimize_refine ();
01382       optimize_print ();
01383     }
01384 #if DEBUG_OPTIMIZE
01385   fprintf (stderr, "optimize_iterate: end\n");
01386 #endif
01387 }
01388
01392 void
01393 optimize_free ()
01394 {
01395   unsigned int i, j;
01396 #if DEBUG_OPTIMIZE
01397   fprintf (stderr, "optimize_free: start\n");
```

```
01398 #endif
01399   for (j = 0; j < optimize->ninputs; ++j)
01400     {
01401       for (i = 0; i < optimize->nexperiments; ++i)
01402         g_mapped_file_unref (optimize->file[j][i]);
01403       g_free (optimize->file[j]);
01404     }
01405   g_free (optimize->error_old);
01406   g_free (optimize->value_old);
01407   g_free (optimize->value);
01408   g_free (optimize->genetic_variable);
01409 #if DEBUG_OPTIMIZE
01410   fprintf (stderr, "optimize_free: end\n");
01411 #endif
01412 }
01413
01417 void
01418 optimize_open ()
01419 {
01420   GTimeZone *tz;
01421   GDateTime *t0, *t;
01422   unsigned int i, j;
01423
01424 #if DEBUG_OPTIMIZE
01425   char *buffer;
01426   fprintf (stderr, "optimize_open: start\n");
01427 #endif
01428
01429   // Getting initial time
01430 #if DEBUG_OPTIMIZE
01431   fprintf (stderr, "optimize_open: getting initial time\n");
01432 #endif
01433   tz = g_time_zone_new_utc ();
01434   t0 = g_date_time_new_now (tz);
01435
01436   // Obtaining and initing the pseudo-random numbers generator seed
01437 #if DEBUG_OPTIMIZE
01438   fprintf (stderr, "optimize_open: getting initial seed\n");
01439 #endif
01440   if (optimize->seed == DEFAULT_RANDOM_SEED)
01441     optimize->seed = input->seed;
01442   gsl_rng_set (optimize->rng, optimize->seed);
01443
01444   // Replacing the working directory
01445 #if DEBUG_OPTIMIZE
01446   fprintf (stderr, "optimize_open: replacing the working directory\n");
01447 #endif
01448   g_chdir (input->directory);
01449
01450   // Getting results file names
01451   optimize->result = input->result;
01452   optimize->variables = input->variables;
01453
01454   // Obtaining the simulator file
01455   optimize->simulator = input->simulator;
01456
01457   // Obtaining the evaluator file
01458   optimize->evaluator = input->evaluator;
01459
01460   // Reading the algorithm
01461   optimize->algorithm = input->algorithm;
01462   switch (optimize->algorithm)
01463     {
01464     case ALGORITHM_MONTE_CARLO:
01465       optimize_algorithm = optimize_MonteCarlo;
01466       break;
01467     case ALGORITHM_SWEEP:
01468       optimize_algorithm = optimize_sweep;
01469       break;
01470     case ALGORITHM_ORTHOGONAL:
01471       optimize_algorithm = optimize_orthogonal;
01472       break;
01473     default:
01474       optimize_algorithm = optimize_genetic;
01475       optimize->mutation_ratio = input->mutation_ratio;
01476       optimize->reproduction_ratio = input->
      reproduction_ratio;
01477       optimize->adaptation_ratio = input->adaptation_ratio;
01478     }
01479   optimize->nvariables = input->nvariables;
01480   optimize->nsimulations = input->nsimulations;
01481   optimize->niterations = input->niterations;
01482   optimize->nbest = input->nbest;
01483   optimize->tolerance = input->tolerance;
01484   optimize->nsteps = input->nsteps;
01485   optimize->nestimates = 0;
01486   optimize->threshold = input->threshold;
```

```
01487   optimize->stop = 0;
01488   if (input->nsteps)
01489     {
01490       optimize->relaxation = input->relaxation;
01491       switch (input->direction)
01492         {
01493         case DIRECTION_METHOD_COORDINATES:
01494           optimize->nestimates = 2 * optimize->nvariables;
01495           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
01496           break;
01497         default:
01498           optimize->nestimates = input->nestimates;
01499           optimize_estimate_direction =
      optimize_estimate_direction_random;
01500         }
01501     }
01502
01503 #if DEBUG_OPTIMIZE
01504   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01505 #endif
01506   optimize->simulation_best
01507     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01508   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01509
01510   // Reading the experimental data
01511 #if DEBUG_OPTIMIZE
01512   buffer = g_get_current_dir ();
01513   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01514   g_free (buffer);
01515 #endif
01516   optimize->nexperiments = input->nexperiments;
01517   optimize->ninputs = input->experiment->ninputs;
01518   optimize->experiment
01519     = (char **) alloca (input->nexperiments * sizeof (char *));
01520   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01521   for (i = 0; i < input->experiment->ninputs; ++i)
01522     optimize->file[i] = (GMappedFile **)
01523       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01524   for (i = 0; i < input->nexperiments; ++i)
01525     {
01526 #if DEBUG_OPTIMIZE
01527       fprintf (stderr, "optimize_open: i=%u\n", i);
01528 #endif
01529       optimize->experiment[i] = input->experiment[i].
      name;
01530       optimize->weight[i] = input->experiment[i].weight;
01531 #if DEBUG_OPTIMIZE
01532       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01533                optimize->experiment[i], optimize->weight[i]);
01534 #endif
01535       for (j = 0; j < input->experiment->ninputs; ++j)
01536         {
01537 #if DEBUG_OPTIMIZE
01538           fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01539 #endif
01540           optimize->file[j][i]
01541             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01542         }
01543     }
01544
01545   // Reading the variables data
01546 #if DEBUG_OPTIMIZE
01547   fprintf (stderr, "optimize_open: reading variables\n");
01548 #endif
01549   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01550   j = input->nvariables * sizeof (double);
01551   optimize->rangemin = (double *) alloca (j);
01552   optimize->rangeminabs = (double *) alloca (j);
01553   optimize->rangemax = (double *) alloca (j);
01554   optimize->rangemaxabs = (double *) alloca (j);
01555   optimize->step = (double *) alloca (j);
01556   j = input->nvariables * sizeof (unsigned int);
01557   optimize->precision = (unsigned int *) alloca (j);
01558   optimize->nsweeps = (unsigned int *) alloca (j);
01559   optimize->nbits = (unsigned int *) alloca (j);
01560   for (i = 0; i < input->nvariables; ++i)
01561     {
01562       optimize->label[i] = input->variable[i].name;
01563       optimize->rangemin[i] = input->variable[i].rangemin;
01564       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01565       optimize->rangemax[i] = input->variable[i].rangemax;
01566       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01567       optimize->precision[i] = input->variable[i].
      precision;
```

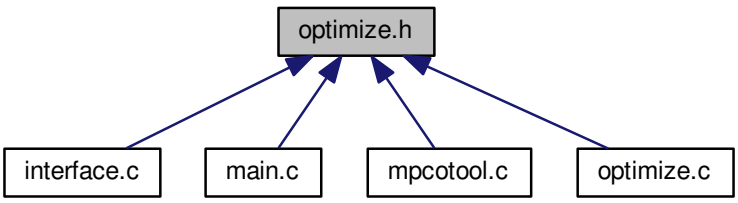
```
01568       optimize->step[i] = input->variable[i].step;
01569       optimize->nsweeps[i] = input->variable[i].nsweeps;
01570       optimize->nbits[i] = input->variable[i].nbits;
01571     }
01572   if (input->algorithm == ALGORITHM_SWEEP
01573           || input->algorithm == ALGORITHM_ORTHOGONAL)
01574     {
01575       optimize->nsimulations = 1;
01576       for (i = 0; i < input->nvariables; ++i)
01577         {
01578           optimize->nsimulations *= optimize->nsweeps[i];
01579 #if DEBUG_OPTIMIZE
01580           fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01581                    optimize->nsweeps[i], optimize->nsimulations);
01582 #endif
01583         }
01584     }
01585   if (optimize->nsteps)
01586     optimize->direction
01587       = (double *) alloca (optimize->nvariables * sizeof (double));
01588
01589   // Setting error norm
01590   switch (input->norm)
01591     {
01592     case ERROR_NORM_EUCLIDIAN:
01593       optimize_norm = optimize_norm_euclidian;
01594       break;
01595     case ERROR_NORM_MAXIMUM:
01596       optimize_norm = optimize_norm_maximum;
01597       break;
01598     case ERROR_NORM_P:
01599       optimize_norm = optimize_norm_p;
01600       optimize->p = input->p;
01601       break;
01602     default:
01603       optimize_norm = optimize_norm_taxicab;
01604     }
01605
01606   // Allocating values
01607 #if DEBUG_OPTIMIZE
01608   fprintf (stderr, "optimize_open: allocating variables\n");
01609   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01610            optimize->nvariables, optimize->algorithm);
01611 #endif
01612   optimize->genetic_variable = NULL;
01613   if (optimize->algorithm == ALGORITHM_GENETIC)
01614     {
01615       optimize->genetic_variable = (GeneticVariable *)
01616         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01617       for (i = 0; i < optimize->nvariables; ++i)
01618         {
01619 #if DEBUG_OPTIMIZE
01620           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01621                    i, optimize->rangemin[i], optimize->rangemax[i],
01622                    optimize->nbits[i]);
01623 #endif
01624           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
01625           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
01626           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01627         }
01628     }
01629 #if DEBUG_OPTIMIZE
01630   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01631            optimize->nvariables, optimize->nsimulations);
01632 #endif
01633   optimize->value = (double *)
01634     g_malloc ((optimize->nsimulations
01635                + optimize->nestimates * optimize->nsteps)
01636               * optimize->nvariables * sizeof (double));
01637
01638   // Calculating simulations to perform for each task
01639 #if HAVE_MPI
01640 #if DEBUG_OPTIMIZE
01641   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01642            optimize->mpi_rank, ntasks);
01643 #endif
01644   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
01645   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
01646   if (optimize->nsteps)
01647     {
01648       optimize->nstart_direction
01649         = optimize->mpi_rank * optimize->nestimates / ntasks;
01650       optimize->nend_direction
```

```
01651         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
01652     }
01653 #else
01654   optimize->nstart = 0;
01655   optimize->nend = optimize->nsimulations;
01656   if (optimize->nsteps)
01657     {
01658       optimize->nstart_direction = 0;
01659       optimize->nend_direction = optimize->nestimates;
01660     }
01661 #endif
01662 #if DEBUG_OPTIMIZE
01663   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01664            optimize->nend);
01665 #endif
01666
01667   // Calculating simulations to perform for each thread
01668   optimize->thread
01669     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01670   for (i = 0; i <= nthreads; ++i)
01671     {
01672       optimize->thread[i] = optimize->nstart
01673         + i * (optimize->nend - optimize->nstart) / nthreads;
01674 #if DEBUG_OPTIMIZE
01675       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01676                optimize->thread[i]);
01677 #endif
01678     }
01679   if (optimize->nsteps)
01680     optimize->thread_direction = (unsigned int *)
01681       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01682
01683   // Opening result files
01684   optimize->file_result = g_fopen (optimize->result, "w");
01685   optimize->file_variables = g_fopen (optimize->variables, "w");
01686
01687   // Performing the algorithm
01688   switch (optimize->algorithm)
01689     {
01690       // Genetic algorithm
01691     case ALGORITHM_GENETIC:
01692       optimize_genetic ();
01693       break;
01694
01695       // Iterative algorithm
01696     default:
01697       optimize_iterate ();
01698     }
01699
01700   // Getting calculation time
01701   t = g_date_time_new_now (tz);
01702   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01703   g_date_time_unref (t);
01704   g_date_time_unref (t0);
01705   g_time_zone_unref (tz);
01706   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01707   fprintf (optimize->file_result, "%s = %.6lg s\n",
01708            _("Calculation time"), optimize->calculation_time);
01709
01710   // Closing result files
01711   fclose (optimize->file_variables);
01712   fclose (optimize->file_result);
01713
01714 #if DEBUG_OPTIMIZE
01715   fprintf (stderr, "optimize_open: end\n");
01716 #endif
01717 }
```

4.19 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_direction (unsigned int simulation, double value)
- void optimize_direction_sequential (unsigned int simulation)
- void * optimize_direction_thread (ParallelData *data)
- double **optimize_estimate_direction_random** (unsigned int variable, unsigned int estimate)
- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)
- void optimize_step_direction (unsigned int simulation)
- void optimize_direction ()
- 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_direction

  Number of threads for the direction search method.
- GMutex **mutex** [1]
- void(∗ optimize_algorithm)()

  Pointer to the function to perform a optimization algorithm step.
- double(∗ optimize_estimate_direction)(unsigned int variable, unsigned int estimate)

  Pointer to the function to estimate the direction.
- double(∗ optimize_norm)(unsigned int simulation)

  Pointer to the error norm function.
- Optimize optimize [1]

  Optimization data.

4.19.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Definition in file optimize.h.

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.

```
00446 {
00447   unsigned int i, j;
00448   double e;
00449 #if DEBUG_OPTIMIZE
00450   fprintf (stderr, "optimize_best: start\n");
00451   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00452            optimize->nsaveds, optimize->nbest);
00453 #endif
00454   if (optimize->nsaveds < optimize->nbest
00455       || value < optimize->error_best[optimize->nsaveds - 1])
00456     {
00457       if (optimize->nsaveds < optimize->nbest)
00458         ++optimize->nsaveds;
00459       optimize->error_best[optimize->nsaveds - 1] = value;
00460       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00461       for (i = optimize->nsaveds; --i;)
00462         {
00463           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00464             {
00465               j = optimize->simulation_best[i];
00466               e = optimize->error_best[i];
00467               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00468               optimize->error_best[i] = optimize->
      error_best[i - 1];
00469               optimize->simulation_best[i - 1] = j;
00470               optimize->error_best[i - 1] = e;
00471             }
00472           else
00473             break;
00474         }
00475     }
00476 #if DEBUG_OPTIMIZE
00477   fprintf (stderr, "optimize_best: end\n");
00478 #endif
00479 }
```

4.19.2.2 optimize_best_direction()

```
void optimize_best_direction (
             unsigned int simulation,
             double value )
```

Function to save the best simulation in a direction search method.

Parameters

simulation	Simulation number.
value	Objective function value.

Definition at line 806 of file optimize.c.

```
00810 {
00811 #if DEBUG_OPTIMIZE
00812   fprintf (stderr, "optimize_best_direction: start\n");
00813   fprintf (stderr,
00814            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00815            simulation, value, optimize->error_best[0]);
00816 #endif
00817   if (value < optimize->error_best[0])
00818     {
00819       optimize->error_best[0] = value;
00820       optimize->simulation_best[0] = simulation;
00821 #if DEBUG_OPTIMIZE
```

```
00822       fprintf (stderr,
00823                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00824                simulation, value);
00825 #endif
00826     }
00827 #if DEBUG_OPTIMIZE
00828   fprintf (stderr, "optimize_best_direction: end\n");
00829 #endif
00830 }
```

4.19.2.3 optimize_direction()

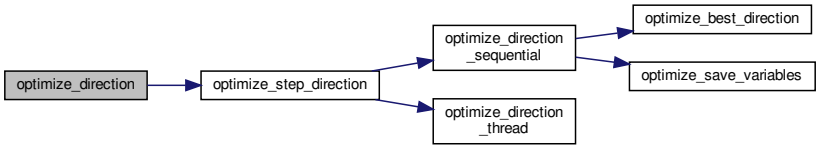
```
void optimize_direction ( )
```

Function to optimize with a direction search method.

Definition at line 1036 of file optimize.c.

```
01037 {
01038   unsigned int i, j, k, b, s, adjust;
01039 #if DEBUG_OPTIMIZE
01040   fprintf (stderr, "optimize_direction: start\n");
01041 #endif
01042   for (i = 0; i < optimize->nvariables; ++i)
01043     optimize->direction[i] = 0.;
01044   b = optimize->simulation_best[0] * optimize->
      nvariables;
01045   s = optimize->nsimulations;
01046   adjust = 1;
01047   for (i = 0; i < optimize->nsteps; ++i, s += optimize->
      nestimates, b = k)
01048     {
01049 #if DEBUG_OPTIMIZE
01050       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01051                i, optimize->simulation_best[0]);
01052 #endif
01053       optimize_step_direction (s);
01054       k = optimize->simulation_best[0] * optimize->
      nvariables;
01055 #if DEBUG_OPTIMIZE
01056       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01057                i, optimize->simulation_best[0]);
01058 #endif
01059       if (k == b)
01060         {
01061           if (adjust)
01062             for (j = 0; j < optimize->nvariables; ++j)
01063               optimize->step[j] *= 0.5;
01064           for (j = 0; j < optimize->nvariables; ++j)
01065             optimize->direction[j] = 0.;
01066           adjust = 1;
01067         }
01068       else
01069         {
01070           for (j = 0; j < optimize->nvariables; ++j)
01071             {
01072 #if DEBUG_OPTIMIZE
01073               fprintf (stderr,
01074                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01075                        j, optimize->value[k + j], j, optimize->
      value[b + j]);
01076 #endif
01077               optimize->direction[j]
01078                 = (1. - optimize->relaxation) * optimize->
      direction[j]
01079                 + optimize->relaxation
01080                 * (optimize->value[k + j] - optimize->value[b + j]);
01081 #if DEBUG_OPTIMIZE
01082               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01083                        j, optimize->direction[j]);
01084 #endif
01085             }
01086           adjust = 0;
01087         }
01088     }
01089 #if DEBUG_OPTIMIZE
01090   fprintf (stderr, "optimize_direction: end\n");
01091 #endif
01092 }
```

Here is the call graph for this function:



4.19.2.4 optimize_direction_sequential()

```
void optimize_direction_sequential (
             unsigned int simulation )
```

Function to estimate the direction search sequentially.

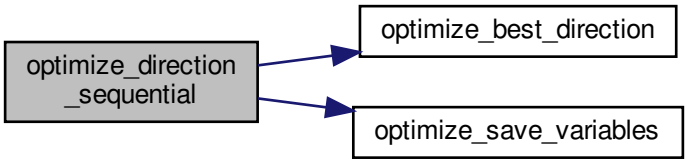
Parameters

simulation	Simulation number.

Definition at line 836 of file optimize.c.

```
00837 {
00838   unsigned int i, j;
00839   double e;
00840 #if DEBUG_OPTIMIZE
00841   fprintf (stderr, "optimize_direction_sequential: start\n");
00842   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00843            "nend_direction=%u\n",
00844            optimize->nstart_direction, optimize->
      nend_direction);
00845 #endif
00846   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00847     {
00848       j = simulation + i;
00849       e = optimize_norm (j);
00850       optimize_best_direction (j, e);
00851       optimize_save_variables (j, e);
00852       if (e < optimize->threshold)
00853         {
00854           optimize->stop = 1;
00855           break;
00856         }
00857 #if DEBUG_OPTIMIZE
00858       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00859 #endif
00860     }
00861 #if DEBUG_OPTIMIZE
00862   fprintf (stderr, "optimize_direction_sequential: end\n");
00863 #endif
00864 }
```

Here is the call graph for this function:



4.19.2.5 optimize_direction_thread()

```
void* optimize_direction_thread (
             ParallelData * data )
```

Function to estimate the direction search on a thread.

Returns

NULL

Parameters

data	Function data.

Definition at line 872 of file optimize.c.

```
00873 {
00874   unsigned int i, thread;
00875   double e;
00876 #if DEBUG_OPTIMIZE
00877   fprintf (stderr, "optimize_direction_thread: start\n");
00878 #endif
00879   thread = data->thread;
00880 #if DEBUG_OPTIMIZE
00881   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00882            thread,
00883            optimize->thread_direction[thread],
00884            optimize->thread_direction[thread + 1]);
00885 #endif
00886   for (i = optimize->thread_direction[thread];
00887        i < optimize->thread_direction[thread + 1]; ++i)
00888     {
00889       e = optimize_norm (i);
00890       g_mutex_lock (mutex);
00891       optimize_best_direction (i, e);
00892       optimize_save_variables (i, e);
00893       if (e < optimize->threshold)
00894         optimize->stop = 1;
00895       g_mutex_unlock (mutex);
00896       if (optimize->stop)
00897         break;
00898 #if DEBUG_OPTIMIZE
00899       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
```

```
00900 #endif
00901     }
00902 #if DEBUG_OPTIMIZE
00903   fprintf (stderr, "optimize_direction_thread: end\n");
00904 #endif
00905   g_thread_exit (NULL);
00906   return NULL;
00907 }
```

4.19.2.6 optimize_estimate_direction_coordinates()

```
double optimize_estimate_direction_coordinates (
             unsigned int variable,
             unsigned int estimate )
```

Function to estimate a component of the direction search vector.

Parameters

variable	Variable number.
estimate	Estimate number.

Definition at line 937 of file optimize.c.

```
00941 {
00942   double x;
00943 #if DEBUG_OPTIMIZE
00944   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00945 #endif
00946   x = optimize->direction[variable];
00947   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00948     {
00949       if (estimate & 1)
00950         x += optimize->step[variable];
00951       else
00952         x -= optimize->step[variable];
00953     }
00954 #if DEBUG_OPTIMIZE
00955   fprintf (stderr,
00956            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00957            variable, x);
00958   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00959 #endif
00960   return x;
00961 }
```

4.19.2.7 optimize_free()

```
void optimize_free ( )
```

Function to free the memory used by the Optimize struct.

Definition at line 1393 of file optimize.c.

```
01394 {
01395   unsigned int i, j;
01396 #if DEBUG_OPTIMIZE
01397   fprintf (stderr, "optimize_free: start\n");
01398 #endif
01399   for (j = 0; j < optimize->ninputs; ++j)
01400     {
01401       for (i = 0; i < optimize->nexperiments; ++i)
01402         g_mapped_file_unref (optimize->file[j][i]);
01403       g_free (optimize->file[j]);
01404     }
01405   g_free (optimize->error_old);
01406   g_free (optimize->value_old);
01407   g_free (optimize->value);
01408   g_free (optimize->genetic_variable);
01409 #if DEBUG_OPTIMIZE
01410   fprintf (stderr, "optimize_free: end\n");
01411 #endif
01412 }
```

4.19.2.8 optimize_genetic()

```
void optimize_genetic ( )
```

Function to optimize with the genetic algorithm.

Definition at line 1133 of file optimize.c.

```
01134 {
01135   double *best_variable = NULL;
01136   char *best_genome = NULL;
01137   double best_objective = 0.;
01138 #if DEBUG_OPTIMIZE
01139   fprintf (stderr, "optimize_genetic: start\n");
01140   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01141            nthreads);
01142   fprintf (stderr,
01143            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01144            optimize->nvariables, optimize->
      nsimulations, optimize->niterations);
01145   fprintf (stderr,
01146            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01147            optimize->mutation_ratio, optimize->
      reproduction_ratio,
01148            optimize->adaptation_ratio);
01149 #endif
01150   genetic_algorithm_default (optimize->nvariables,
01151                              optimize->genetic_variable,
01152                              optimize->nsimulations,
01153                              optimize->niterations,
01154                              optimize->mutation_ratio,
01155                              optimize->reproduction_ratio,
01156                              optimize->adaptation_ratio,
01157                              optimize->seed,
01158                              optimize->threshold,
01159                              &optimize_genetic_objective,
01160                              &best_genome, &best_variable, &best_objective);
01161 #if DEBUG_OPTIMIZE
01162   fprintf (stderr, "optimize_genetic: the best\n");
01163 #endif
01164   optimize->error_old = (double *) g_malloc (sizeof (double));
01165   optimize->value_old
01166     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01167   optimize->error_old[0] = best_objective;
01168   memcpy (optimize->value_old, best_variable,
01169           optimize->nvariables * sizeof (double));
01170   g_free (best_genome);
01171   g_free (best_variable);
01172   optimize_print ();
01173 #if DEBUG_OPTIMIZE
01174   fprintf (stderr, "optimize_genetic: end\n");
01175 #endif
01176 }
```

4.19.2.9 optimize_genetic_objective()

```
double optimize_genetic_objective (
                Entity * entity )
```

Function to calculate the objective function of an entity.

Returns

objective function value.

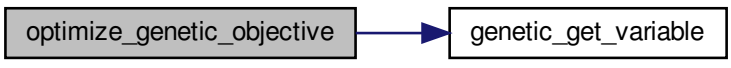
Parameters

entity	entity data.

Definition at line 1100 of file optimize.c.

```
01101 {
01102   unsigned int j;
01103   double objective;
01104   char buffer[64];
01105 #if DEBUG_OPTIMIZE
01106   fprintf (stderr, "optimize_genetic_objective: start\n");
01107 #endif
01108   for (j = 0; j < optimize->nvariables; ++j)
01109     {
01110       optimize->value[entity->id * optimize->nvariables + j]
01111         = genetic_get_variable (entity, optimize->genetic_variable + j);
01112     }
01113   objective = optimize_norm (entity->id);
01114   g_mutex_lock (mutex);
01115   for (j = 0; j < optimize->nvariables; ++j)
01116     {
01117       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01118       fprintf (optimize->file_variables, buffer,
01119                genetic_get_variable (entity, optimize->genetic_variable + j));
01120     }
01121   fprintf (optimize->file_variables, "%.14le\n", objective);
01122   g_mutex_unlock (mutex);
01123 #if DEBUG_OPTIMIZE
01124   fprintf (stderr, "optimize_genetic_objective: end\n");
01125 #endif
01126   return objective;
01127 }
```

Here is the call graph for this function:



4.19.2.10 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.

```
00096 {
00097   char buffer[32], value[32];
00098   GRegex *regex;
00099   FILE *file;
00100     char *buffer2, *buffer3 = NULL, *content;
00101   gsize length;
00102   unsigned int i;
00103
00104 #if DEBUG_OPTIMIZE
00105   fprintf (stderr, "optimize_input: start\n");
00106 #endif
00107
00108   // Checking the file
00109   if (!stencil)
00110     goto optimize_input_end;
00111
00112   // Opening stencil
00113   content = g_mapped_file_get_contents (stencil);
00114   length = g_mapped_file_get_length (stencil);
00115 #if DEBUG_OPTIMIZE
00116   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00117 #endif
00118   file = g_fopen (input, "w");
00119
00120   // Parsing stencil
00121   for (i = 0; i < optimize->nvariables; ++i)
00122     {
00123 #if DEBUG_OPTIMIZE
00124       fprintf (stderr, "optimize_input: variable=%u\n", i);
00125 #endif
00126       snprintf (buffer, 32, "@variable%u@", i + 1);
00127       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00128                            NULL);
00129       if (i == 0)
00130         {
00131           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00132                                              optimize->label[i],
00133                                              (GRegexMatchFlags) 0, NULL);
00134 #if DEBUG_OPTIMIZE
00135           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00136 #endif
00137         }
00138       else
00139         {
00140           length = strlen (buffer3);
00141           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00142                                              optimize->label[i],
00143                                              (GRegexMatchFlags) 0, NULL);
00144           g_free (buffer3);
00145         }
00146       g_regex_unref (regex);
00147       length = strlen (buffer2);
00148       snprintf (buffer, 32, "@value%u@", i + 1);
00149       regex = g_regex_new (buffer, (GRegexCompileFlags) 0, (GRegexMatchFlags) 0,
00150                            NULL);
00151       snprintf (value, 32, format[optimize->precision[i]],
00152                 optimize->value[simulation * optimize->
      nvariables + i]);
```

```
00153
00154 #if DEBUG_OPTIMIZE
00155       fprintf (stderr, "optimize_input: value=%s\n", value);
00156 #endif
00157       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00158                                          (GRegexMatchFlags) 0, NULL);
00159       g_free (buffer2);
00160       g_regex_unref (regex);
00161     }
00162
00163   // Saving input file
00164   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00165   g_free (buffer3);
00166   fclose (file);
00167
00168 optimize_input_end:
00169 #if DEBUG_OPTIMIZE
00170   fprintf (stderr, "optimize_input: end\n");
00171 #endif
00172   return;
00173 }
```

4.19.2.11 optimize_iterate()

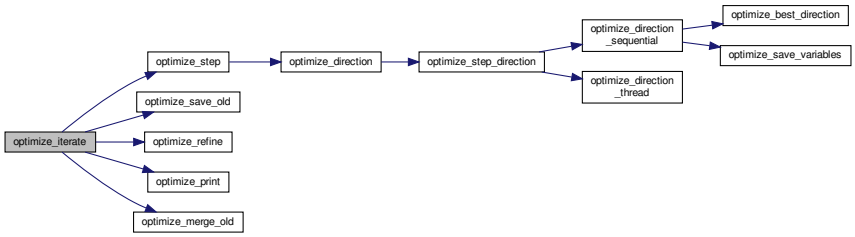
```
void optimize_iterate ( )
```

Function to iterate the algorithm.

Definition at line 1363 of file optimize.c.

```
01364 {
01365   unsigned int i;
01366 #if DEBUG_OPTIMIZE
01367   fprintf (stderr, "optimize_iterate: start\n");
01368 #endif
01369   optimize->error_old = (double *) g_malloc (optimize->
      nbest * sizeof (double));
01370   optimize->value_old =
01371     (double *) g_malloc (optimize->nbest * optimize->
      nvariables *
01372                          sizeof (double));
01373   optimize_step ();
01374   optimize_save_old ();
01375   optimize_refine ();
01376   optimize_print ();
01377   for (i = 1; i < optimize->niterations && !optimize->
      stop; ++i)
01378     {
01379       optimize_step ();
01380       optimize_merge_old ();
01381       optimize_refine ();
01382       optimize_print ();
01383     }
01384 #if DEBUG_OPTIMIZE
01385   fprintf (stderr, "optimize_iterate: end\n");
01386 #endif
01387 }
```

Here is the call graph for this function:



4.19.2.12 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.

```
00562 {
00563   unsigned int i, j, k, s[optimize->nbest];
00564   double e[optimize->nbest];
00565 #if DEBUG_OPTIMIZE
00566   fprintf (stderr, "optimize_merge: start\n");
00567 #endif
00568   i = j = k = 0;
00569   do
00570     {
00571       if (i == optimize->nsaveds)
00572         {
00573           s[k] = simulation_best[j];
00574           e[k] = error_best[j];
00575           ++j;
00576           ++k;
00577           if (j == nsaveds)
00578             break;
00579         }
00580       else if (j == nsaveds)
00581         {
00582           s[k] = optimize->simulation_best[i];
00583           e[k] = optimize->error_best[i];
00584           ++i;
00585           ++k;
00586           if (i == optimize->nsaveds)
00587             break;
00588         }
00589       else if (optimize->error_best[i] > error_best[j])
00590         {
00591           s[k] = simulation_best[j];
00592           e[k] = error_best[j];
00593           ++j;
00594           ++k;
00595         }
00596       else
00597         {
00598           s[k] = optimize->simulation_best[i];
00599           e[k] = optimize->error_best[i];
00600           ++i;
00601           ++k;
00602         }
00603     }
00604   while (k < optimize->nbest);
00605   optimize->nsaveds = k;
00606   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00607   memcpy (optimize->error_best, e, k * sizeof (double));
00608 #if DEBUG_OPTIMIZE
00609   fprintf (stderr, "optimize_merge: end\n");
00610 #endif
00611 }
```

4.19.2.13 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 1214 of file optimize.c.

```
01215 {
01216   unsigned int i, j, k;
01217   double v[optimize->nbest * optimize->nvariables], e[
      optimize->nbest],
01218     *enew, *eold;
01219 #if DEBUG_OPTIMIZE
01220   fprintf (stderr, "optimize_merge_old: start\n");
01221 #endif
01222   enew = optimize->error_best;
01223   eold = optimize->error_old;
01224   i = j = k = 0;
01225   do
01226     {
01227       if (*enew < *eold)
01228         {
01229           memcpy (v + k * optimize->nvariables,
01230                   optimize->value
01231                   + optimize->simulation_best[i] *
      optimize->nvariables,
01232                   optimize->nvariables * sizeof (double));
01233           e[k] = *enew;
01234           ++k;
01235           ++enew;
01236           ++i;
01237         }
01238       else
01239         {
01240           memcpy (v + k * optimize->nvariables,
01241                   optimize->value_old + j * optimize->
      nvariables,
01242                   optimize->nvariables * sizeof (double));
01243           e[k] = *eold;
01244           ++k;
01245           ++eold;
01246           ++j;
01247         }
01248     }
01249   while (k < optimize->nbest);
01250   memcpy (optimize->value_old, v, k * optimize->
      nvariables * sizeof (double));
01251   memcpy (optimize->error_old, e, k * sizeof (double));
01252 #if DEBUG_OPTIMIZE
01253   fprintf (stderr, "optimize_merge_old: end\n");
01254 #endif
01255 }
```

4.19.2.14 optimize_MonteCarlo()

```
void optimize_MonteCarlo ( )
```

Function to optimize with the Monte-Carlo algorithm.

Definition at line 715 of file optimize.c.

```
00716 {
00717   unsigned int i, j;
00718   GThread *thread[nthreads];
00719   ParallelData data[nthreads];
00720 #if DEBUG_OPTIMIZE
00721   fprintf (stderr, "optimize_MonteCarlo: start\n");
00722 #endif
00723   for (i = 0; i < optimize->nsimulations; ++i)
00724     for (j = 0; j < optimize->nvariables; ++j)
00725       optimize->value[i * optimize->nvariables + j]
00726         = optimize->rangemin[j] + gsl_rng_uniform (optimize->
      rng)
00727         * (optimize->rangemax[j] - optimize->rangemin[j]);
00728   optimize->nsaveds = 0;
00729   if (nthreads <= 1)
00730     optimize_sequential ();
00731   else
00732     {
00733       for (i = 0; i < nthreads; ++i)
00734         {
00735           data[i].thread = i;
00736           thread[i]
00737             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00738         }
00739       for (i = 0; i < nthreads; ++i)
00740         g_thread_join (thread[i]);
00741     }
00742 #if HAVE_MPI
00743   // Communicating tasks results
00744   optimize_synchronise ();
00745 #endif
00746 #if DEBUG_OPTIMIZE
00747   fprintf (stderr, "optimize_MonteCarlo: end\n");
00748 #endif
00749 }
```

4.19.2.15 optimize_norm_euclidian()

```
double optimize_norm_euclidian (
             unsigned int simulation )
```

Function to calculate the Euclidian error norm.

Returns

Euclidian error norm.

Parameters

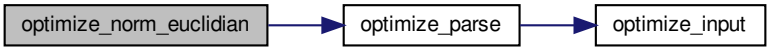
simulation	simulation number.

Definition at line 292 of file optimize.c.

```
00293 {
00294   double e, ei;
00295   unsigned int i;
00296 #if DEBUG_OPTIMIZE
00297   fprintf (stderr, "optimize_norm_euclidian: start\n");
00298 #endif
00299   e = 0.;
00300   for (i = 0; i < optimize->nexperiments; ++i)
00301     {
00302       ei = optimize_parse (simulation, i);
00303       e += ei * ei;
00304     }
00305   e = sqrt (e);
00306 #if DEBUG_OPTIMIZE
```

```
00307   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00308   fprintf (stderr, "optimize_norm_euclidian: end\n");
00309 #endif
00310   return e;
00311 }
```

Here is the call graph for this function:



4.19.2.16 optimize_norm_maximum()

```
double optimize_norm_maximum (
              unsigned int simulation )
```

Function to calculate the maximum error norm.

Returns

Maximum error norm.

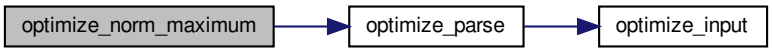
Parameters

simulation	simulation number.

Definition at line 319 of file optimize.c.

```
00320 {
00321   double e, ei;
00322   unsigned int i;
00323 #if DEBUG_OPTIMIZE
00324   fprintf (stderr, "optimize_norm_maximum: start\n");
00325 #endif
00326   e = 0.;
00327   for (i = 0; i < optimize->nexperiments; ++i)
00328     {
00329       ei = fabs (optimize_parse (simulation, i));
00330       e = fmax (e, ei);
00331     }
00332 #if DEBUG_OPTIMIZE
00333   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00334   fprintf (stderr, "optimize_norm_maximum: end\n");
00335 #endif
00336   return e;
00337 }
```

Here is the call graph for this function:



4.19.2.17 optimize_norm_p()

```
double optimize_norm_p (
            unsigned int simulation )
```

Function to calculate the P error norm.

Returns

P error norm.

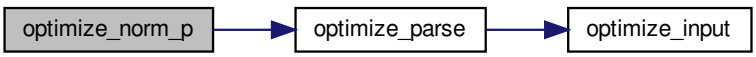
Parameters

simulation	simulation number.

Definition at line 345 of file optimize.c.

```
00346 {
00347   double e, ei;
00348   unsigned int i;
00349 #if DEBUG_OPTIMIZE
00350   fprintf (stderr, "optimize_norm_p: start\n");
00351 #endif
00352   e = 0.;
00353   for (i = 0; i < optimize->nexperiments; ++i)
00354     {
00355       ei = fabs (optimize_parse (simulation, i));
00356       e += pow (ei, optimize->p);
00357     }
00358   e = pow (e, 1. / optimize->p);
00359 #if DEBUG_OPTIMIZE
00360   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00361   fprintf (stderr, "optimize_norm_p: end\n");
00362 #endif
00363   return e;
00364 }
```

Here is the call graph for this function:



4.19.2.18 optimize_norm_taxicab()

```
double optimize_norm_taxicab (
             unsigned int simulation )
```

Function to calculate the taxicab error norm.

Returns

Taxicab error norm.

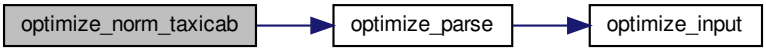
Parameters

simulation	simulation number.

Definition at line 372 of file optimize.c.

```
00373 {
00374   double e;
00375   unsigned int i;
00376 #if DEBUG_OPTIMIZE
00377   fprintf (stderr, "optimize_norm_taxicab: start\n");
00378 #endif
00379   e = 0.;
00380   for (i = 0; i < optimize->nexperiments; ++i)
00381     e += fabs (optimize_parse (simulation, i));
00382 #if DEBUG_OPTIMIZE
00383   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00384   fprintf (stderr, "optimize_norm_taxicab: end\n");
00385 #endif
00386   return e;
00387 }
```

Here is the call graph for this function:



4.19.2.19 optimize_open()

```
void optimize_open ( )
```

Function to open and perform a optimization.

Definition at line 1418 of file optimize.c.

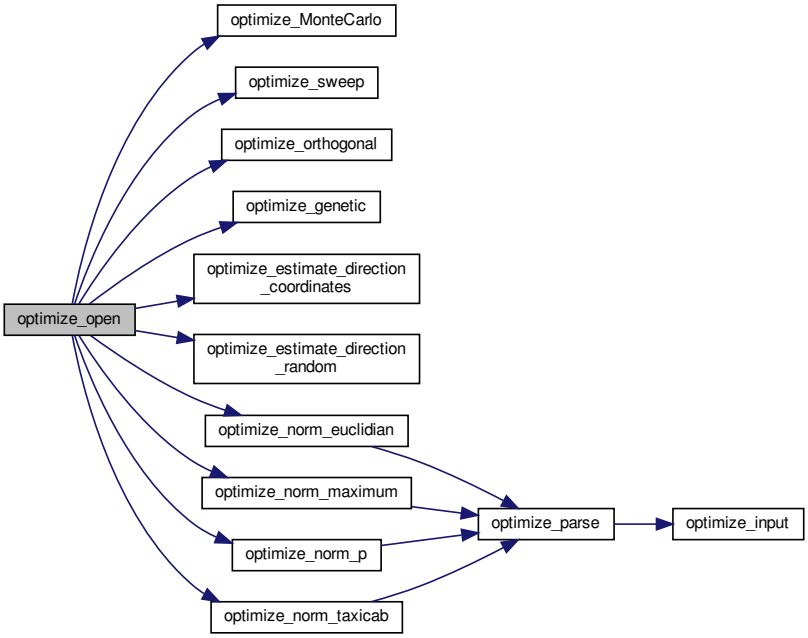
```
01419 {
01420   GTimeZone *tz;
01421   GDateTime *t0, *t;
01422   unsigned int i, j;
01423 
01424 #if DEBUG_OPTIMIZE
01425   char *buffer;
01426   fprintf (stderr, "optimize_open: start\n");
01427 #endif
01428 
01429   // Getting initial time
01430 #if DEBUG_OPTIMIZE
01431   fprintf (stderr, "optimize_open: getting initial time\n");
01432 #endif
01433   tz = g_time_zone_new_utc ();
01434   t0 = g_date_time_new_now (tz);
01435 
01436   // Obtaining and initing the pseudo-random numbers generator seed
01437 #if DEBUG_OPTIMIZE
01438   fprintf (stderr, "optimize_open: getting initial seed\n");
01439 #endif
01440   if (optimize->seed == DEFAULT_RANDOM_SEED)
01441     optimize->seed = input->seed;
01442   gsl_rng_set (optimize->rng, optimize->seed);
01443 
01444   // Replacing the working directory
01445 #if DEBUG_OPTIMIZE
01446   fprintf (stderr, "optimize_open: replacing the working directory\n");
01447 #endif
01448   g_chdir (input->directory);
01449 
01450   // Getting results file names
01451   optimize->result = input->result;
01452   optimize->variables = input->variables;
01453 
01454   // Obtaining the simulator file
01455   optimize->simulator = input->simulator;
01456 
01457   // Obtaining the evaluator file
01458   optimize->evaluator = input->evaluator;
01459 
01460   // Reading the algorithm
01461   optimize->algorithm = input->algorithm;
01462   switch (optimize->algorithm)
01463     {
01464     case ALGORITHM_MONTE_CARLO:
01465       optimize_algorithm = optimize_MonteCarlo;
01466       break;
01467     case ALGORITHM_SWEEP:
01468       optimize_algorithm = optimize_sweep;
01469       break;
01470     case ALGORITHM_ORTHOGONAL:
01471       optimize_algorithm = optimize_orthogonal;
01472       break;
01473     default:
01474       optimize_algorithm = optimize_genetic;
01475       optimize->mutation_ratio = input->
      mutation_ratio;
01476       optimize->reproduction_ratio = input->
      reproduction_ratio;
01477       optimize->adaptation_ratio = input->
      adaptation_ratio;
01478     }
01479   optimize->nvariables = input->nvariables;
01480   optimize->nsimulations = input->nsimulations;
01481   optimize->niterations = input->niterations;
01482   optimize->nbest = input->nbest;
01483   optimize->tolerance = input->tolerance;
01484   optimize->nsteps = input->nsteps;
01485   optimize->nestimates = 0;
01486   optimize->threshold = input->threshold;
01487   optimize->stop = 0;
01488   if (input->nsteps)
01489     {
01490       optimize->relaxation = input->relaxation;
01491       switch (input->direction)
01492         {
01493         case DIRECTION_METHOD_COORDINATES:
01494           optimize->nestimates = 2 * optimize->
      nvariables;
01495           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
01496           break;
01497         default:
01498           optimize->nestimates = input->nestimates;
01499           optimize_estimate_direction =
      optimize_estimate_direction_random;
```

```
01500         }
01501     }
01502
01503 #if DEBUG_OPTIMIZE
01504   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01505 #endif
01506   optimize->simulation_best
01507     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01508   optimize->error_best = (double *) alloca (optimize->
      nbest * sizeof (double));
01509
01510   // Reading the experimental data
01511 #if DEBUG_OPTIMIZE
01512   buffer = g_get_current_dir ();
01513   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01514   g_free (buffer);
01515 #endif
01516   optimize->nexperiments = input->nexperiments;
01517   optimize->ninputs = input->experiment->ninputs;
01518   optimize->experiment
01519     = (char **) alloca (input->nexperiments * sizeof (char *));
01520   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double
      ));
01521   for (i = 0; i < input->experiment->ninputs; ++i)
01522     optimize->file[i] = (GMappedFile **)
01523       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01524   for (i = 0; i < input->nexperiments; ++i)
01525     {
01526 #if DEBUG_OPTIMIZE
01527       fprintf (stderr, "optimize_open: i=%u\n", i);
01528 #endif
01529       optimize->experiment[i] = input->experiment[i].
      name;
01530       optimize->weight[i] = input->experiment[i].
      weight;
01531 #if DEBUG_OPTIMIZE
01532       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01533                optimize->experiment[i], optimize->
      weight[i]);
01534 #endif
01535       for (j = 0; j < input->experiment->ninputs; ++j)
01536         {
01537 #if DEBUG_OPTIMIZE
01538           fprintf (stderr, "optimize_open: stencil%u\n", j + 1);
01539 #endif
01540           optimize->file[j][i]
01541             = g_mapped_file_new (input->experiment[i].stencil[j], 0, NULL);
01542         }
01543     }
01544
01545   // Reading the variables data
01546 #if DEBUG_OPTIMIZE
01547   fprintf (stderr, "optimize_open: reading variables\n");
01548 #endif
01549   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01550   j = input->nvariables * sizeof (double);
01551   optimize->rangemin = (double *) alloca (j);
01552   optimize->rangeminabs = (double *) alloca (j);
01553   optimize->rangemax = (double *) alloca (j);
01554   optimize->rangemaxabs = (double *) alloca (j);
01555   optimize->step = (double *) alloca (j);
01556   j = input->nvariables * sizeof (unsigned int);
01557   optimize->precision = (unsigned int *) alloca (j);
01558   optimize->nsweeps = (unsigned int *) alloca (j);
01559   optimize->nbits = (unsigned int *) alloca (j);
01560   for (i = 0; i < input->nvariables; ++i)
01561     {
01562       optimize->label[i] = input->variable[i].name;
01563       optimize->rangemin[i] = input->variable[i].
      rangemin;
01564       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01565       optimize->rangemax[i] = input->variable[i].
      rangemax;
01566       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01567       optimize->precision[i] = input->variable[i].
      precision;
01568       optimize->step[i] = input->variable[i].step;
01569       optimize->nsweeps[i] = input->variable[i].
      nsweeps;
01570       optimize->nbits[i] = input->variable[i].nbits;
01571     }
01572   if (input->algorithm == ALGORITHM_SWEEP
01573       || input->algorithm == ALGORITHM_ORTHOGONAL)
01574     {
01575       optimize->nsimulations = 1;
```

```
01576       for (i = 0; i < input->nvariables; ++i)
01577         {
01578           optimize->nsimulations *= optimize->
      nsweeps[i];
01579 #if DEBUG_OPTIMIZE
01580           fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01581                    optimize->nsweeps[i], optimize->
      nsimulations);
01582 #endif
01583         }
01584     }
01585   if (optimize->nsteps)
01586     optimize->direction
01587       = (double *) alloca (optimize->nvariables * sizeof (double));
01588
01589   // Setting error norm
01590   switch (input->norm)
01591     {
01592     case ERROR_NORM_EUCLIDIAN:
01593       optimize_norm = optimize_norm_euclidian;
01594       break;
01595     case ERROR_NORM_MAXIMUM:
01596       optimize_norm = optimize_norm_maximum;
01597       break;
01598     case ERROR_NORM_P:
01599       optimize_norm = optimize_norm_p;
01600       optimize->p = input->p;
01601       break;
01602     default:
01603       optimize_norm = optimize_norm_taxicab;
01604     }
01605
01606   // Allocating values
01607 #if DEBUG_OPTIMIZE
01608   fprintf (stderr, "optimize_open: allocating variables\n");
01609   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01610            optimize->nvariables, optimize->algorithm);
01611 #endif
01612   optimize->genetic_variable = NULL;
01613   if (optimize->algorithm == ALGORITHM_GENETIC)
01614     {
01615       optimize->genetic_variable = (GeneticVariable *)
01616         g_malloc (optimize->nvariables * sizeof (
      GeneticVariable));
01617       for (i = 0; i < optimize->nvariables; ++i)
01618         {
01619 #if DEBUG_OPTIMIZE
01620           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01621                    i, optimize->rangemin[i], optimize->
      rangemax[i],
01622                    optimize->nbits[i]);
01623 #endif
01624           optimize->genetic_variable[i].minimum =
      optimize->rangemin[i];
01625           optimize->genetic_variable[i].maximum =
      optimize->rangemax[i];
01626           optimize->genetic_variable[i].nbits = optimize->
      nbits[i];
01627         }
01628     }
01629 #if DEBUG_OPTIMIZE
01630   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01631            optimize->nvariables, optimize->
      nsimulations);
01632 #endif
01633   optimize->value = (double *)
01634     g_malloc ((optimize->nsimulations
01635                + optimize->nestimates * optimize->
      nsteps)
01636               * optimize->nvariables * sizeof (double));
01637
01638   // Calculating simulations to perform for each task
01639 #if HAVE_MPI
01640 #if DEBUG_OPTIMIZE
01641   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01642            optimize->mpi_rank, ntasks);
01643 #endif
01644   optimize->nstart = optimize->mpi_rank * optimize->
      nsimulations / ntasks;
01645   optimize->nend = (1 + optimize->mpi_rank) *
      optimize->nsimulations / ntasks;
01646   if (optimize->nsteps)
01647     {
01648       optimize->nstart_direction
01649         = optimize->mpi_rank * optimize->nestimates /
      ntasks;
01650       optimize->nend_direction
```

```
01651         = (1 + optimize->mpi_rank) * optimize->
      nestimates / ntasks;
01652     }
01653 #else
01654   optimize->nstart = 0;
01655   optimize->nend = optimize->nsimulations;
01656   if (optimize->nsteps)
01657     {
01658       optimize->nstart_direction = 0;
01659       optimize->nend_direction = optimize->
      nestimates;
01660     }
01661 #endif
01662 #if DEBUG_OPTIMIZE
01663   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->
      nstart,
01664            optimize->nend);
01665 #endif
01666
01667   // Calculating simulations to perform for each thread
01668   optimize->thread
01669     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01670   for (i = 0; i <= nthreads; ++i)
01671     {
01672       optimize->thread[i] = optimize->nstart
01673         + i * (optimize->nend - optimize->nstart) / nthreads;
01674 #if DEBUG_OPTIMIZE
01675       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01676                optimize->thread[i]);
01677 #endif
01678     }
01679   if (optimize->nsteps)
01680     optimize->thread_direction = (unsigned int *)
01681       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01682
01683   // Opening result files
01684   optimize->file_result = g_fopen (optimize->result, "w");
01685   optimize->file_variables = g_fopen (optimize->
      variables, "w");
01686
01687   // Performing the algorithm
01688   switch (optimize->algorithm)
01689     {
01690       // Genetic algorithm
01691     case ALGORITHM_GENETIC:
01692       optimize_genetic ();
01693       break;
01694
01695       // Iterative algorithm
01696     default:
01697       optimize_iterate ();
01698     }
01699
01700   // Getting calculation time
01701   t = g_date_time_new_now (tz);
01702   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01703   g_date_time_unref (t);
01704   g_date_time_unref (t0);
01705   g_time_zone_unref (tz);
01706   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->
      calculation_time);
01707   fprintf (optimize->file_result, "%s = %.6lg s\n",
01708            _("Calculation time"), optimize->calculation_time);
01709
01710   // Closing result files
01711   fclose (optimize->file_variables);
01712   fclose (optimize->file_result);
01713
01714 #if DEBUG_OPTIMIZE
01715   fprintf (stderr, "optimize_open: end\n");
01716 #endif
01717 }
```

Here is the call graph for this function:



4.19.2.20 optimize_orthogonal()

```
void optimize_orthogonal ( )
```

Function to optimize with the orthogonal sampling algorithm.

Definition at line 755 of file optimize.c.

```
00756 {
00757   unsigned int i, j, k, l;
00758   double e;
00759   GThread *thread[nthreads];
00760   ParallelData data[nthreads];
00761 #if DEBUG_OPTIMIZE
00762   fprintf (stderr, "optimize_orthogonal: start\n");
00763 #endif
00764   for (i = 0; i < optimize->nsimulations; ++i)
00765     {
00766       k = i;
00767       for (j = 0; j < optimize->nvariables; ++j)
00768         {
00769           l = k % optimize->nsweeps[j];
00770           k /= optimize->nsweeps[j];
00771           e = optimize->rangemin[j];
00772           if (optimize->nsweeps[j] > 1)
00773             e += (l + gsl_rng_uniform (optimize->rng))
00774                                * (optimize->rangemax[j] - optimize->
      rangemin[j])
00775                / optimize->nsweeps[j];
00776           optimize->value[i * optimize->nvariables + j] = e;
00777         }
00778     }
00779   optimize->nsaveds = 0;
00780   if (nthreads <= 1)
```

```
00781     optimize_sequential ();
00782   else
00783     {
00784       for (i = 0; i < nthreads; ++i)
00785         {
00786           data[i].thread = i;
00787           thread[i]
00788             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00789         }
00790       for (i = 0; i < nthreads; ++i)
00791         g_thread_join (thread[i]);
00792     }
00793 #if HAVE_MPI
00794   // Communicating tasks results
00795   optimize_synchronise ();
00796 #endif
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_orthogonal: end\n");
00799 #endif
00800 }
```

4.19.2.21 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.

```
00184 {
00185   unsigned int i;
00186   double e;
00187   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00188     *buffer3, *buffer4;
00189   FILE *file_result;
00190
00191 #if DEBUG_OPTIMIZE
00192   fprintf (stderr, "optimize_parse: start\n");
00193   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00194            simulation, experiment);
00195 #endif
00196
00197   // Opening input files
00198   for (i = 0; i < optimize->ninputs; ++i)
00199     {
00200       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00201 #if DEBUG_OPTIMIZE
00202       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00203 #endif
00204       optimize_input (simulation, &input[i][0], optimize->
     file[i][experiment]);
00205     }
00206   for (; i < MAX_NINPUTS; ++i)
00207     strcpy (&input[i][0], "");
```

```
00208 #if DEBUG_OPTIMIZE
00209   fprintf (stderr, "optimize_parse: parsing end\n");
00210 #endif
00211
00212   // Performing the simulation
00213   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00214   buffer2 = g_path_get_dirname (optimize->simulator);
00215   buffer3 = g_path_get_basename (optimize->simulator);
00216   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00217   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s",
00218             buffer4, input[0], input[1], input[2], input[3], input[4],
00219             input[5], input[6], input[7], output);
00220   g_free (buffer4);
00221   g_free (buffer3);
00222   g_free (buffer2);
00223 #if DEBUG_OPTIMIZE
00224   fprintf (stderr, "optimize_parse: %s\n", buffer);
00225 #endif
00226   system (buffer);
00227
00228   // Checking the objective value function
00229   if (optimize->evaluator)
00230     {
00231       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00232       buffer2 = g_path_get_dirname (optimize->evaluator);
00233       buffer3 = g_path_get_basename (optimize->evaluator);
00234       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00235       snprintf (buffer, 512, "\"%s\" %s %s %s",
00236                 buffer4, output, optimize->experiment[experiment], result);
00237       g_free (buffer4);
00238       g_free (buffer3);
00239       g_free (buffer2);
00240 #if DEBUG_OPTIMIZE
00241       fprintf (stderr, "optimize_parse: %s\n", buffer);
00242       fprintf (stderr, "optimize_parse: result=%s\n", result);
00243 #endif
00244       system (buffer);
00245       file_result = g_fopen (result, "r");
00246       e = atof (fgets (buffer, 512, file_result));
00247       fclose (file_result);
00248     }
00249   else
00250     {
00251 #if DEBUG_OPTIMIZE
00252       fprintf (stderr, "optimize_parse: output=%s\n", output);
00253 #endif
00254       strcpy (result, "");
00255       file_result = g_fopen (output, "r");
00256       e = atof (fgets (buffer, 512, file_result));
00257       fclose (file_result);
00258     }
00259
00260   // Removing files
00261 #if !DEBUG_OPTIMIZE
00262   for (i = 0; i < optimize->ninputs; ++i)
00263     {
00264       if (optimize->file[i][0])
00265         {
00266           snprintf (buffer, 512, RM " %s", &input[i][0]);
00267           system (buffer);
00268         }
00269     }
00270   snprintf (buffer, 512, RM " %s %s", output, result);
00271   system (buffer);
00272 #endif
00273
00274   // Processing pending events
00275   if (show_pending)
00276     show_pending ();
00277
00278 #if DEBUG_OPTIMIZE
00279   fprintf (stderr, "optimize_parse: end\n");
00280 #endif
00281
00282   // Returning the objective function
00283   return e * optimize->weight[experiment];
00284 }
```

Here is the call graph for this function:

4.19.2.22 optimize_print()

```
void optimize_print ( )
```

Function to print the results.

Definition at line 393 of file optimize.c.

```
00394 {
00395   unsigned int i;
00396   char buffer[512];
00397 #if HAVE_MPI
00398   if (optimize->mpi_rank)
00399     return;
00400 #endif
00401   printf ("%s\n", _("Best result"));
00402   fprintf (optimize->file_result, "%s\n", _("Best result"));
00403   printf ("error = %.15le\n", optimize->error_old[0]);
00404   fprintf (optimize->file_result, "error = %.15le\n",
      optimize->error_old[0]);
00405   for (i = 0; i < optimize->nvariables; ++i)
00406     {
00407       snprintf (buffer, 512, "%s = %s\n",
00408                 optimize->label[i], format[optimize->
      precision[i]]);
00409       printf (buffer, optimize->value_old[i]);
00410       fprintf (optimize->file_result, buffer, optimize->
      value_old[i]);
00411     }
00412   fflush (optimize->file_result);
00413 }
```

4.19.2.23 optimize_refine()

```
void optimize_refine ( )
```

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

Definition at line 1262 of file optimize.c.

```
01263 {
01264   unsigned int i, j;
01265   double d;
01266 #if HAVE_MPI
01267   MPI_Status mpi_stat;
01268 #endif
01269 #if DEBUG_OPTIMIZE
01270   fprintf (stderr, "optimize_refine: start\n");
01271 #endif
01272 #if HAVE_MPI
01273   if (!optimize->mpi_rank)
01274     {
01275 #endif
01276       for (j = 0; j < optimize->nvariables; ++j)
01277         {
01278           optimize->rangemin[j] = optimize->rangemax[j]
01279             = optimize->value_old[j];
01280         }
01281       for (i = 0; ++i < optimize->nbest;)
01282         {
01283           for (j = 0; j < optimize->nvariables; ++j)
01284             {
01285               optimize->rangemin[j]
01286                 = fmin (optimize->rangemin[j],
01287                         optimize->value_old[i * optimize->
      nvariables + j]);
01288               optimize->rangemax[j]
01289                 = fmax (optimize->rangemax[j],
01290                         optimize->value_old[i * optimize->
      nvariables + j]);
01291             }
01292         }
01293       for (j = 0; j < optimize->nvariables; ++j)
01294         {
01295           d = optimize->tolerance
01296             * (optimize->rangemax[j] - optimize->
      rangemin[j]);
01297           switch (optimize->algorithm)
01298             {
01299             case ALGORITHM_MONTE_CARLO:
01300               d *= 0.5;
01301               break;
01302             default:
01303               if (optimize->nsweeps[j] > 1)
01304                 d /= optimize->nsweeps[j] - 1;
01305               else
01306                 d = 0.;
01307             }
01308           optimize->rangemin[j] -= d;
01309           optimize->rangemin[j]
01310             = fmax (optimize->rangemin[j], optimize->
      rangeminabs[j]);
01311           optimize->rangemax[j] += d;
01312           optimize->rangemax[j]
01313             = fmin (optimize->rangemax[j], optimize->
      rangemaxabs[j]);
01314           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01315                   optimize->rangemin[j], optimize->
      rangemax[j]);
01316           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01317                    optimize->label[j], optimize->rangemin[j],
01318                    optimize->rangemax[j]);
01319         }
01320 #if HAVE_MPI
01321       for (i = 1; i < ntasks; ++i)
01322         {
01323           MPI_Send (optimize->rangemin, optimize->
      nvariables, MPI_DOUBLE, i,
01324                     1, MPI_COMM_WORLD);
01325           MPI_Send (optimize->rangemax, optimize->
      nvariables, MPI_DOUBLE, i,
01326                     1, MPI_COMM_WORLD);
01327         }
01328     }
01329   else
01330     {
01331       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0,
      1,
01332                 MPI_COMM_WORLD, &mpi_stat);
01333       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0,
      1,
01334                 MPI_COMM_WORLD, &mpi_stat);
01335     }
01336 #endif
01337 #if DEBUG_OPTIMIZE
01338   fprintf (stderr, "optimize_refine: end\n");
01339 #endif
```

```
01340 }
```

4.19.2.24 optimize_save_old()

```
void optimize_save_old ( )
```

Function to save the best results on iterative methods.

Definition at line 1182 of file optimize.c.

```
01183 {
01184   unsigned int i, j;
01185 #if DEBUG_OPTIMIZE
01186   fprintf (stderr, "optimize_save_old: start\n");
01187   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01188 #endif
01189   memcpy (optimize->error_old, optimize->error_best,
01190           optimize->nbest * sizeof (double));
01191   for (i = 0; i < optimize->nbest; ++i)
01192     {
01193       j = optimize->simulation_best[i];
01194 #if DEBUG_OPTIMIZE
01195       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01196 #endif
01197       memcpy (optimize->value_old + i * optimize->
      nvariables,
01198               optimize->value + j * optimize->nvariables,
01199               optimize->nvariables * sizeof (double));
01200     }
01201 #if DEBUG_OPTIMIZE
01202   for (i = 0; i < optimize->nvariables; ++i)
01203     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01204              i, optimize->value_old[i]);
01205   fprintf (stderr, "optimize_save_old: end\n");
01206 #endif
01207 }
```

4.19.2.25 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.

```
00421 {
00422   unsigned int i;
00423   char buffer[64];
```

```
00424 #if DEBUG_OPTIMIZE
00425   fprintf (stderr, "optimize_save_variables: start\n");
00426 #endif
00427   for (i = 0; i < optimize->nvariables; ++i)
00428     {
00429       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00430       fprintf (optimize->file_variables, buffer,
00431                optimize->value[simulation * optimize->
      nvariables + i]);
00432     }
00433   fprintf (optimize->file_variables, "%.14le\n", error);
00434   fflush (optimize->file_variables);
00435 #if DEBUG_OPTIMIZE
00436   fprintf (stderr, "optimize_save_variables: end\n");
00437 #endif
00438 }
```

4.19.2.26 optimize_sequential()

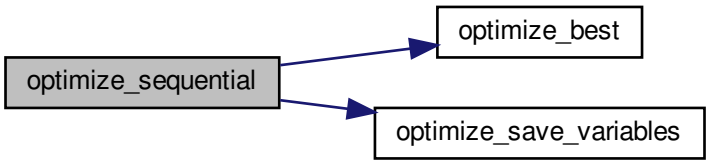
```
void optimize_sequential ( )
```

Function to optimize sequentially.

Definition at line 485 of file optimize.c.

```
00486 {
00487   unsigned int i;
00488   double e;
00489 #if DEBUG_OPTIMIZE
00490   fprintf (stderr, "optimize_sequential: start\n");
00491   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00492            optimize->nstart, optimize->nend);
00493 #endif
00494   for (i = optimize->nstart; i < optimize->nend; ++i)
00495     {
00496       e = optimize_norm (i);
00497       optimize_best (i, e);
00498       optimize_save_variables (i, e);
00499       if (e < optimize->threshold)
00500         {
00501           optimize->stop = 1;
00502           break;
00503         }
00504 #if DEBUG_OPTIMIZE
00505       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00506 #endif
00507     }
00508 #if DEBUG_OPTIMIZE
00509   fprintf (stderr, "optimize_sequential: end\n");
00510 #endif
00511 }
```

Here is the call graph for this function:



4.19.2.27 optimize_step()

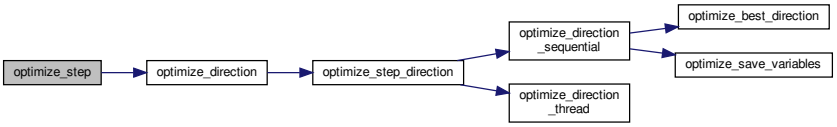
```
void optimize_step ( )
```

Function to do a step of the iterative algorithm.

Definition at line 1346 of file optimize.c.

```
01347 {
01348 #if DEBUG_OPTIMIZE
01349   fprintf (stderr, "optimize_step: start\n");
01350 #endif
01351   optimize_algorithm ();
01352   if (optimize->nsteps)
01353     optimize_direction ();
01354 #if DEBUG_OPTIMIZE
01355   fprintf (stderr, "optimize_step: end\n");
01356 #endif
01357 }
```

Here is the call graph for this function:



4.19.2.28 optimize_step_direction()

```
void optimize_step_direction (
             unsigned int simulation )
```

Function to do a step of the direction search method.

Parameters

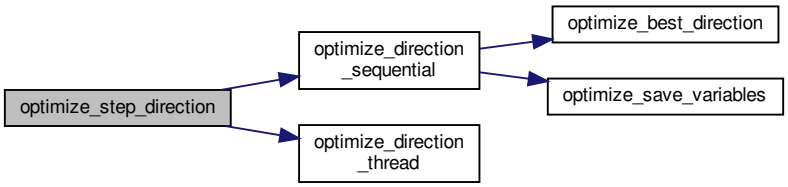
simulation	Simulation number.

Definition at line 967 of file optimize.c.

```
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->
      nvariables;
00979 #if DEBUG_OPTIMIZE
```

```
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011               / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
01022             (NULL, (GThreadFunc) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
```

Here is the call graph for this function:



4.19.2.29 optimize_sweep()

```
void optimize_sweep ( )
```

Function to optimize with the sweep algorithm.

Definition at line 665 of file optimize.c.

```
00666 {
00667   unsigned int i, j, k, l;
00668   double e;
00669   GThread *thread[nthreads];
00670   ParallelData data[nthreads];
00671 #if DEBUG_OPTIMIZE
00672   fprintf (stderr, "optimize_sweep: start\n");
00673 #endif
00674   for (i = 0; i < optimize->nsimulations; ++i)
00675     {
00676       k = i;
00677       for (j = 0; j < optimize->nvariables; ++j)
00678         {
00679           l = k % optimize->nsweeps[j];
00680           k /= optimize->nsweeps[j];
00681           e = optimize->rangemin[j];
00682           if (optimize->nsweeps[j] > 1)
00683             e += l * (optimize->rangemax[j] - optimize->
      rangemin[j])
00684                / (optimize->nsweeps[j] - 1);
00685           optimize->value[i * optimize->nvariables + j] = e;
00686         }
00687     }
00688   optimize->nsaveds = 0;
00689   if (nthreads <= 1)
00690     optimize_sequential ();
00691   else
00692     {
00693       for (i = 0; i < nthreads; ++i)
00694         {
00695           data[i].thread = i;
00696           thread[i]
00697             = g_thread_new (NULL, (GThreadFunc) optimize_thread, &data[i]);
00698         }
00699       for (i = 0; i < nthreads; ++i)
00700         g_thread_join (thread[i]);
00701     }
00702 #if HAVE_MPI
00703   // Communicating tasks results
00704   optimize_synchronise ();
00705 #endif
00706 #if DEBUG_OPTIMIZE
00707   fprintf (stderr, "optimize_sweep: end\n");
00708 #endif
00709 }
```

4.19.2.30 optimize_synchronise()

```
void optimize_synchronise ( )
```

Function to synchronise the optimization results of MPI tasks.

Definition at line 618 of file optimize.c.

```
00619 {
00620   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00621   double error_best[optimize->nbest];
00622   MPI_Status mpi_stat;
00623 #if DEBUG_OPTIMIZE
00624   fprintf (stderr, "optimize_synchronise: start\n");
00625 #endif
00626   if (optimize->mpi_rank == 0)
00627     {
00628       for (i = 1; i < ntasks; ++i)
00629         {
00630           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00631           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00632                     MPI_COMM_WORLD, &mpi_stat);
```

```
00633           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00634                     MPI_COMM_WORLD, &mpi_stat);
00635           optimize_merge (nsaveds, simulation_best, error_best);
00636           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00637           if (stop)
00638             optimize->stop = 1;
00639         }
00640       for (i = 1; i < ntasks; ++i)
00641         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00642     }
00643   else
00644     {
00645       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00646       MPI_Send (optimize->simulation_best, optimize->
      nsaveds, MPI_INT, 0, 1,
00647                 MPI_COMM_WORLD);
00648       MPI_Send (optimize->error_best, optimize->
      nsaveds, MPI_DOUBLE, 0, 1,
00649                 MPI_COMM_WORLD);
00650       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00651       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00652       if (stop)
00653         optimize->stop = 1;
00654     }
00655 #if DEBUG_OPTIMIZE
00656   fprintf (stderr, "optimize_synchronise: end\n");
00657 #endif
00658 }
```

4.19.2.31 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.

```
00520 {
00521   unsigned int i, thread;
00522   double e;
00523 #if DEBUG_OPTIMIZE
00524   fprintf (stderr, "optimize_thread: start\n");
00525 #endif
00526   thread = data->thread;
00527 #if DEBUG_OPTIMIZE
00528   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00529            optimize->thread[thread], optimize->thread[thread + 1]);
00530 #endif
00531   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00532     {
00533       e = optimize_norm (i);
00534       g_mutex_lock (mutex);
00535       optimize_best (i, e);
00536       optimize_save_variables (i, e);
00537       if (e < optimize->threshold)
00538         optimize->stop = 1;
00539       g_mutex_unlock (mutex);
```

```
00540       if (optimize->stop)
00541         break;
00542 #if DEBUG_OPTIMIZE
00543       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00544 #endif
00545     }
00546 #if DEBUG_OPTIMIZE
00547   fprintf (stderr, "optimize_thread: end\n");
00548 #endif
00549   g_thread_exit (NULL);
00550   return NULL;
00551 }
```

4.20 optimize.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef OPTIMIZE__H
00039 #define OPTIMIZE__H 1
00040
00045 typedef struct
00046 {
00047   GMappedFile **file[MAX_NINPUTS];
00048   char **experiment;
00049   char **label;
00050   gsl_rng *rng;
00051   GeneticVariable *genetic_variable;
00053   FILE *file_result;
00054   FILE *file_variables;
00055   char *result;
00056   char *variables;
00057   char *simulator;
00058   char *evaluator;
00060   double *value;
00061   double *rangemin;
00062   double *rangemax;
00063   double *rangeminabs;
00064   double *rangemaxabs;
00065   double *error_best;
00066   double *weight;
00067   double *step;
00069   double *direction;
00070   double *value_old;
00072   double *error_old;
00074   unsigned int *precision;
00075   unsigned int *nsweeps;
00076   unsigned int *nbits;
00078   unsigned int *thread;
00080   unsigned int *thread_direction;
00083   unsigned int *simulation_best;
00084   double tolerance;
00085   double mutation_ratio;
00086   double reproduction_ratio;
```

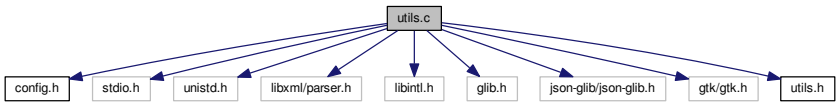
```
00087   double adaptation_ratio;
00088   double relaxation;
00089   double calculation_time;
00090   double p;
00091   double threshold;
00092   unsigned long int seed;
00094   unsigned int nvariables;
00095   unsigned int nexperiments;
00096   unsigned int ninputs;
00097   unsigned int nsimulations;
00098   unsigned int nsteps;
00100   unsigned int nestimates;
00102   unsigned int algorithm;
00103   unsigned int nstart;
00104   unsigned int nend;
00105   unsigned int nstart_direction;
00107   unsigned int nend_direction;
00109   unsigned int niterations;
00110   unsigned int nbest;
00111   unsigned int nsaveds;
00112   unsigned int stop;
00113 #if HAVE_MPI
00114   int mpi_rank;
00115 #endif
00116 } Optimize;
00117
00122 typedef struct
00123 {
00124   unsigned int thread;
00125 } ParallelData;
00126
00127 // Global variables
00128 extern int ntasks;
00129 extern unsigned int nthreads;
00130 extern unsigned int nthreads_direction;
00131 extern GMutex mutex[1];
00132 extern void (*optimize_algorithm) ();
00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134                                               unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
00137
00138 // Public functions
00139 void optimize_input (unsigned int simulation, char *input,
00140                      GMappedFile * stencil);
00141 double optimize_parse (unsigned int simulation, unsigned int experiment);
00142 double optimize_norm_euclidian (unsigned int simulation);
00143 double optimize_norm_maximum (unsigned int simulation);
00144 double optimize_norm_p (unsigned int simulation);
00145 double optimize_norm_taxicab (unsigned int simulation);
00146 void optimize_print ();
00147 void optimize_save_variables (unsigned int simulation, double error);
00148 void optimize_best (unsigned int simulation, double value);
00149 void optimize_sequential ();
00150 void *optimize_thread (ParallelData * data);
00151 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00152                      double *error_best);
00153 #if HAVE_MPI
00154 void optimize_synchronise ();
00155 #endif
00156 void optimize_sweep ();
00157 void optimize_MonteCarlo ();
00158 void optimize_orthogonal ();
00159 void optimize_best_direction (unsigned int simulation, double value);
00160 void optimize_direction_sequential (unsigned int simulation);
00161 void *optimize_direction_thread (ParallelData * data);
00162 double optimize_estimate_direction_random (unsigned int variable,
00163                                            unsigned int estimate);
00164 double optimize_estimate_direction_coordinates (unsigned int
      variable,
00165                                                 unsigned int estimate);
00166 void optimize_step_direction (unsigned int simulation);
00167 void optimize_direction ();
00168 double optimize_genetic_objective (Entity * entity);
00169 void optimize_genetic ();
00170 void optimize_save_old ();
00171 void optimize_merge_old ();
00172 void optimize_refine ();
00173 void optimize_step ();
00174 void optimize_iterate ();
00175 void optimize_free ();
00176 void optimize_open ();
00177
00178 #endif
```

4.21 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.

4.21.1 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.

4.21.2 Function Documentation

4.21.2.1 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line 440 of file utils.c.

```
00441 {
00442 #ifdef G_OS_WIN32
00443   SYSTEM_INFO sysinfo;
00444   GetSystemInfo (&sysinfo);
00445   return sysinfo.dwNumberOfProcessors;
00446 #else
00447   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00448 #endif
00449 }
```

4.21.2.2 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.

```
00471 {
00472   unsigned int i;
00473   for (i = 0; i < n; ++i)
00474     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475       break;
00476   return i;
00477 }
```

4.21.2.3 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.

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

4.21.2.4 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.

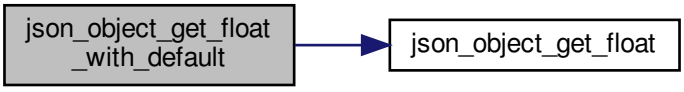
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 376 of file utils.c.

```
00382 {
00383   double x;
00384   if (json_object_get_member (object, prop))
00385     x = json_object_get_float (object, prop, error_code);
00386   else
00387     {
00388       x = default_value;
00389       *error_code = 0;
00390     }
00391   return x;
00392 }
```

Here is the call graph for this function:



4.21.2.5 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.

```
00279 {
00280   const char *buffer;
00281   int i = 0;
00282   buffer = json_object_get_string_member (object, prop);
00283   if (!buffer)
00284     *error_code = 1;
00285   else
00286     {
00287       if (sscanf (buffer, "%d", &i) != 1)
00288         *error_code = 2;
00289       else
00290         *error_code = 0;
00291     }
00292   return i;
00293 }
```

4.21.2.6 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.

```
00304 {
00305   const char *buffer;
00306   unsigned int i = 0;
00307   buffer = json_object_get_string_member (object, prop);
00308   if (!buffer)
```

```
00309     *error_code = 1;
00310   else
00311     {
00312       if (sscanf (buffer, "%u", &i) != 1)
00313         *error_code = 2;
00314       else
00315         *error_code = 0;
00316     }
00317   return i;
00318 }
```

4.21.2.7 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.

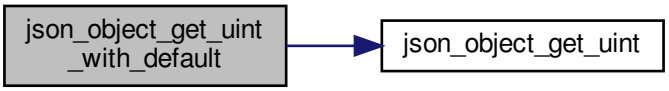
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 327 of file utils.c.

```
00332 {
00333   unsigned int i;
00334   if (json_object_get_member (object, prop))
00335     i = json_object_get_uint (object, prop, error_code);
00336   else
00337     {
00338       i = default_value;
00339       *error_code = 0;
00340     }
00341   return i;
00342 }
```

Here is the call graph for this function:



4.21.2.8 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.

```
00428 {
00429   char buffer[64];
00430   snprintf (buffer, 64, "%.14lg", value);
00431   json_object_set_string_member (object, prop, buffer);
00432 }
```

4.21.2.9 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.

```
00401 {
00402   char buffer[64];
00403   snprintf (buffer, 64, "%d", value);
00404   json_object_set_string_member (object, prop, buffer);
00405 }
```

4.21.2.10 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.

```
00415 {
00416   char buffer[64];
00417   snprintf (buffer, 64, "%u", value);
00418   json_object_set_string_member (object, prop, buffer);
00419 }
```

4.21.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 457 of file utils.c.

```
00458 {
00459   while (gtk_events_pending ())
00460     gtk_main_iteration ();
00461 }
```

4.21.2.12 show_error()

```
void show_error (
            char * msg )
```

Function to show a dialog with an error message.

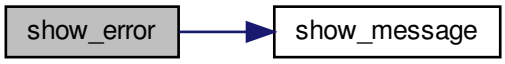
Parameters

msg	Error message.

Definition at line 101 of file utils.c.

```
00102 {
00103   show_message (_("ERROR!"), msg, ERROR_TYPE);
00104 }
```

Here is the call graph for this function:



4.21.2.13 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.

```
00074 {
00075 #if HAVE_GTK
00076   GtkMessageDialog *dlg;
00077
00078   // Creating the dialog
00079   dlg = (GtkMessageDialog *)
00080     gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00081                             (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00082
00083   // Setting the dialog title
00084   gtk_window_set_title (GTK_WINDOW (dlg), title);
00085
00086   // Showing the dialog and waiting response
00087   gtk_dialog_run (GTK_DIALOG (dlg));
00088
00089   // Closing and freeing memory
00090   gtk_widget_destroy (GTK_WIDGET (dlg));
00091
00092 #else
00093   printf ("%s: %s\n", title, msg);
00094 #endif
00095 }
```

4.21.2.14 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.

```
00191 {
00192   double x = 0.;
00193   xmlChar *buffer;
00194   buffer = xmlGetProp (node, prop);
00195   if (!buffer)
00196     *error_code = 1;
00197   else
00198     {
00199       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00200         *error_code = 2;
00201       else
00202         *error_code = 0;
00203       xmlFree (buffer);
00204     }
00205   return x;
00206 }
```

4.21.2.15 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.

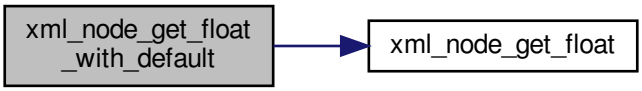
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 215 of file utils.c.

```
00219 {
00220   double x;
00221   if (xmlHasProp (node, prop))
00222     x = xml_node_get_float (node, prop, error_code);
00223   else
00224     {
00225       x = default_value;
00226       *error_code = 0;
00227     }
00228   return x;
00229 }
```

Here is the call graph for this function:



4.21.2.16 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.

```
00115 {
00116   int i = 0;
00117   xmlChar *buffer;
00118   buffer = xmlGetProp (node, prop);
00119   if (!buffer)
00120     *error_code = 1;
00121   else
00122     {
00123       if (sscanf ((char *) buffer, "%d", &i) != 1)
00124         *error_code = 2;
00125       else
00126         *error_code = 0;
00127       xmlFree (buffer);
00128     }
00129   return i;
00130 }
```

4.21.2.17 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.

```
00141 {
00142   unsigned int i = 0;
00143   xmlChar *buffer;
00144   buffer = xmlGetProp (node, prop);
00145   if (!buffer)
00146     *error_code = 1;
00147   else
00148     {
00149       if (sscanf ((char *) buffer, "%u", &i) != 1)
00150         *error_code = 2;
00151       else
00152         *error_code = 0;
00153       xmlFree (buffer);
00154     }
00155   return i;
00156 }
```

4.21.2.18 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.

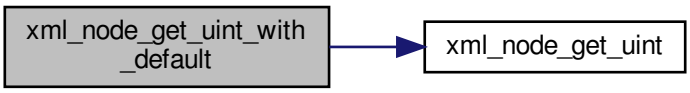
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 165 of file utils.c.

```
00170 {
00171   unsigned int i;
00172   if (xmlHasProp (node, prop))
00173     i = xml_node_get_uint (node, prop, error_code);
00174   else
00175     {
00176       i = default_value;
00177       *error_code = 0;
00178     }
00179   return i;
00180 }
```

Here is the call graph for this function:



4.21.2.19 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.

```
00264 {
00265   xmlChar buffer[64];
00266   snprintf ((char *) buffer, 64, "%.14lg", value);
00267   xmlSetProp (node, prop, buffer);
00268 }
```

4.21.2.20 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.

```
00238 {
00239   xmlChar buffer[64];
00240   snprintf ((char *) buffer, 64, "%d", value);
00241   xmlSetProp (node, prop, buffer);
00242 }
```

4.21.2.21 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.

```
00251 {
00252   xmlChar buffer[64];
00253   snprintf ((char *) buffer, 64, "%u", value);
00254   xmlSetProp (node, prop, buffer);
00255 }
```

4.22 utils.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <unistd.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <json-glib/json-glib.h>
00046 #ifdef G_OS_WIN32
00047 #include <windows.h>
00048 #endif
00049 #if HAVE_GTK
00050 #include <gtk/gtk.h>
00051 #endif
00052 #include "utils.h"
00053
00054 #if HAVE_GTK
00055 GtkWindow *main_window;
00056 #endif
00057
00058 char *error_message;
00059 void (*show_pending) () = NULL;
00061
00065 void
00066 show_message (char *title,
00067               char *msg,
00068               int type
00069 #if !HAVE_GTK
```

```
00070                __attribute__ ((unused))
00071 #endif
00073   )
00074 {
00075 #if HAVE_GTK
00076   GtkMessageDialog *dlg;
00077
00078   // Creating the dialog
00079   dlg = (GtkMessageDialog *)
00080     gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00081                             (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00082
00083   // Setting the dialog title
00084   gtk_window_set_title (GTK_WINDOW (dlg), title);
00085
00086   // Showing the dialog and waiting response
00087   gtk_dialog_run (GTK_DIALOG (dlg));
00088
00089   // Closing and freeing memory
00090   gtk_widget_destroy (GTK_WIDGET (dlg));
00091
00092 #else
00093   printf ("%s: %s\n", title, msg);
00094 #endif
00095 }
00096
00100 void
00101 show_error (char *msg)
00102 {
00103   show_message (_("ERROR!"), msg, ERROR_TYPE);
00104 }
00105
00111 int
00112 xml_node_get_int (xmlNode * node,
00113                   const xmlChar * prop,
00114                   int *error_code)
00115 {
00116   int i = 0;
00117   xmlChar *buffer;
00118   buffer = xmlGetProp (node, prop);
00119   if (!buffer)
00120     *error_code = 1;
00121   else
00122     {
00123       if (sscanf ((char *) buffer, "%d", &i) != 1)
00124         *error_code = 2;
00125       else
00126         *error_code = 0;
00127       xmlFree (buffer);
00128     }
00129   return i;
00130 }
00131
00137 unsigned int
00138 xml_node_get_uint (xmlNode * node,
00139                    const xmlChar * prop,
00140                    int *error_code)
00141 {
00142   unsigned int i = 0;
00143   xmlChar *buffer;
00144   buffer = xmlGetProp (node, prop);
00145   if (!buffer)
00146     *error_code = 1;
00147   else
00148     {
00149       if (sscanf ((char *) buffer, "%u", &i) != 1)
00150         *error_code = 2;
00151       else
00152         *error_code = 0;
00153       xmlFree (buffer);
00154     }
00155   return i;
00156 }
00157
00164 unsigned int
00165 xml_node_get_uint_with_default (xmlNode * node,
00166                                 const xmlChar * prop,
00167                                 unsigned int default_value,
00169                                 int *error_code)
00170 {
00171   unsigned int i;
00172   if (xmlHasProp (node, prop))
00173     i = xml_node_get_uint (node, prop, error_code);
00174   else
00175     {
00176       i = default_value;
00177       *error_code = 0;
```

```
00178     }
00179   return i;
00180 }
00181
00187 double
00188 xml_node_get_float (xmlNode * node,
00189                     const xmlChar * prop,
00190                     int *error_code)
00191 {
00192   double x = 0.;
00193   xmlChar *buffer;
00194   buffer = xmlGetProp (node, prop);
00195   if (!buffer)
00196     *error_code = 1;
00197   else
00198     {
00199       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00200         *error_code = 2;
00201       else
00202         *error_code = 0;
00203       xmlFree (buffer);
00204     }
00205   return x;
00206 }
00207
00214 double
00215 xml_node_get_float_with_default (xmlNode * node,
00216                                  const xmlChar * prop,
00217                                  double default_value,
00218                                  int *error_code)
00219 {
00220   double x;
00221   if (xmlHasProp (node, prop))
00222     x = xml_node_get_float (node, prop, error_code);
00223   else
00224     {
00225       x = default_value;
00226       *error_code = 0;
00227     }
00228   return x;
00229 }
00230
00234 void
00235 xml_node_set_int (xmlNode * node,
00236                   const xmlChar * prop,
00237                   int value)
00238 {
00239   xmlChar buffer[64];
00240   snprintf ((char *) buffer, 64, "%d", value);
00241   xmlSetProp (node, prop, buffer);
00242 }
00243
00247 void
00248 xml_node_set_uint (xmlNode * node,
00249                    const xmlChar * prop,
00250                    unsigned int value)
00251 {
00252   xmlChar buffer[64];
00253   snprintf ((char *) buffer, 64, "%u", value);
00254   xmlSetProp (node, prop, buffer);
00255 }
00256
00260 void
00261 xml_node_set_float (xmlNode * node,
00262                     const xmlChar * prop,
00263                     double value)
00264 {
00265   xmlChar buffer[64];
00266   snprintf ((char *) buffer, 64, "%.14lg", value);
00267   xmlSetProp (node, prop, buffer);
00268 }
00269
00275 int
00276 json_object_get_int (JsonObject * object,
00277                      const char *prop,
00278                      int *error_code)
00279 {
00280   const char *buffer;
00281   int i = 0;
00282   buffer = json_object_get_string_member (object, prop);
00283   if (!buffer)
00284     *error_code = 1;
00285   else
00286     {
00287       if (sscanf (buffer, "%d", &i) != 1)
00288         *error_code = 2;
00289       else
```

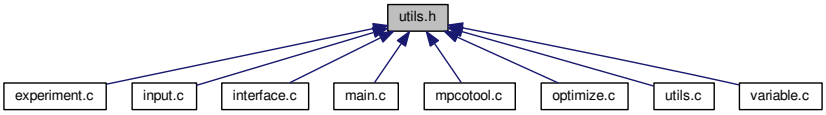
```
00290         *error_code = 0;
00291     }
00292   return i;
00293 }
00294
00300 unsigned int
00301 json_object_get_uint (JsonObject * object,
00302                       const char *prop,
00303                       int *error_code)
00304 {
00305   const char *buffer;
00306   unsigned int i = 0;
00307   buffer = json_object_get_string_member (object, prop);
00308   if (!buffer)
00309     *error_code = 1;
00310   else
00311     {
00312       if (sscanf (buffer, "%u", &i) != 1)
00313         *error_code = 2;
00314       else
00315         *error_code = 0;
00316     }
00317   return i;
00318 }
00319
00326 unsigned int
00327 json_object_get_uint_with_default (JsonObject * object,
00328                                    const char *prop,
00329                                    unsigned int default_value,
00331                                    int *error_code)
00332 {
00333   unsigned int i;
00334   if (json_object_get_member (object, prop))
00335     i = json_object_get_uint (object, prop, error_code);
00336   else
00337     {
00338       i = default_value;
00339       *error_code = 0;
00340     }
00341   return i;
00342 }
00343
00349 double
00350 json_object_get_float (JsonObject * object,
00351                        const char *prop,
00352                        int *error_code)
00353 {
00354   const char *buffer;
00355   double x = 0.;
00356   buffer = json_object_get_string_member (object, prop);
00357   if (!buffer)
00358     *error_code = 1;
00359   else
00360     {
00361       if (sscanf (buffer, "%lf", &x) != 1)
00362         *error_code = 2;
00363       else
00364         *error_code = 0;
00365     }
00366   return x;
00367 }
00368
00375 double
00376 json_object_get_float_with_default (JsonObject * object,
00378                                     const char *prop,
00379                                     double default_value,
00381                                     int *error_code)
00382 {
00383   double x;
00384   if (json_object_get_member (object, prop))
00385     x = json_object_get_float (object, prop, error_code);
00386   else
00387     {
00388       x = default_value;
00389       *error_code = 0;
00390     }
00391   return x;
00392 }
00393
00397 void
00398 json_object_set_int (JsonObject * object,
00399                      const char *prop,
00400                      int value)
00401 {
00402   char buffer[64];
00403   snprintf (buffer, 64, "%d", value);
00404   json_object_set_string_member (object, prop, buffer);
```

```
00405 }
00406
00410 void
00411 json_object_set_uint (JsonObject * object,
00412                       const char *prop,
00413                       unsigned int value)
00415 {
00416   char buffer[64];
00417   snprintf (buffer, 64, "%u", value);
00418   json_object_set_string_member (object, prop, buffer);
00419 }
00420
00424 void
00425 json_object_set_float (JsonObject * object,
00426                        const char *prop,
00427                        double value)
00428 {
00429   char buffer[64];
00430   snprintf (buffer, 64, "%.14lg", value);
00431   json_object_set_string_member (object, prop, buffer);
00432 }
00433
00439 int
00440 cores_number ()
00441 {
00442 #ifdef G_OS_WIN32
00443   SYSTEM_INFO sysinfo;
00444   GetSystemInfo (&sysinfo);
00445   return sysinfo.dwNumberOfProcessors;
00446 #else
00447   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00448 #endif
00449 }
00450
00451 #if HAVE_GTK
00452
00456 void
00457 process_pending ()
00458 {
00459   while (gtk_events_pending ())
00460     gtk_main_iteration ();
00461 }
00462
00468 unsigned int
00469 gtk_array_get_active (GtkRadioButton * array[],
00470                       unsigned int n)
00471 {
00472   unsigned int i;
00473   for (i = 0; i < n; ++i)
00474     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475       break;
00476   return i;
00477 }
00478
00479 #endif
```

4.23 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.

4.23.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Definition in file utils.h.

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.

```
00441 {
00442 #ifdef G_OS_WIN32
00443   SYSTEM_INFO sysinfo;
00444   GetSystemInfo (&sysinfo);
00445   return sysinfo.dwNumberOfProcessors;
00446 #else
00447   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00448 #endif
00449 }
```

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.

```
00471 {
00472   unsigned int i;
00473   for (i = 0; i < n; ++i)
00474     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00475       break;
00476   return i;
00477 }
```

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.

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

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.

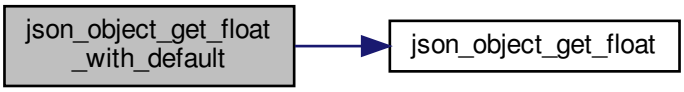
Parameters

object	JSON object.
prop	JSON property.
default_value	default value.
error_code	Error code.

Definition at line 376 of file utils.c.

```
00382 {
00383   double x;
00384   if (json_object_get_member (object, prop))
00385     x = json_object_get_float (object, prop, error_code);
00386   else
00387     {
00388       x = default_value;
00389       *error_code = 0;
00390     }
00391   return x;
00392 }
```

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.

```
00279 {
00280   const char *buffer;
00281   int i = 0;
00282   buffer = json_object_get_string_member (object, prop);
00283   if (!buffer)
00284     *error_code = 1;
00285   else
00286     {
00287       if (sscanf (buffer, "%d", &i) != 1)
00288         *error_code = 2;
00289       else
00290         *error_code = 0;
00291     }
00292   return i;
00293 }
```

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.

```
00304 {
00305   const char *buffer;
00306   unsigned int i = 0;
00307   buffer = json_object_get_string_member (object, prop);
00308   if (!buffer)
00309     *error_code = 1;
00310   else
00311     {
00312       if (sscanf (buffer, "%u", &i) != 1)
00313         *error_code = 2;
00314       else
00315         *error_code = 0;
00316     }
00317   return i;
00318 }
```

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.

```
00332 {
00333   unsigned int i;
00334   if (json_object_get_member (object, prop))
00335     i = json_object_get_uint (object, prop, error_code);
00336   else
00337     {
00338       i = default_value;
00339       *error_code = 0;
00340     }
00341   return i;
00342 }
```

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.

```
00428 {
00429   char buffer[64];
00430   snprintf (buffer, 64, "%.14lg", value);
00431   json_object_set_string_member (object, prop, buffer);
00432 }
```

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.

```
00401 {
00402   char buffer[64];
00403   snprintf (buffer, 64, "%d", value);
00404   json_object_set_string_member (object, prop, buffer);
00405 }
```

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.

```
00415 {
00416   char buffer[64];
00417   snprintf (buffer, 64, "%u", value);
00418   json_object_set_string_member (object, prop, buffer);
00419 }
```

4.23.2.11 process_pending()

```
void process_pending ( )
```

Function to process events on long computation.

Definition at line 457 of file utils.c.

```
00458 {
00459   while (gtk_events_pending ())
00460     gtk_main_iteration ();
00461 }
```

4.23.2.12 show_error()

```
void show_error (
             char * msg )
```

Function to show a dialog with an error message.

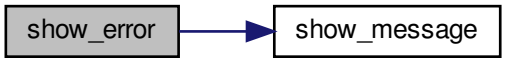
Parameters

msg	Error message.

Definition at line 101 of file utils.c.

```
00102 {
00103   show_message (_("ERROR!"), msg, ERROR_TYPE);
00104 }
```

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.

```
00074 {
00075 #if HAVE_GTK
00076   GtkMessageDialog *dlg;
00077
00078   // Creating the dialog
00079   dlg = (GtkMessageDialog *)
00080     gtk_message_dialog_new (main_window, GTK_DIALOG_MODAL,
00081                             (GtkMessageType) type, GTK_BUTTONS_OK, "%s", msg);
00082
00083   // Setting the dialog title
00084   gtk_window_set_title (GTK_WINDOW (dlg), title);
00085
00086   // Showing the dialog and waiting response
00087   gtk_dialog_run (GTK_DIALOG (dlg));
00088
00089   // Closing and freeing memory
00090   gtk_widget_destroy (GTK_WIDGET (dlg));
00091
00092 #else
00093   printf ("%s: %s\n", title, msg);
00094 #endif
00095 }
```

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.

```
00191 {
00192   double x = 0.;
00193   xmlChar *buffer;
00194   buffer = xmlGetProp (node, prop);
00195   if (!buffer)
00196     *error_code = 1;
00197   else
00198     {
00199       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00200         *error_code = 2;
00201       else
00202         *error_code = 0;
00203       xmlFree (buffer);
00204     }
00205   return x;
00206 }
```

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.

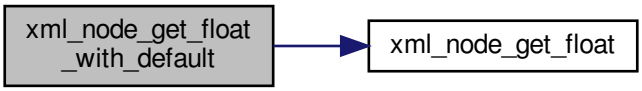
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 215 of file utils.c.

```
00219 {
00220   double x;
00221   if (xmlHasProp (node, prop))
00222     x = xml_node_get_float (node, prop, error_code);
00223   else
00224     {
00225       x = default_value;
00226       *error_code = 0;
00227     }
00228   return x;
00229 }
```

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.

```
00115 {
00116   int i = 0;
00117   xmlChar *buffer;
00118   buffer = xmlGetProp (node, prop);
00119   if (!buffer)
00120     *error_code = 1;
00121   else
00122     {
00123       if (sscanf ((char *) buffer, "%d", &i) != 1)
00124         *error_code = 2;
00125       else
00126         *error_code = 0;
00127       xmlFree (buffer);
00128     }
00129   return i;
00130 }
```

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.

```
00141 {
00142   unsigned int i = 0;
00143   xmlChar *buffer;
00144   buffer = xmlGetProp (node, prop);
00145   if (!buffer)
00146     *error_code = 1;
00147   else
00148     {
00149       if (sscanf ((char *) buffer, "%u", &i) != 1)
00150         *error_code = 2;
00151       else
00152         *error_code = 0;
00153       xmlFree (buffer);
00154     }
00155   return i;
00156 }
```

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.

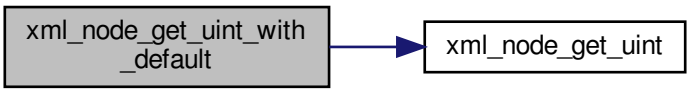
Parameters

node	XML node.
prop	XML property.
default_value	default value.
error_code	Error code.

Definition at line 165 of file utils.c.

```
00170 {
00171   unsigned int i;
00172   if (xmlHasProp (node, prop))
00173     i = xml_node_get_uint (node, prop, error_code);
00174   else
00175     {
00176       i = default_value;
00177       *error_code = 0;
00178     }
00179   return i;
00180 }
```

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.

```
00264 {
00265   xmlChar buffer[64];
00266   snprintf ((char *) buffer, 64, "%.14lg", value);
00267   xmlSetProp (node, prop, buffer);
00268 }
```

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.

```
00238 {
00239   xmlChar buffer[64];
00240   snprintf ((char *) buffer, 64, "%d", value);
00241   xmlSetProp (node, prop, buffer);
00242 }
```

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.

```
00251 {
00252   xmlChar buffer[64];
00253   snprintf ((char *) buffer, 64, "%u", value);
00254   xmlSetProp (node, prop, buffer);
00255 }
```

4.24 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef UTILS__H
00039 #define UTILS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 extern char *error_message;
00057 extern void (*show_pending) ();
00058
00059 // Public functions
00060 void show_message (char *title, char *msg, int type);
00061 void show_error (char *msg);
00062 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00063 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00064                                 int *error_code);
00065 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00066                                              const xmlChar * prop,
00067                                              unsigned int default_value,
00068                                              int *error_code);
00069 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00070                            int *error_code);
00071 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
```

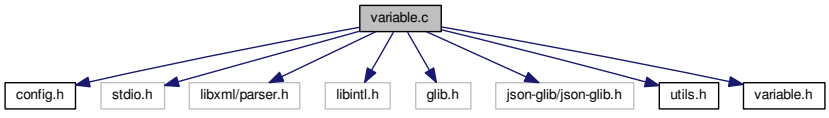
```
    ,
00072                                     double default_value, int *error_code);
00073 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00074 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00075                         unsigned int value);
00076 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00077 int json_object_get_int (JsonObject * object, const char *prop,
00078                          int *error_code);
00079 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00080                                    int *error_code);
00081 unsigned int json_object_get_uint_with_default (JsonObject * object,
00082                                                 const char *prop,
00083                                                 unsigned int default_value,
00084                                                 int *error_code);
00085 double json_object_get_float (JsonObject * object, const char *prop,
00086                               int *error_code);
00087 double json_object_get_float_with_default (JsonObject * object,
00088                                            const char *prop,
00089                                            double default_value,
00090                                            int *error_code);
00091 void json_object_set_int (JsonObject * object, const char *prop, int value);
00092 void json_object_set_uint (JsonObject * object, const char *prop,
00093                            unsigned int value);
00094 void json_object_set_float (JsonObject * object, const char *prop,
00095                             double value);
00096 int cores_number ();
00097 #if HAVE_GTK
00098 void process_pending ();
00099 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00100 #endif
00101
00102 #endif
```

4.25 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.

4.25.1 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.

4.25.2 Function Documentation

4.25.2.1 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.

```
00104 {
00105   char buffer[64];
00106   if (!variable->name)
00107     snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00108   else
00109     snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00110   error_message = g_strdup (buffer);
00111 }
```

4.25.2.2 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.

```
00083 {
00084 #if DEBUG_VARIABLE
00085   fprintf (stderr, "variable_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     xmlFree (variable->name);
00089   else
00090     g_free (variable->name);
00091 #if DEBUG_VARIABLE
00092   fprintf (stderr, "variable_free: end\n");
00093 #endif
00094 }
```

4.25.2.3 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.

```
00065 {
```

```
00066 #if DEBUG_VARIABLE
00067   fprintf (stderr, "variable_new: start\n");
00068 #endif
00069   variable->name = NULL;
00070 #if DEBUG_VARIABLE
00071   fprintf (stderr, "variable_new: end\n");
00072 #endif
00073 }
```

4.25.2.4 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 direction search method.

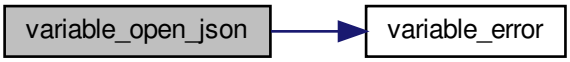
Definition at line 279 of file variable.c.

```
00284 {
00285   JsonObject *object;
00286   const char *label;
00287   int error_code;
00288 #if DEBUG_VARIABLE
00289   fprintf (stderr, "variable_open_json: start\n");
00290 #endif
00291   object = json_node_get_object (node);
00292   label = json_object_get_string_member (object, LABEL_NAME);
00293   if (!label)
00294     {
00295       variable_error (variable, _("no name"));
00296       goto exit_on_error;
00297     }
00298   variable->name = g_strdup (label);
00299   if (json_object_get_member (object, LABEL_MINIMUM))
00300     {
00301       variable->rangemin
00302         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00303       if (error_code)
00304         {
00305           variable_error (variable, _("bad minimum"));
00306           goto exit_on_error;
00307         }
00308       variable->rangeminabs
00309         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00310                                               -G_MAXDOUBLE, &error_code);
00311       if (error_code)
00312         {
00313           variable_error (variable, _("bad absolute minimum"));
```

```
00314           goto exit_on_error;
00315         }
00316       if (variable->rangemin < variable->rangeminabs)
00317         {
00318           variable_error (variable, _("minimum range not allowed"));
00319           goto exit_on_error;
00320         }
00321     }
00322   else
00323     {
00324       variable_error (variable, _("no minimum range"));
00325       goto exit_on_error;
00326     }
00327   if (json_object_get_member (object, LABEL_MAXIMUM))
00328     {
00329       variable->rangemax
00330         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, _("bad maximum"));
00334           goto exit_on_error;
00335         }
00336       variable->rangemaxabs
00337         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MAXIMUM,
00338                                               G_MAXDOUBLE, &error_code);
00339       if (error_code)
00340         {
00341           variable_error (variable, _("bad absolute maximum"));
00342           goto exit_on_error;
00343         }
00344       if (variable->rangemax > variable->rangemaxabs)
00345         {
00346           variable_error (variable, _("maximum range not allowed"));
00347           goto exit_on_error;
00348         }
00349       if (variable->rangemax < variable->rangemin)
00350         {
00351           variable_error (variable, _("bad range"));
00352           goto exit_on_error;
00353         }
00354     }
00355   else
00356     {
00357       variable_error (variable, _("no maximum range"));
00358       goto exit_on_error;
00359     }
00360   variable->precision
00361     = json_object_get_uint_with_default (object,
    LABEL_PRECISION,
00362                                          DEFAULT_PRECISION, &error_code);
00363   if (error_code || variable->precision >= NPRECISIONS)
00364     {
00365       variable_error (variable, _("bad precision"));
00366       goto exit_on_error;
00367     }
00368   if (algorithm == ALGORITHM_SWEEP || algorithm ==
    ALGORITHM_ORTHOGONAL)
00369     {
00370       if (json_object_get_member (object, LABEL_NSWEEPS))
00371         {
00372           variable->nsweeps
00373             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00374           if (error_code || !variable->nsweeps)
00375             {
00376               variable_error (variable, _("bad sweeps"));
00377               goto exit_on_error;
00378             }
00379         }
00380       else
00381         {
00382           variable_error (variable, _("no sweeps number"));
00383           goto exit_on_error;
00384         }
00385 #if DEBUG_VARIABLE
00386       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00387 #endif
00388     }
00389   if (algorithm == ALGORITHM_GENETIC)
00390     {
00391       // Obtaining bits representing each variable
00392       if (json_object_get_member (object, LABEL_NBITS))
00393         {
00394           variable->nbits
00395             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00396           if (error_code || !variable->nbits)
00397             {
```

```
00398               variable_error (variable, _("invalid bits number"));
00399               goto exit_on_error;
00400             }
00401         }
00402       else
00403         {
00404           variable_error (variable, _("no bits number"));
00405           goto exit_on_error;
00406         }
00407     }
00408   else if (nsteps)
00409     {
00410       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00411       if (error_code || variable->step < 0.)
00412         {
00413           variable_error (variable, _("bad step size"));
00414           goto exit_on_error;
00415         }
00416     }
00417
00418 #if DEBUG_VARIABLE
00419   fprintf (stderr, "variable_open_json: end\n");
00420 #endif
00421   return 1;
00422 exit_on_error:
00423   variable_free (variable, INPUT_TYPE_JSON);
00424 #if DEBUG_VARIABLE
00425   fprintf (stderr, "variable_open_json: end\n");
00426 #endif
00427   return 0;
00428 }
```

Here is the call graph for this function:



4.25.2.5 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

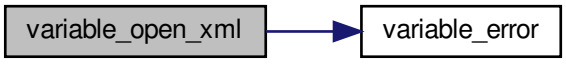
variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the direction search method.

Definition at line 119 of file variable.c.

```
00124 {
00125   int error_code;
00126
00127 #if DEBUG_VARIABLE
00128   fprintf (stderr, "variable_open_xml: start\n");
00129 #endif
00130
00131   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00132   if (!variable->name)
00133     {
00134       variable_error (variable, _("no name"));
00135       goto exit_on_error;
00136     }
00137   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138     {
00139       variable->rangemin
00140         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                               &error_code);
00142       if (error_code)
00143         {
00144           variable_error (variable, _("bad minimum"));
00145           goto exit_on_error;
00146         }
00147       variable->rangeminabs = xml_node_get_float_with_default
00148         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149          &error_code);
00150       if (error_code)
00151         {
00152           variable_error (variable, _("bad absolute minimum"));
00153           goto exit_on_error;
00154         }
00155       if (variable->rangemin < variable->rangeminabs)
00156         {
00157           variable_error (variable, _("minimum range not allowed"));
00158           goto exit_on_error;
00159         }
00160     }
00161   else
00162     {
00163       variable_error (variable, _("no minimum range"));
00164       goto exit_on_error;
00165     }
00166   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167     {
00168       variable->rangemax
00169         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170                               &error_code);
00171       if (error_code)
00172         {
00173           variable_error (variable, _("bad maximum"));
00174           goto exit_on_error;
00175         }
00176       variable->rangemaxabs = xml_node_get_float_with_default
00177         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178          &error_code);
00179       if (error_code)
00180         {
00181           variable_error (variable, _("bad absolute maximum"));
00182           goto exit_on_error;
00183         }
00184       if (variable->rangemax > variable->rangemaxabs)
00185         {
00186           variable_error (variable, _("maximum range not allowed"));
00187           goto exit_on_error;
00188         }
00189       if (variable->rangemax < variable->rangemin)
00190         {
00191           variable_error (variable, _("bad range"));
00192           goto exit_on_error;
00193         }
00194     }
00195   else
00196     {
00197       variable_error (variable, _("no maximum range"));
00198       goto exit_on_error;
00199     }
00200   variable->precision
00201     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_PRECISION,
00202                                       DEFAULT_PRECISION, &error_code);
00203   if (error_code || variable->precision >= NPRECISIONS)
00204     {
00205       variable_error (variable, _("bad precision"));
00206       goto exit_on_error;
```

```
00207     }
00208   if (algorithm == ALGORITHM_SWEEP || algorithm ==
      ALGORITHM_ORTHOGONAL)
00209     {
00210       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00211         {
00212           variable->nsweeps
00213             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00214                                  &error_code);
00215           if (error_code || !variable->nsweeps)
00216             {
00217               variable_error (variable, _("bad sweeps"));
00218               goto exit_on_error;
00219             }
00220         }
00221       else
00222         {
00223           variable_error (variable, _("no sweeps number"));
00224           goto exit_on_error;
00225         }
00226 #if DEBUG_VARIABLE
00227       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00228 #endif
00229     }
00230   if (algorithm == ALGORITHM_GENETIC)
00231     {
00232       // Obtaining bits representing each variable
00233       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234         {
00235           variable->nbits
00236             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                                  &error_code);
00238           if (error_code || !variable->nbits)
00239             {
00240               variable_error (variable, _("invalid bits number"));
00241               goto exit_on_error;
00242             }
00243         }
00244       else
00245         {
00246           variable_error (variable, _("no bits number"));
00247           goto exit_on_error;
00248         }
00249     }
00250   else if (nsteps)
00251     {
00252       variable->step
00253         = xml_node_get_float (node, (const xmlChar *)
      LABEL_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
00255         {
00256           variable_error (variable, _("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG_VARIABLE
00262   fprintf (stderr, "variable_open_xml: end\n");
00263 #endif
00264   return 1;
00265 exit_on_error:
00266   variable_free (variable, INPUT_TYPE_XML);
00267 #if DEBUG_VARIABLE
00268   fprintf (stderr, "variable_open_xml: end\n");
00269 #endif
00270   return 0;
00271 }
```

Here is the call graph for this function:



4.25.3 Variable Documentation

4.25.3.1 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.

4.25.3.2 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.

4.26 variable.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "utils.h"
00046 #include "variable.h"
00047
00048 #define DEBUG_VARIABLE 0
00049
00050 const char *format[NPRECISIONS] = {
00051   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00052   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00053 };
00054
00055 const double precision[NPRECISIONS] = {
00056   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00057   1e-12, 1e-13, 1e-14
00058 };
00059
00063 void
00064 variable_new (Variable * variable)
00065 {
00066 #if DEBUG_VARIABLE
00067   fprintf (stderr, "variable_new: start\n");
00068 #endif
00069   variable->name = NULL;
00070 #if DEBUG_VARIABLE
00071   fprintf (stderr, "variable_new: end\n");
00072 #endif
00073 }
00074
00078 void
00079 variable_free (Variable * variable,
00081                unsigned int type)
00083 {
00084 #if DEBUG_VARIABLE
00085   fprintf (stderr, "variable_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     xmlFree (variable->name);
00089   else
00090     g_free (variable->name);
00091 #if DEBUG_VARIABLE
00092   fprintf (stderr, "variable_free: end\n");
00093 #endif
00094 }
00095
00099 void
00100 variable_error (Variable * variable,
00102                 char *message)
00104 {
00105   char buffer[64];
00106   if (!variable->name)
00107     snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00108   else
00109     snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00110   error_message = g_strdup (buffer);
00111 }
00112
00118 int
00119 variable_open_xml (Variable * variable,
00120                    xmlNode * node,
00121                    unsigned int algorithm,
00122                    unsigned int nsteps)
00124 {
00125   int error_code;
00126
00127 #if DEBUG_VARIABLE
00128   fprintf (stderr, "variable_open_xml: start\n");
00129 #endif
00130
00131   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00132   if (!variable->name)
00133     {
```

```
00134       variable_error (variable, _("no name"));
00135       goto exit_on_error;
00136     }
00137   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138     {
00139       variable->rangemin
00140         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                               &error_code);
00142       if (error_code)
00143         {
00144           variable_error (variable, _("bad minimum"));
00145           goto exit_on_error;
00146         }
00147       variable->rangeminabs = xml_node_get_float_with_default
00148         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149          &error_code);
00150       if (error_code)
00151         {
00152           variable_error (variable, _("bad absolute minimum"));
00153           goto exit_on_error;
00154         }
00155       if (variable->rangemin < variable->rangeminabs)
00156         {
00157           variable_error (variable, _("minimum range not allowed"));
00158           goto exit_on_error;
00159         }
00160     }
00161   else
00162     {
00163       variable_error (variable, _("no minimum range"));
00164       goto exit_on_error;
00165     }
00166   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167     {
00168       variable->rangemax
00169         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170                               &error_code);
00171       if (error_code)
00172         {
00173           variable_error (variable, _("bad maximum"));
00174           goto exit_on_error;
00175         }
00176       variable->rangemaxabs = xml_node_get_float_with_default
00177         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178          &error_code);
00179       if (error_code)
00180         {
00181           variable_error (variable, _("bad absolute maximum"));
00182           goto exit_on_error;
00183         }
00184       if (variable->rangemax > variable->rangemaxabs)
00185         {
00186           variable_error (variable, _("maximum range not allowed"));
00187           goto exit_on_error;
00188         }
00189       if (variable->rangemax < variable->rangemin)
00190         {
00191           variable_error (variable, _("bad range"));
00192           goto exit_on_error;
00193         }
00194     }
00195   else
00196     {
00197       variable_error (variable, _("no maximum range"));
00198       goto exit_on_error;
00199     }
00200   variable->precision
00201     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_PRECISION,
00202                                       DEFAULT_PRECISION, &error_code);
00203   if (error_code || variable->precision >= NPRECISIONS)
00204     {
00205       variable_error (variable, _("bad precision"));
00206       goto exit_on_error;
00207     }
00208   if (algorithm == ALGORITHM_SWEEP || algorithm ==
      ALGORITHM_ORTHOGONAL)
00209     {
00210       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00211         {
00212           variable->nsweeps
00213             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00214                                  &error_code);
00215           if (error_code || !variable->nsweeps)
00216             {
00217               variable_error (variable, _("bad sweeps"));
00218               goto exit_on_error;
```

```
00219             }
00220         }
00221       else
00222         {
00223           variable_error (variable, _("no sweeps number"));
00224           goto exit_on_error;
00225         }
00226 #if DEBUG_VARIABLE
00227       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00228 #endif
00229     }
00230   if (algorithm == ALGORITHM_GENETIC)
00231     {
00232       // Obtaining bits representing each variable
00233       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234         {
00235           variable->nbits
00236             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                                  &error_code);
00238           if (error_code || !variable->nbits)
00239             {
00240               variable_error (variable, _("invalid bits number"));
00241               goto exit_on_error;
00242             }
00243         }
00244       else
00245         {
00246           variable_error (variable, _("no bits number"));
00247           goto exit_on_error;
00248         }
00249     }
00250   else if (nsteps)
00251     {
00252       variable->step
00253         = xml_node_get_float (node, (const xmlChar *)
      LABEL_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
00255         {
00256           variable_error (variable, _("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG_VARIABLE
00262   fprintf (stderr, "variable_open_xml: end\n");
00263 #endif
00264   return 1;
00265 exit_on_error:
00266   variable_free (variable, INPUT_TYPE_XML);
00267 #if DEBUG_VARIABLE
00268   fprintf (stderr, "variable_open_xml: end\n");
00269 #endif
00270   return 0;
00271 }
00272
00278 int
00279 variable_open_json (Variable * variable,
00280                     JsonNode * node,
00281                     unsigned int algorithm,
00282                     unsigned int nsteps)
00284 {
00285   JsonObject *object;
00286   const char *label;
00287   int error_code;
00288 #if DEBUG_VARIABLE
00289   fprintf (stderr, "variable_open_json: start\n");
00290 #endif
00291   object = json_node_get_object (node);
00292   label = json_object_get_string_member (object, LABEL_NAME);
00293   if (!label)
00294     {
00295       variable_error (variable, _("no name"));
00296       goto exit_on_error;
00297     }
00298   variable->name = g_strdup (label);
00299   if (json_object_get_member (object, LABEL_MINIMUM))
00300     {
00301       variable->rangemin
00302         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00303       if (error_code)
00304         {
00305           variable_error (variable, _("bad minimum"));
00306           goto exit_on_error;
00307         }
00308       variable->rangeminabs
00309         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
```

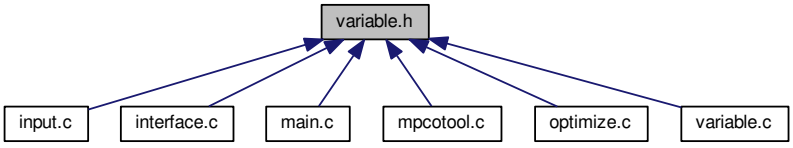
```
00310                                                 -G_MAXDOUBLE, &error_code);
00311       if (error_code)
00312         {
00313           variable_error (variable, _("bad absolute minimum"));
00314           goto exit_on_error;
00315         }
00316       if (variable->rangemin < variable->rangeminabs)
00317         {
00318           variable_error (variable, _("minimum range not allowed"));
00319           goto exit_on_error;
00320         }
00321     }
00322   else
00323     {
00324       variable_error (variable, _("no minimum range"));
00325       goto exit_on_error;
00326     }
00327   if (json_object_get_member (object, LABEL_MAXIMUM))
00328     {
00329       variable->rangemax
00330         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, _("bad maximum"));
00334           goto exit_on_error;
00335         }
00336       variable->rangemaxabs
00337         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MAXIMUM,
00338                                               G_MAXDOUBLE, &error_code);
00339       if (error_code)
00340         {
00341           variable_error (variable, _("bad absolute maximum"));
00342           goto exit_on_error;
00343         }
00344       if (variable->rangemax > variable->rangemaxabs)
00345         {
00346           variable_error (variable, _("maximum range not allowed"));
00347           goto exit_on_error;
00348         }
00349       if (variable->rangemax < variable->rangemin)
00350         {
00351           variable_error (variable, _("bad range"));
00352           goto exit_on_error;
00353         }
00354     }
00355   else
00356     {
00357       variable_error (variable, _("no maximum range"));
00358       goto exit_on_error;
00359     }
00360   variable->precision
00361     = json_object_get_uint_with_default (object,
    LABEL_PRECISION,
00362                                          DEFAULT_PRECISION, &error_code);
00363   if (error_code || variable->precision >= NPRECISIONS)
00364     {
00365       variable_error (variable, _("bad precision"));
00366       goto exit_on_error;
00367     }
00368   if (algorithm == ALGORITHM_SWEEP || algorithm ==
    ALGORITHM_ORTHOGONAL)
00369     {
00370       if (json_object_get_member (object, LABEL_NSWEEPS))
00371         {
00372           variable->nsweeps
00373             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00374           if (error_code || !variable->nsweeps)
00375             {
00376               variable_error (variable, _("bad sweeps"));
00377               goto exit_on_error;
00378             }
00379         }
00380       else
00381         {
00382           variable_error (variable, _("no sweeps number"));
00383           goto exit_on_error;
00384         }
00385 #if DEBUG_VARIABLE
00386       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00387 #endif
00388     }
00389   if (algorithm == ALGORITHM_GENETIC)
00390     {
00391       // Obtaining bits representing each variable
00392       if (json_object_get_member (object, LABEL_NBITS))
00393         {
```

```
00394           variable->nbits
00395             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00396           if (error_code || !variable->nbits)
00397             {
00398               variable_error (variable, _("invalid bits number"));
00399               goto exit_on_error;
00400             }
00401         }
00402       else
00403         {
00404           variable_error (variable, _("no bits number"));
00405           goto exit_on_error;
00406         }
00407     }
00408   else if (nsteps)
00409     {
00410       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00411       if (error_code || variable->step < 0.)
00412         {
00413           variable_error (variable, _("bad step size"));
00414           goto exit_on_error;
00415         }
00416     }
00417
00418 #if DEBUG_VARIABLE
00419   fprintf (stderr, "variable_open_json: end\n");
00420 #endif
00421   return 1;
00422 exit_on_error:
00423   variable_free (variable, INPUT_TYPE_JSON);
00424 #if DEBUG_VARIABLE
00425   fprintf (stderr, "variable_open_json: end\n");
00426 #endif
00427   return 0;
00428 }
```

4.27 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.

4.27.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Copyright 2012-2018, all rights reserved.

Definition in file variable.h.

4.27.2 Enumeration Type Documentation

4.27.2.1 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.

```
00043 {
00044   ALGORITHM_MONTE_CARLO = 0,
00045   ALGORITHM_SWEEP = 1,
00046   ALGORITHM_GENETIC = 2,
00047   ALGORITHM_ORTHOGONAL = 3
00048 };
```

4.27.3 Function Documentation

4.27.3.1 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.

```
00104 {
00105   char buffer[64];
00106   if (!variable->name)
00107     snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00108   else
00109     snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name, message);
00110   error_message = g_strdup (buffer);
00111 }
```

4.27.3.2 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.

```
00083 {
00084 #if DEBUG_VARIABLE
00085   fprintf (stderr, "variable_free: start\n");
00086 #endif
00087   if (type == INPUT_TYPE_XML)
00088     xmlFree (variable->name);
00089   else
00090     g_free (variable->name);
00091 #if DEBUG_VARIABLE
00092   fprintf (stderr, "variable_free: end\n");
00093 #endif
00094 }
```

4.27.3.3 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.

```
00065 {
00066 #if DEBUG_VARIABLE
00067   fprintf (stderr, "variable_new: start\n");
00068 #endif
00069   variable->name = NULL;
00070 #if DEBUG_VARIABLE
00071   fprintf (stderr, "variable_new: end\n");
00072 #endif
00073 }
```

4.27.3.4 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

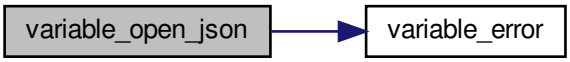
variable	Variable struct.
node	XML node.
algorithm	Algorithm type.
nsteps	Number of steps to do the direction search method.

Definition at line 279 of file variable.c.

```
00284 {
00285   JsonObject *object;
00286   const char *label;
00287   int error_code;
00288 #if DEBUG_VARIABLE
00289   fprintf (stderr, "variable_open_json: start\n");
00290 #endif
00291   object = json_node_get_object (node);
00292   label = json_object_get_string_member (object, LABEL_NAME);
00293   if (!label)
00294     {
00295       variable_error (variable, _("no name"));
00296       goto exit_on_error;
00297     }
00298   variable->name = g_strdup (label);
00299   if (json_object_get_member (object, LABEL_MINIMUM))
00300     {
00301       variable->rangemin
00302         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00303       if (error_code)
00304         {
00305           variable_error (variable, _("bad minimum"));
00306           goto exit_on_error;
00307         }
00308       variable->rangeminabs
00309         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00310                                               -G_MAXDOUBLE, &error_code);
00311       if (error_code)
00312         {
00313           variable_error (variable, _("bad absolute minimum"));
00314           goto exit_on_error;
00315         }
00316       if (variable->rangemin < variable->rangeminabs)
00317         {
00318           variable_error (variable, _("minimum range not allowed"));
00319           goto exit_on_error;
00320         }
00321     }
00322   else
00323     {
00324       variable_error (variable, _("no minimum range"));
00325       goto exit_on_error;
00326     }
00327   if (json_object_get_member (object, LABEL_MAXIMUM))
00328     {
00329       variable->rangemax
00330         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, _("bad maximum"));
00334           goto exit_on_error;
00335         }
00336       variable->rangemaxabs
00337         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00338                                               G_MAXDOUBLE, &error_code);
00339       if (error_code)
00340         {
00341           variable_error (variable, _("bad absolute maximum"));
00342           goto exit_on_error;
00343         }
00344       if (variable->rangemax > variable->rangemaxabs)
00345         {
00346           variable_error (variable, _("maximum range not allowed"));
00347           goto exit_on_error;
00348         }
00349       if (variable->rangemax < variable->rangemin)
00350         {
00351           variable_error (variable, _("bad range"));
```

```
00352           goto exit_on_error;
00353         }
00354     }
00355   else
00356     {
00357       variable_error (variable, _("no maximum range"));
00358       goto exit_on_error;
00359     }
00360   variable->precision
00361     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00362                                          DEFAULT_PRECISION, &error_code);
00363   if (error_code || variable->precision >= NPRECISIONS)
00364     {
00365       variable_error (variable, _("bad precision"));
00366       goto exit_on_error;
00367     }
00368   if (algorithm == ALGORITHM_SWEEP || algorithm ==
      ALGORITHM_ORTHOGONAL)
00369     {
00370       if (json_object_get_member (object, LABEL_NSWEEPS))
00371         {
00372           variable->nsweeps
00373             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00374           if (error_code || !variable->nsweeps)
00375             {
00376               variable_error (variable, _("bad sweeps"));
00377               goto exit_on_error;
00378             }
00379         }
00380       else
00381         {
00382           variable_error (variable, _("no sweeps number"));
00383           goto exit_on_error;
00384         }
00385 #if DEBUG_VARIABLE
00386       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00387 #endif
00388     }
00389   if (algorithm == ALGORITHM_GENETIC)
00390     {
00391       // Obtaining bits representing each variable
00392       if (json_object_get_member (object, LABEL_NBITS))
00393         {
00394           variable->nbits
00395             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00396           if (error_code || !variable->nbits)
00397             {
00398               variable_error (variable, _("invalid bits number"));
00399               goto exit_on_error;
00400             }
00401         }
00402       else
00403         {
00404           variable_error (variable, _("no bits number"));
00405           goto exit_on_error;
00406         }
00407     }
00408   else if (nsteps)
00409     {
00410       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00411       if (error_code || variable->step < 0.)
00412         {
00413           variable_error (variable, _("bad step size"));
00414           goto exit_on_error;
00415         }
00416     }
00417
00418 #if DEBUG_VARIABLE
00419   fprintf (stderr, "variable_open_json: end\n");
00420 #endif
00421   return 1;
00422 exit_on_error:
00423   variable_free (variable, INPUT_TYPE_JSON);
00424 #if DEBUG_VARIABLE
00425   fprintf (stderr, "variable_open_json: end\n");
00426 #endif
00427   return 0;
00428 }
```

Here is the call graph for this function:



4.27.3.5 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 direction search method.

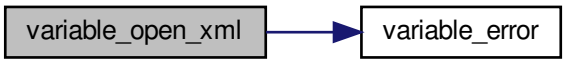
Definition at line 119 of file variable.c.

```
00124 {
00125   int error_code;
00126
00127 #if DEBUG_VARIABLE
00128   fprintf (stderr, "variable_open_xml: start\n");
00129 #endif
00130
00131   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00132   if (!variable->name)
00133     {
00134       variable_error (variable, _("no name"));
00135       goto exit_on_error;
00136     }
00137   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00138     {
00139       variable->rangemin
00140         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00141                               &error_code);
00142       if (error_code)
00143         {
00144           variable_error (variable, _("bad minimum"));
00145           goto exit_on_error;
```

```
00146         }
00147       variable->rangeminabs = xml_node_get_float_with_default
00148         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00149          &error_code);
00150       if (error_code)
00151         {
00152           variable_error (variable, _("bad absolute minimum"));
00153           goto exit_on_error;
00154         }
00155       if (variable->rangemin < variable->rangeminabs)
00156         {
00157           variable_error (variable, _("minimum range not allowed"));
00158           goto exit_on_error;
00159         }
00160     }
00161   else
00162     {
00163       variable_error (variable, _("no minimum range"));
00164       goto exit_on_error;
00165     }
00166   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00167     {
00168       variable->rangemax
00169         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00170                               &error_code);
00171       if (error_code)
00172         {
00173           variable_error (variable, _("bad maximum"));
00174           goto exit_on_error;
00175         }
00176       variable->rangemaxabs = xml_node_get_float_with_default
00177         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00178          &error_code);
00179       if (error_code)
00180         {
00181           variable_error (variable, _("bad absolute maximum"));
00182           goto exit_on_error;
00183         }
00184       if (variable->rangemax > variable->rangemaxabs)
00185         {
00186           variable_error (variable, _("maximum range not allowed"));
00187           goto exit_on_error;
00188         }
00189       if (variable->rangemax < variable->rangemin)
00190         {
00191           variable_error (variable, _("bad range"));
00192           goto exit_on_error;
00193         }
00194     }
00195   else
00196     {
00197       variable_error (variable, _("no maximum range"));
00198       goto exit_on_error;
00199     }
00200   variable->precision
00201     = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_PRECISION,
00202                                       DEFAULT_PRECISION, &error_code);
00203   if (error_code || variable->precision >= NPRECISIONS)
00204     {
00205       variable_error (variable, _("bad precision"));
00206       goto exit_on_error;
00207     }
00208   if (algorithm == ALGORITHM_SWEEP || algorithm ==
    ALGORITHM_ORTHOGONAL)
00209     {
00210       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00211         {
00212           variable->nsweeps
00213             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00214                                  &error_code);
00215           if (error_code || !variable->nsweeps)
00216             {
00217               variable_error (variable, _("bad sweeps"));
00218               goto exit_on_error;
00219             }
00220         }
00221       else
00222         {
00223           variable_error (variable, _("no sweeps number"));
00224           goto exit_on_error;
00225         }
00226 #if DEBUG_VARIABLE
00227       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00228 #endif
00229     }
00230   if (algorithm == ALGORITHM_GENETIC)
```

```
00231     {
00232       // Obtaining bits representing each variable
00233       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00234         {
00235           variable->nbits
00236             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00237                                  &error_code);
00238           if (error_code || !variable->nbits)
00239             {
00240               variable_error (variable, _("invalid bits number"));
00241               goto exit_on_error;
00242             }
00243         }
00244       else
00245         {
00246           variable_error (variable, _("no bits number"));
00247           goto exit_on_error;
00248         }
00249     }
00250   else if (nsteps)
00251     {
00252       variable->step
00253         = xml_node_get_float (node, (const xmlChar *)
    LABEL_STEP, &error_code);
00254       if (error_code || variable->step < 0.)
00255         {
00256           variable_error (variable, _("bad step size"));
00257           goto exit_on_error;
00258         }
00259     }
00260
00261 #if DEBUG_VARIABLE
00262   fprintf (stderr, "variable_open_xml: end\n");
00263 #endif
00264   return 1;
00265 exit_on_error:
00266   variable_free (variable, INPUT_TYPE_XML);
00267 #if DEBUG_VARIABLE
00268   fprintf (stderr, "variable_open_xml: end\n");
00269 #endif
00270   return 0;
00271 }
```

Here is the call graph for this function:



4.28 variable.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2018, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
```

```
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef VARIABLE__H
00039 #define VARIABLE__H 1
00040
00042 enum Algorithm
00043 {
00044   ALGORITHM_MONTE_CARLO = 0,
00045   ALGORITHM_SWEEP = 1,
00046   ALGORITHM_GENETIC = 2,
00047   ALGORITHM_ORTHOGONAL = 3
00048 };
00049
00054 typedef struct
00055 {
00056   char *name;
00057   double rangemin;
00058   double rangemax;
00059   double rangeminabs;
00060   double rangemaxabs;
00061   double step;
00062   unsigned int precision;
00063   unsigned int nsweeps;
00064   unsigned int nbits;
00065 } Variable;
00066
00067 extern const char *format[NPRECISIONS];
00068 extern const double precision[NPRECISIONS];
00069
00070 // Public functions
00071 void variable_new (Variable * variable);
00072 void variable_free (Variable * variable, unsigned int type);
00073 void variable_error (Variable * variable, char *message);
00074 int variable_open_xml (Variable * variable, xmlNode * node,
00075                        unsigned int algorithm, unsigned int nsteps);
00076 int variable_open_json (Variable * variable, JsonNode * node,
00077                         unsigned int algorithm, unsigned int nsteps);
00078
00079 #endif
```

Index