MPCOTool

3.4.0

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 * template [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:

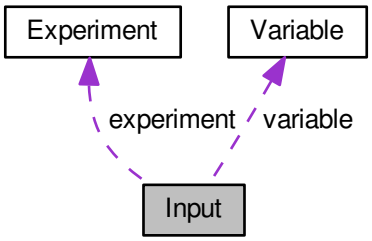
- experiment.h

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:

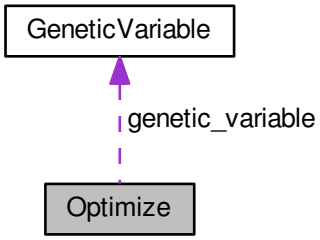
- input.h

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 56 of file variable.h.

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

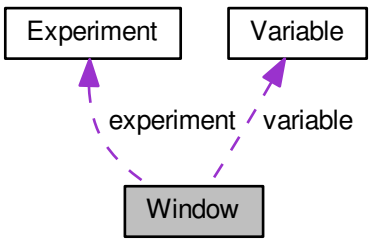
- variable.h

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

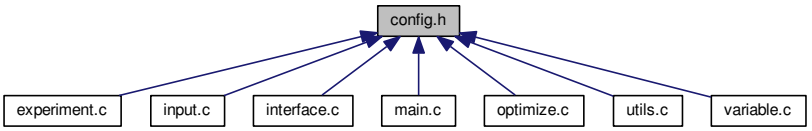
Chapter 4

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 3

  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_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

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 128 of file config.h.

```
00129 {
00130   INPUT_TYPE_XML = 0,
00131   INPUT_TYPE_JSON = 1
00132 };
```

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 2012-2016, 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 3
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"
```

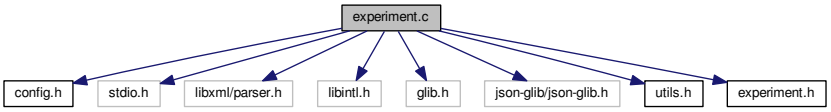
```
00094 #define LABEL_NSTEPS "nsteps"
00095 #define LABEL_NSWEEPS "nsweeps"
00096 #define LABEL_P "p"
00097 #define LABEL_PRECISION "precision"
00098 #define LABEL_RANDOM "random"
00099 #define LABEL_RELAXATION "relaxation"
00100 #define LABEL_REPRODUCTION "reproduction"
00101 #define LABEL_RESULT_FILE "result_file"
00102 #define LABEL_SIMULATOR "simulator"
00103 #define LABEL_SEED "seed"
00104 #define LABEL_STEP "step"
00105 #define LABEL_SWEEP "sweep"
00106 #define LABEL_TAXICAB "taxicab"
00107 #define LABEL_TEMPLATE1 "template1"
00108 #define LABEL_TEMPLATE2 "template2"
00109 #define LABEL_TEMPLATE3 "template3"
00110 #define LABEL_TEMPLATE4 "template4"
00111 #define LABEL_TEMPLATE5 "template5"
00112 #define LABEL_TEMPLATE6 "template6"
00113 #define LABEL_TEMPLATE7 "template7"
00114 #define LABEL_TEMPLATE8 "template8"
00115 #define LABEL_THRESHOLD "threshold"
00116 #define LABEL_TOLERANCE "tolerance"
00117 #define LABEL_VARIABLE "variable"
00118 #define LABEL_VARIABLES "variables"
00119 #define LABEL_VARIABLES_FILE "variables_file"
00120 #define LABEL_WEIGHT "weight"
00121
00122 // Enumerations
00123
00128 enum INPUT_TYPE
00129 {
00130   INPUT_TYPE_XML = 0,
00131   INPUT_TYPE_JSON = 1
00132 };
00133
00134 #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)

  Function to create a new Experiment struct.
- void experiment_free (Experiment *experiment, unsigned int type)

  Function to free the memory of an Experiment struct.
- void experiment_error (Experiment *experiment, char *message)

  Function to print a message error opening an Experiment struct.
- int experiment_open_xml (Experiment *experiment, xmlNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.
- int experiment_open_json (Experiment *experiment, JsonNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.

Variables

- const char * template [MAX_NINPUTS]

  Array of xmlChar strings with template 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 121 of file experiment.c.

```
00122 {
00123   char buffer[64];
00124   if (!experiment->name)
00125     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128               experiment->name, message);
00129   error_message = g_strdup (buffer);
00130 }
```

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 88 of file experiment.c.

```
00089 {
00090   unsigned int i;
00091 #if DEBUG_EXPERIMENT
00092   fprintf (stderr, "experiment_free: start\n");
00093 #endif
00094   if (type == INPUT_TYPE_XML)
00095     {
00096       for (i = 0; i < experiment->ninputs; ++i)
00097         xmlFree (experiment->template[i]);
00098       xmlFree (experiment->name);
00099     }
00100   else
00101     {
00102       for (i = 0; i < experiment->ninputs; ++i)
00103         g_free (experiment->template[i]);
00104       g_free (experiment->name);
00105     }
00106   experiment->ninputs = 0;
00107 #if DEBUG_EXPERIMENT
00108   fprintf (stderr, "experiment_free: end\n");
00109 #endif
00110 }
```

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 64 of file experiment.c.

```
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new: start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->template[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new: end\n");
00076 #endif
00077 }
```

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.

Parameters

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

Returns

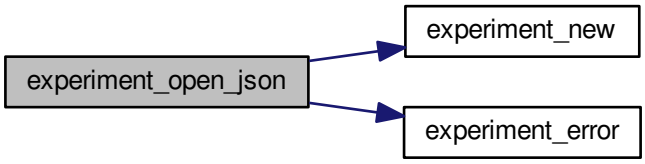
1 on success, 0 on error.

Definition at line 254 of file experiment.c.

```
00256 {
00257   char buffer[64];
00258   JsonObject *object;
00259   const char *name;
00260   int error_code;
00261   unsigned int i;
00262
00263 #if DEBUG_EXPERIMENT
00264   fprintf (stderr, "experiment_open_json: start\n");
00265 #endif
00266
00267   // Resetting experiment data
00268   experiment_new (experiment);
00269
00270   // Getting JSON object
00271   object = json_node_get_object (node);
00272
```

```
00273   // Reading the experimental data
00274   name = json_object_get_string_member (object, LABEL_NAME);
00275   if (!name)
00276     {
00277       experiment_error (experiment, _("no data file name"));
00278       goto exit_on_error;
00279     }
00280   experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284   experiment->weight
00285     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00286                                           &error_code);
00287   if (error_code)
00288     {
00289       experiment_error (experiment, _("bad weight"));
00290       goto exit_on_error;
00291     }
00292 #if DEBUG_EXPERIMENT
00293   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294 #endif
00295   name = json_object_get_string_member (object, template[0]);
00296   if (name)
00297     {
00298 #if DEBUG_EXPERIMENT
00299       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300                name, template[0]);
00301 #endif
00302       ++experiment->ninputs;
00303     }
00304   else
00305     {
00306       experiment_error (experiment, _("no template"));
00307       goto exit_on_error;
00308     }
00309   experiment->template[0] = g_strdup (name);
00310   for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312 #if DEBUG_EXPERIMENT
00313       fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314 #endif
00315       if (json_object_get_member (object, template[i]))
00316         {
00317           if (ninputs && ninputs <= i)
00318             {
00319               experiment_error (experiment, _("bad templates number"));
00320               goto exit_on_error;
00321             }
00322           name = json_object_get_string_member (object, template[i]);
00323 #if DEBUG_EXPERIMENT
00324           fprintf (stderr,
00325                    "experiment_open_json: experiment=%s template%u=%s\n",
00326                    experiment->nexperiments, name, template[i]);
00327 #endif
00328           experiment->template[i] = g_strdup (name);
00329           ++experiment->ninputs;
00330         }
00331       else if (ninputs && ninputs > i)
00332         {
00333           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334           experiment_error (experiment, buffer);
00335           goto exit_on_error;
00336         }
00337       else
00338         break;
00339     }
00340
00341 #if DEBUG_EXPERIMENT
00342   fprintf (stderr, "experiment_open_json: end\n");
00343 #endif
00344   return 1;
00345
00346 exit_on_error:
00347   experiment_free (experiment, INPUT_TYPE_JSON);
00348 #if DEBUG_EXPERIMENT
00349   fprintf (stderr, "experiment_open_json: end\n");
00350 #endif
00351   return 0;
00352 }
```

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.

Parameters

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

Returns

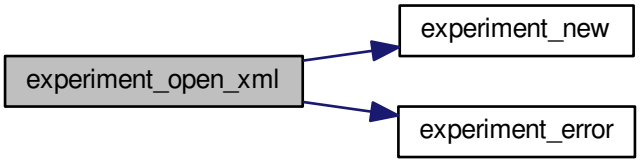
1 on success, 0 on error.

Definition at line 145 of file experiment.c.

```
00147 {
00148   char buffer[64];
00149   int error_code;
00150   unsigned int i;
00151
00152 #if DEBUG_EXPERIMENT
00153   fprintf (stderr, "experiment_open_xml: start\n");
00154 #endif
00155
00156   // Resetting experiment data
00157   experiment_new (experiment);
00158
00159   // Reading the experimental data
00160   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161   if (!experiment->name)
00162     {
00163       experiment_error (experiment, _("no data file name"));
00164       goto exit_on_error;
00165     }
00166 #if DEBUG_EXPERIMENT
00167   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
```

```
00168 #endif
00169   experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00172                                      &error_code);
00173   if (error_code)
00174     {
00175       experiment_error (experiment, _("bad weight"));
00176       goto exit_on_error;
00177     }
00178 #if DEBUG_EXPERIMENT
00179   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180 #endif
00181   experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183   if (experiment->template[0])
00184     {
00185 #if DEBUG_EXPERIMENT
00186       fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00187                experiment->name, template[0]);
00188 #endif
00189       ++experiment->ninputs;
00190     }
00191   else
00192     {
00193       experiment_error (experiment, _("no template"));
00194       goto exit_on_error;
00195     }
00196   for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198 #if DEBUG_EXPERIMENT
00199       fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200 #endif
00201       if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203           if (ninputs && ninputs <= i)
00204             {
00205               experiment_error (experiment, _("bad templates number"));
00206               goto exit_on_error;
00207             }
00208           experiment->template[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210 #if DEBUG_EXPERIMENT
00211           fprintf (stderr,
00212                    "experiment_open_xml: experiment=%s template%u=%s\n",
00213                    experiment->nexperiments, experiment->name,
00214                    experiment->template[i]);
00215 #endif
00216           ++experiment->ninputs;
00217         }
00218       else if (ninputs && ninputs > i)
00219         {
00220           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00221           experiment_error (experiment, buffer);
00222           goto exit_on_error;
00223         }
00224       else
00225         break;
00226     }
00227
00228 #if DEBUG_EXPERIMENT
00229   fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231   return 1;
00232
00233 exit_on_error:
00234   experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236   fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238   return 0;
00239 }
```

Here is the call graph for this function:



4.3.3 Variable Documentation

4.3.3.1 template

```
const char* template[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 template 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 2012-2016, 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 *template[MAX_NINPUTS] = {
00051   LABEL_TEMPLATE1, LABEL_TEMPLATE2,
      LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00052   LABEL_TEMPLATE5, LABEL_TEMPLATE6,
      LABEL_TEMPLATE7, LABEL_TEMPLATE8
00053 };
00054
00056
00063 void
00064 experiment_new (Experiment * experiment)
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new: start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->template[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new: end\n");
00076 #endif
00077 }
00078
00087 void
00088 experiment_free (Experiment * experiment, unsigned int type)
00089 {
00090   unsigned int i;
00091 #if DEBUG_EXPERIMENT
00092   fprintf (stderr, "experiment_free: start\n");
00093 #endif
00094   if (type == INPUT_TYPE_XML)
00095     {
00096       for (i = 0; i < experiment->ninputs; ++i)
00097         xmlFree (experiment->template[i]);
00098       xmlFree (experiment->name);
00099     }
00100   else
00101     {
00102       for (i = 0; i < experiment->ninputs; ++i)
00103         g_free (experiment->template[i]);
00104       g_free (experiment->name);
00105     }
00106   experiment->ninputs = 0;
00107 #if DEBUG_EXPERIMENT
00108   fprintf (stderr, "experiment_free: end\n");
00109 #endif
00110 }
00111
00120 void
00121 experiment_error (Experiment * experiment, char *message)
00122 {
00123   char buffer[64];
00124   if (!experiment->name)
00125     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128               experiment->name, message);
00129   error_message = g_strdup (buffer);
00130 }
00131
00144 int
00145 experiment_open_xml (Experiment * experiment, xmlNode * node,
00146                      unsigned int ninputs)
00147 {
```

```
00148   char buffer[64];
00149   int error_code;
00150   unsigned int i;
00151 
00152 #if DEBUG_EXPERIMENT
00153   fprintf (stderr, "experiment_open_xml: start\n");
00154 #endif
00155 
00156   // Resetting experiment data
00157   experiment_new (experiment);
00158 
00159   // Reading the experimental data
00160   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161   if (!experiment->name)
00162     {
00163       experiment_error (experiment, _("no data file name"));
00164       goto exit_on_error;
00165     }
00166 #if DEBUG_EXPERIMENT
00167   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168 #endif
00169   experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00172                                      &error_code);
00173   if (error_code)
00174     {
00175       experiment_error (experiment, _("bad weight"));
00176       goto exit_on_error;
00177     }
00178 #if DEBUG_EXPERIMENT
00179   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180 #endif
00181   experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183   if (experiment->template[0])
00184     {
00185 #if DEBUG_EXPERIMENT
00186       fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00187                experiment->name, template[0]);
00188 #endif
00189       ++experiment->ninputs;
00190     }
00191   else
00192     {
00193       experiment_error (experiment, _("no template"));
00194       goto exit_on_error;
00195     }
00196   for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198 #if DEBUG_EXPERIMENT
00199       fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200 #endif
00201       if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203           if (ninputs && ninputs <= i)
00204             {
00205               experiment_error (experiment, _("bad templates number"));
00206               goto exit_on_error;
00207             }
00208           experiment->template[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210 #if DEBUG_EXPERIMENT
00211           fprintf (stderr,
00212                    "experiment_open_xml: experiment=%s template%u=%s\n",
00213                    experiment->nexperiments, experiment->name,
00214                    experiment->template[i]);
00215 #endif
00216           ++experiment->ninputs;
00217         }
00218       else if (ninputs && ninputs > i)
00219         {
00220           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00221           experiment_error (experiment, buffer);
00222           goto exit_on_error;
00223         }
00224       else
00225         break;
00226     }
00227 
00228 #if DEBUG_EXPERIMENT
00229   fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231   return 1;
00232 
00233 exit_on_error:
```

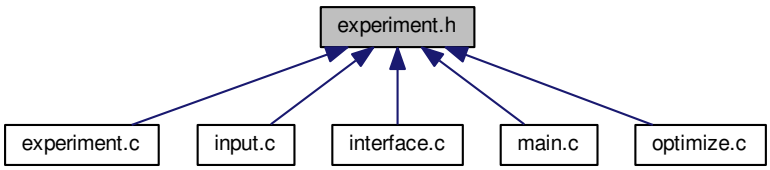
```
00234   experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236   fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238   return 0;
00239 }
00240
00253 int
00254 experiment_open_json (Experiment * experiment, JsonNode * node,
00255                       unsigned int ninputs)
00256 {
00257   char buffer[64];
00258   JsonObject *object;
00259   const char *name;
00260   int error_code;
00261   unsigned int i;
00262
00263 #if DEBUG_EXPERIMENT
00264   fprintf (stderr, "experiment_open_json: start\n");
00265 #endif
00266
00267   // Resetting experiment data
00268   experiment_new (experiment);
00269
00270   // Getting JSON object
00271   object = json_node_get_object (node);
00272
00273   // Reading the experimental data
00274   name = json_object_get_string_member (object, LABEL_NAME);
00275   if (!name)
00276     {
00277       experiment_error (experiment, _("no data file name"));
00278       goto exit_on_error;
00279     }
00280   experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284   experiment->weight
00285     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00286                                           &error_code);
00287   if (error_code)
00288     {
00289       experiment_error (experiment, _("bad weight"));
00290       goto exit_on_error;
00291     }
00292 #if DEBUG_EXPERIMENT
00293   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294 #endif
00295   name = json_object_get_string_member (object, template[0]);
00296   if (name)
00297     {
00298 #if DEBUG_EXPERIMENT
00299       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300                name, template[0]);
00301 #endif
00302       ++experiment->ninputs;
00303     }
00304   else
00305     {
00306       experiment_error (experiment, _("no template"));
00307       goto exit_on_error;
00308     }
00309   experiment->template[0] = g_strdup (name);
00310   for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312 #if DEBUG_EXPERIMENT
00313       fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314 #endif
00315       if (json_object_get_member (object, template[i]))
00316         {
00317           if (ninputs && ninputs <= i)
00318             {
00319               experiment_error (experiment, _("bad templates number"));
00320               goto exit_on_error;
00321             }
00322           name = json_object_get_string_member (object, template[i]);
00323 #if DEBUG_EXPERIMENT
00324           fprintf (stderr,
00325                    "experiment_open_json: experiment=%s template%u=%s\n",
00326                    experiment->nexperiments, name, template[i]);
00327 #endif
00328           experiment->template[i] = g_strdup (name);
00329           ++experiment->ninputs;
00330         }
00331       else if (ninputs && ninputs > i)
```

```
00332         {
00333           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334           experiment_error (experiment, buffer);
00335           goto exit_on_error;
00336         }
00337       else
00338         break;
00339     }
00340
00341 #if DEBUG_EXPERIMENT
00342   fprintf (stderr, "experiment_open_json: end\n");
00343 #endif
00344   return 1;
00345
00346 exit_on_error:
00347   experiment_free (experiment, INPUT_TYPE_JSON);
00348 #if DEBUG_EXPERIMENT
00349   fprintf (stderr, "experiment_open_json: end\n");
00350 #endif
00351   return 0;
00352 }
```

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)

  Function to create a new Experiment struct.
- void experiment_free (Experiment *experiment, unsigned int type)

  Function to free the memory of an Experiment struct.
- void experiment_error (Experiment *experiment, char *message)

  Function to print a message error opening an Experiment struct.
- int experiment_open_xml (Experiment *experiment, xmlNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.
- int experiment_open_json (Experiment *experiment, JsonNode *node, unsigned int ninputs)

  Function to open the Experiment struct on a XML node.

Variables

- const char * template [MAX_NINPUTS]

  Array of xmlChar strings with template 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 121 of file experiment.c.

```
00122 {
00123   char buffer[64];
00124   if (!experiment->name)
00125     snprintf (buffer, 64, "%s: %s", _("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128               experiment->name, message);
00129   error_message = g_strdup (buffer);
00130 }
```

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 88 of file experiment.c.

```
00089 {
00090   unsigned int i;
00091 #if DEBUG_EXPERIMENT
00092   fprintf (stderr, "experiment_free: start\n");
00093 #endif
00094   if (type == INPUT_TYPE_XML)
00095     {
00096       for (i = 0; i < experiment->ninputs; ++i)
00097         xmlFree (experiment->template[i]);
00098       xmlFree (experiment->name);
00099     }
00100   else
00101     {
00102       for (i = 0; i < experiment->ninputs; ++i)
00103         g_free (experiment->template[i]);
00104       g_free (experiment->name);
00105     }
00106   experiment->ninputs = 0;
00107 #if DEBUG_EXPERIMENT
00108   fprintf (stderr, "experiment_free: end\n");
00109 #endif
00110 }
```

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 64 of file experiment.c.

```
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new: start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
```

```
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->template[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new: end\n");
00076 #endif
00077 }
```

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.

Parameters

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

Returns

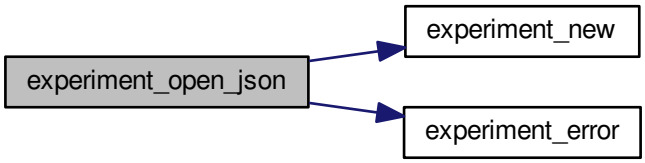
1 on success, 0 on error.

Definition at line 254 of file experiment.c.

```
00256 {
00257   char buffer[64];
00258   JsonObject *object;
00259   const char *name;
00260   int error_code;
00261   unsigned int i;
00262
00263 #if DEBUG_EXPERIMENT
00264   fprintf (stderr, "experiment_open_json: start\n");
00265 #endif
00266
00267   // Resetting experiment data
00268   experiment_new (experiment);
00269
00270   // Getting JSON object
00271   object = json_node_get_object (node);
00272
00273   // Reading the experimental data
00274   name = json_object_get_string_member (object, LABEL_NAME);
00275   if (!name)
00276     {
00277       experiment_error (experiment, _("no data file name"));
00278       goto exit_on_error;
00279     }
00280   experiment->name = g_strdup (name);
00281 #if DEBUG_EXPERIMENT
00282   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283 #endif
00284   experiment->weight
00285     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00286                                           &error_code);
00287   if (error_code)
00288     {
00289       experiment_error (experiment, _("bad weight"));
00290       goto exit_on_error;
```

```
00291     }
00292 #if DEBUG_EXPERIMENT
00293   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294 #endif
00295   name = json_object_get_string_member (object, template[0]);
00296   if (name)
00297     {
00298 #if DEBUG_EXPERIMENT
00299       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300                name, template[0]);
00301 #endif
00302       ++experiment->ninputs;
00303     }
00304   else
00305     {
00306       experiment_error (experiment, _("no template"));
00307       goto exit_on_error;
00308     }
00309   experiment->template[0] = g_strdup (name);
00310   for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312 #if DEBUG_EXPERIMENT
00313       fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314 #endif
00315       if (json_object_get_member (object, template[i]))
00316         {
00317           if (ninputs && ninputs <= i)
00318             {
00319               experiment_error (experiment, _("bad templates number"));
00320               goto exit_on_error;
00321             }
00322           name = json_object_get_string_member (object, template[i]);
00323 #if DEBUG_EXPERIMENT
00324           fprintf (stderr,
00325                    "experiment_open_json: experiment=%s template%u=%s\n",
00326                    experiment->nexperiments, name, template[i]);
00327 #endif
00328           experiment->template[i] = g_strdup (name);
00329           ++experiment->ninputs;
00330         }
00331       else if (ninputs && ninputs > i)
00332         {
00333           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334           experiment_error (experiment, buffer);
00335           goto exit_on_error;
00336         }
00337       else
00338         break;
00339     }
00340
00341 #if DEBUG_EXPERIMENT
00342   fprintf (stderr, "experiment_open_json: end\n");
00343 #endif
00344   return 1;
00345
00346 exit_on_error:
00347   experiment_free (experiment, INPUT_TYPE_JSON);
00348 #if DEBUG_EXPERIMENT
00349   fprintf (stderr, "experiment_open_json: end\n");
00350 #endif
00351   return 0;
00352 }
```

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.

Parameters

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

Returns

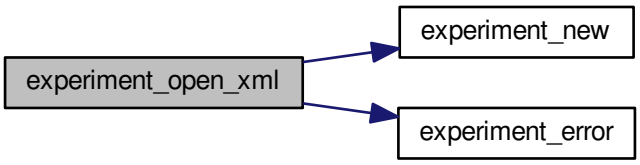
1 on success, 0 on error.

Definition at line 145 of file experiment.c.

```
00147 {
00148   char buffer[64];
00149   int error_code;
00150   unsigned int i;
00151
00152 #if DEBUG_EXPERIMENT
00153   fprintf (stderr, "experiment_open_xml: start\n");
00154 #endif
00155
00156   // Resetting experiment data
00157   experiment_new (experiment);
00158
00159   // Reading the experimental data
00160   experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161   if (!experiment->name)
00162     {
00163       experiment_error (experiment, _("no data file name"));
00164       goto exit_on_error;
00165     }
00166 #if DEBUG_EXPERIMENT
00167   fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168 #endif
00169   experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00172                                      &error_code);
00173   if (error_code)
00174     {
00175       experiment_error (experiment, _("bad weight"));
00176       goto exit_on_error;
00177     }
00178 #if DEBUG_EXPERIMENT
00179   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180 #endif
00181   experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183   if (experiment->template[0])
00184     {
00185 #if DEBUG_EXPERIMENT
00186       fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00187                experiment->name, template[0]);
00188 #endif
00189       ++experiment->ninputs;
00190     }
00191   else
```

```
00192     {
00193       experiment_error (experiment, _("no template"));
00194       goto exit_on_error;
00195     }
00196   for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198 #if DEBUG_EXPERIMENT
00199       fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200 #endif
00201       if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203           if (ninputs && ninputs <= i)
00204             {
00205               experiment_error (experiment, _("bad templates number"));
00206               goto exit_on_error;
00207             }
00208           experiment->template[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210 #if DEBUG_EXPERIMENT
00211           fprintf (stderr,
00212                    "experiment_open_xml: experiment=%s template%u=%s\n",
00213                    experiment->nexperiments, experiment->name,
00214                    experiment->template[i]);
00215 #endif
00216           ++experiment->ninputs;
00217         }
00218       else if (ninputs && ninputs > i)
00219         {
00220           snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00221           experiment_error (experiment, buffer);
00222           goto exit_on_error;
00223         }
00224       else
00225         break;
00226     }
00227
00228 #if DEBUG_EXPERIMENT
00229   fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231   return 1;
00232
00233 exit_on_error:
00234   experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236   fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238   return 0;
00239 }
```

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.
```

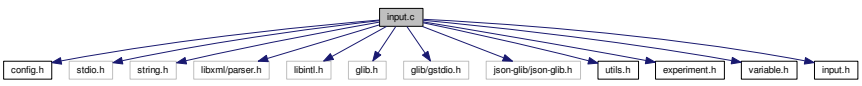
```
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     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 *template[MAX_NINPUTS];
00049   double weight;
00050   unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const char *template[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 ()

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

  Function to free the memory of the input file data.
- void input_error (char *message)

  Function to print an error message opening an Input struct.
- int input_open_xml (xmlDoc *doc)

  Function to open the input file in XML format.
- int input_open_json (JsonParser *parser)

  Function to open the input file in JSON format.
- int input_open (char *filename)

  Function to open the input file.

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 124 of file input.c.

```
00125 {
00126   char buffer[64];
00127   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128   error_message = g_strdup (buffer);
00129 }
```

4.7.2.2 input_open()

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

Function to open the input file.

Parameters

filename	Input data file name.

Returns

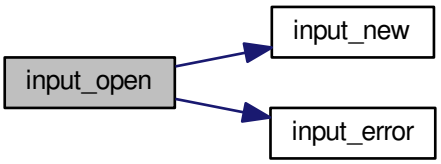
1_on_success, 0_on_error.

Definition at line 952 of file input.c.

```
00953 {
00954   xmlDoc *doc;
00955   JsonParser *parser;
00956
00957 #if DEBUG_INPUT
00958   fprintf (stderr, "input_open: start\n");
00959 #endif
00960
00961   // Resetting input data
00962   input_new ();
00963
00964   // Opening input file
00965 #if DEBUG_INPUT
00966   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00967   fprintf (stderr, "input_open: trying XML format\n");
00968 #endif
00969   doc = xmlParseFile (filename);
00970   if (!doc)
00971     {
00972 #if DEBUG_INPUT
00973       fprintf (stderr, "input_open: trying JSON format\n");
00974 #endif
00975       parser = json_parser_new ();
00976       if (!json_parser_load_from_file (parser, filename, NULL))
00977         {
00978           input_error (_("Unable to parse the input file"));
00979           goto exit_on_error;
00980         }
00981       if (!input_open_json (parser))
00982         goto exit_on_error;
00983     }
00984   else if (!input_open_xml (doc))
```

```
00985     goto exit_on_error;
00986
00987   // Getting the working directory
00988   input->directory = g_path_get_dirname (filename);
00989   input->name = g_path_get_basename (filename);
00990
00991 #if DEBUG_INPUT
00992   fprintf (stderr, "input_open: end\n");
00993 #endif
00994   return 1;
00995
00996 exit_on_error:
00997   show_error (error_message);
00998   g_free (error_message);
00999   input_free ();
01000 #if DEBUG_INPUT
01001   fprintf (stderr, "input_open: end\n");
01002 #endif
01003   return 0;
01004 }
```

Here is the call graph for this function:



4.7.2.3 input_open_json()

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

Function to open the input file in JSON format.

Parameters

parser	JsonParser struct.

Returns

1_on_success, 0_on_error.

Definition at line 562 of file input.c.

```
00563 {
00564   JsonNode *node, *child;
00565   JsonObject *object;
00566   JsonArray *array;
```

```
00567   const char *buffer;
00568   int error_code;
00569   unsigned int i, n;
00570
00571 #if DEBUG_INPUT
00572   fprintf (stderr, "input_open_json: start\n");
00573 #endif
00574
00575   // Resetting input data
00576   input->type = INPUT_TYPE_JSON;
00577
00578   // Getting the root node
00579 #if DEBUG_INPUT
00580   fprintf (stderr, "input_open_json: getting the root node\n");
00581 #endif
00582   node = json_parser_get_root (parser);
00583   object = json_node_get_object (node);
00584
00585   // Getting result and variables file names
00586   if (!input->result)
00587     {
00588       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589       if (!buffer)
00590         buffer = result_name;
00591       input->result = g_strdup (buffer);
00592     }
00593   else
00594     input->result = g_strdup (result_name);
00595   if (!input->variables)
00596     {
00597       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598       if (!buffer)
00599         buffer = variables_name;
00600       input->variables = g_strdup (buffer);
00601     }
00602   else
00603     input->variables = g_strdup (variables_name);
00604
00605   // Opening simulator program name
00606   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607   if (!buffer)
00608     {
00609       input_error (_("Bad simulator program"));
00610       goto exit_on_error;
00611     }
00612   input->simulator = g_strdup (buffer);
00613
00614   // Opening evaluator program name
00615   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616   if (buffer)
00617     input->evaluator = g_strdup (buffer);
00618
00619   // Obtaining pseudo-random numbers generator seed
00620   input->seed
00621     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00622                                          DEFAULT_RANDOM_SEED, &error_code);
00623   if (error_code)
00624     {
00625       input_error (_("Bad pseudo-random numbers generator seed"));
00626       goto exit_on_error;
00627     }
00628
00629   // Opening algorithm
00630   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633       input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635       // Obtaining simulations number
00636       input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00638       if (error_code)
00639         {
00640           input_error (_("Bad simulations number"));
00641           goto exit_on_error;
00642         }
00643     }
00644   else if (!strcmp (buffer, LABEL_SWEEP))
00645     input->algorithm = ALGORITHM_SWEEP;
00646   else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648       input->algorithm = ALGORITHM_GENETIC;
00649
00650       // Obtaining population
00651       if (json_object_get_member (object, LABEL_NPOPULATION))
```

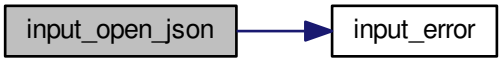
```
00652         {
00653           input->nsimulations
00654             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00655           if (error_code || input->nsimulations < 3)
00656             {
00657               input_error (_("Invalid population number"));
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           input_error (_("No population number"));
00664           goto exit_on_error;
00665         }
00666
00667       // Obtaining generations
00668       if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670           input->niterations
00671             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00672           if (error_code || !input->niterations)
00673             {
00674               input_error (_("Invalid generations number"));
00675               goto exit_on_error;
00676             }
00677         }
00678       else
00679         {
00680           input_error (_("No generations number"));
00681           goto exit_on_error;
00682         }
00683
00684       // Obtaining mutation probability
00685       if (json_object_get_member (object, LABEL_MUTATION))
00686         {
00687           input->mutation_ratio
00688             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00689           if (error_code || input->mutation_ratio < 0.
00690               || input->mutation_ratio >= 1.)
00691             {
00692               input_error (_("Invalid mutation probability"));
00693               goto exit_on_error;
00694             }
00695         }
00696       else
00697         {
00698           input_error (_("No mutation probability"));
00699           goto exit_on_error;
00700         }
00701
00702       // Obtaining reproduction probability
00703       if (json_object_get_member (object, LABEL_REPRODUCTION))
00704         {
00705           input->reproduction_ratio
00706             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00707           if (error_code || input->reproduction_ratio < 0.
00708               || input->reproduction_ratio >= 1.0)
00709             {
00710               input_error (_("Invalid reproduction probability"));
00711               goto exit_on_error;
00712             }
00713         }
00714       else
00715         {
00716           input_error (_("No reproduction probability"));
00717           goto exit_on_error;
00718         }
00719
00720       // Obtaining adaptation probability
00721       if (json_object_get_member (object, LABEL_ADAPTATION))
00722         {
00723           input->adaptation_ratio
00724             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00725           if (error_code || input->adaptation_ratio < 0.
00726               || input->adaptation_ratio >= 1.)
00727             {
00728               input_error (_("Invalid adaptation probability"));
00729               goto exit_on_error;
00730             }
00731         }
00732       else
00733         {
```

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

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

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

Here is the call graph for this function:



4.7.2.4 input_open_xml()

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

Function to open the input file in XML format.

Parameters

doc	xmlDoc struct.

Returns

1_on_success, 0_on_error.

Definition at line 139 of file input.c.

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

```
00207   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209     {
00210       input->algorithm = ALGORITHM_MONTE_CARLO;
00211
00212       // Obtaining simulations number
00213       input->nsimulations
00214         = xml_node_get_int (node, (const xmlChar *)
    LABEL_NSIMULATIONS,
00215                             &error_code);
00216       if (error_code)
00217         {
00218           input_error (_("Bad simulations number"));
00219           goto exit_on_error;
00220         }
00221     }
00222   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223     input->algorithm = ALGORITHM_SWEEP;
00224   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226       input->algorithm = ALGORITHM_GENETIC;
00227
00228       // Obtaining population
00229       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231           input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00233                                  &error_code);
00234           if (error_code || input->nsimulations < 3)
00235             {
00236               input_error (_("Invalid population number"));
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (_("No population number"));
00243           goto exit_on_error;
00244         }
00245
00246       // Obtaining generations
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248         {
00249           input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251                                  &error_code);
00252           if (error_code || !input->niterations)
00253             {
00254               input_error (_("Invalid generations number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           input_error (_("No generations number"));
00261           goto exit_on_error;
00262         }
00263
00264       // Obtaining mutation probability
00265       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266         {
00267           input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                                   &error_code);
00270           if (error_code || input->mutation_ratio < 0.
00271               || input->mutation_ratio >= 1.)
00272             {
00273               input_error (_("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
00277       else
00278         {
00279           input_error (_("No mutation probability"));
00280           goto exit_on_error;
00281         }
00282
00283       // Obtaining reproduction probability
00284       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285         {
00286           input->reproduction_ratio
00287             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                                   &error_code);
00289           if (error_code || input->reproduction_ratio < 0.
00290               || input->reproduction_ratio >= 1.0)
00291             {
00292               input_error (_("Invalid reproduction probability"));
```

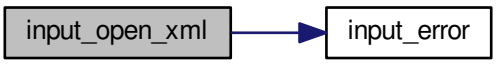
```
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (_("No reproduction probability"));
00299           goto exit_on_error;
00300         }
00301
00302       // Obtaining adaptation probability
00303       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304         {
00305           input->adaptation_ratio
00306             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                                   &error_code);
00308           if (error_code || input->adaptation_ratio < 0.
00309               || input->adaptation_ratio >= 1.)
00310             {
00311               input_error (_("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (_("No adaptation probability"));
00318           goto exit_on_error;
00319         }
00320
00321       // Checking survivals
00322       i = input->mutation_ratio * input->nsimulations;
00323       i += input->reproduction_ratio * input->
    nsimulations;
00324       i += input->adaptation_ratio * input->
    nsimulations;
00325       if (i > input->nsimulations - 2)
00326         {
00327           input_error
00328             (_("No enough survival entities to reproduce the population"));
00329           goto exit_on_error;
00330         }
00331     }
00332   else
00333     {
00334       input_error (_("Unknown algorithm"));
00335       goto exit_on_error;
00336     }
00337   xmlFree (buffer);
00338   buffer = NULL;
00339
00340   if (input->algorithm == ALGORITHM_MONTE_CARLO
00341       || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344       // Obtaining iterations number
00345       input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NITERATIONS,
00347                              &error_code);
00348       if (error_code == 1)
00349         input->niterations = 1;
00350       else if (error_code)
00351         {
00352           input_error (_("Bad iterations number"));
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining best number
00357       input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_NBEST,
00359                                           1, &error_code);
00360       if (error_code || !input->nbest)
00361         {
00362           input_error (_("Invalid best number"));
00363           goto exit_on_error;
00364         }
00365       if (input->nbest > input->nsimulations)
00366         {
00367           input_error (_("Best number higher than simulations number"));
00368           goto exit_on_error;
00369         }
00370
00371       // Obtaining tolerance
00372       input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                            (const xmlChar *) LABEL_TOLERANCE,
00375                                            0., &error_code);
```

```
00376       if (error_code || input->tolerance < 0.)
00377         {
00378           input_error (_("Invalid tolerance"));
00379           goto exit_on_error;
00380         }
00381
00382       // Getting direction search method parameters
00383       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384         {
00385           input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00387                                &error_code);
00388           if (error_code)
00389             {
00390               input_error (_("Invalid steps number"));
00391               goto exit_on_error;
00392             }
00393           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->direction = DIRECTION_METHOD_COORDINATES;
00396           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397             {
00398               input->direction = DIRECTION_METHOD_RANDOM;
00399               input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NESTIMATES,
00401                                      &error_code);
00402               if (error_code || !input->nestimates)
00403                 {
00404                   input_error (_("Invalid estimates number"));
00405                   goto exit_on_error;
00406                 }
00407             }
00408           else
00409             {
00410               input_error
00411                 (_("Unknown method to estimate the direction search"));
00412               goto exit_on_error;
00413             }
00414           xmlFree (buffer);
00415           buffer = NULL;
00416           input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                                (const xmlChar *)
00419                                                LABEL_RELAXATION,
00420                                                DEFAULT_RELAXATION,
00421                                                &error_code);
00422           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00423             {
00424               input_error (_("Invalid relaxation parameter"));
00425               goto exit_on_error;
00426             }
00427         }
00428       else
00429         input->nsteps = 0;
00430     }
00431   // Obtaining the threshold
00432   input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_THRESHOLD,
00434                                      0., &error_code);
00435   if (error_code)
00436     {
00437       input_error (_("Invalid threshold"));
00438       goto exit_on_error;
00439     }
00440
00441   // Reading the experimental data
00442   for (child = node->children; child; child = child->next)
00443     {
00444       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448                input->nexperiments);
00449 #endif
00450       input->experiment = (Experiment *)
00451         g_realloc (input->experiment,
00452                    (1 + input->nexperiments) * sizeof (
    Experiment));
00453       if (!input->nexperiments)
00454         {
00455           if (!experiment_open_xml (input->experiment, child, 0))
00456             goto exit_on_error;
00457         }
00458       else
```

```
00459         {
00460           if (!experiment_open_xml (input->experiment +
    input->nexperiments,
00461                                     child, input->experiment->
    ninputs))
00462             goto exit_on_error;
00463         }
00464       ++input->nexperiments;
00465 #if DEBUG_INPUT
00466       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00467                input->nexperiments);
00468 #endif
00469     }
00470   if (!input->nexperiments)
00471     {
00472       input_error (_("No optimization experiments"));
00473       goto exit_on_error;
00474     }
00475   buffer = NULL;
00476
00477   // Reading the variables data
00478   for (; child; child = child->next)
00479     {
00480 #if DEBUG_INPUT
00481       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00482 #endif
00483       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00484         {
00485           snprintf (buffer2, 64, "%s %u: %s",
00486                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00487           input_error (buffer2);
00488           goto exit_on_error;
00489         }
00490       input->variable = (Variable *)
00491         g_realloc (input->variable,
00492                    (1 + input->nvariables) * sizeof (Variable));
00493       if (!variable_open_xml (input->variable +
    input->nvariables, child,
00494                               input->algorithm, input->nsteps))
00495         goto exit_on_error;
00496       ++input->nvariables;
00497     }
00498   if (!input->nvariables)
00499     {
00500       input_error (_("No optimization variables"));
00501       goto exit_on_error;
00502     }
00503   buffer = NULL;
00504
00505   // Obtaining the error norm
00506   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507     {
00508       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510         input->norm = ERROR_NORM_EUCLIDIAN;
00511       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512         input->norm = ERROR_NORM_MAXIMUM;
00513       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514         {
00515           input->norm = ERROR_NORM_P;
00516           input->p
00517             =
00518             xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00519           if (!error_code)
00520             {
00521               input_error (_("Bad P parameter"));
00522               goto exit_on_error;
00523             }
00524         }
00525       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526         input->norm = ERROR_NORM_TAXICAB;
00527       else
00528         {
00529           input_error (_("Unknown error norm"));
00530           goto exit_on_error;
00531         }
00532       xmlFree (buffer);
00533     }
00534   else
00535     input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537   // Closing the XML document
00538   xmlFreeDoc (doc);
00539
00540 #if DEBUG_INPUT
00541   fprintf (stderr, "input_open_xml: end\n");
00542 #endif
```

```
00543   return 1;
00544
00545 exit_on_error:
00546   xmlFree (buffer);
00547   xmlFreeDoc (doc);
00548 #if DEBUG_INPUT
00549   fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551   return 0;
00552 }
```

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 2012-2016, 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
00063 void
00064 input_new ()
00065 {
00066 #if DEBUG_INPUT
00067   fprintf (stderr, "input_new: start\n");
00068 #endif
00069   input->nvariables = input->nexperiments = input->nsteps = 0;
00070   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00071   input->experiment = NULL;
00072   input->variable = NULL;
00073 #if DEBUG_INPUT
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085   unsigned int i;
00086 #if DEBUG_INPUT
00087   fprintf (stderr, "input_free: start\n");
00088 #endif
00089   g_free (input->name);
00090   g_free (input->directory);
00091   for (i = 0; i < input->nexperiments; ++i)
00092     experiment_free (input->experiment + i, input->type);
00093   for (i = 0; i < input->nvariables; ++i)
00094     variable_free (input->variable + i, input->type);
00095   g_free (input->experiment);
00096   g_free (input->variable);
00097   if (input->type == INPUT_TYPE_XML)
00098     {
00099       xmlFree (input->evaluator);
00100       xmlFree (input->simulator);
00101       xmlFree (input->result);
00102       xmlFree (input->variables);
00103     }
00104   else
00105     {
00106       g_free (input->evaluator);
00107       g_free (input->simulator);
00108       g_free (input->result);
00109       g_free (input->variables);
00110     }
00111   input->nexperiments = input->nvariables = input->nsteps = 0;
00112 #if DEBUG_INPUT
00113   fprintf (stderr, "input_free: end\n");
00114 #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126   char buffer[64];
00127   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128   error_message = g_strdup (buffer);
00129 }
00130
00138 int
00139 input_open_xml (xmlDoc * doc)
00140 {
00141   char buffer2[64];
00142   xmlNode *node, *child;
00143   xmlChar *buffer;
00144   int error_code;
00145   unsigned int i;
00146
00147 #if DEBUG_INPUT
00148   fprintf (stderr, "input_open_xml: start\n");
00149 #endif
00150
00151   // Resetting input data
00152   buffer = NULL;
00153   input->type = INPUT_TYPE_XML;
00154
00155   // Getting the root node
00156 #if DEBUG_INPUT
00157   fprintf (stderr, "input_open_xml: getting the root node\n");
00158 #endif
00159   node = xmlDocGetRootElement (doc);
00160   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162       input_error (_("Bad root XML node"));
```

```
00163       goto exit_on_error;
00164     }
00165
00166   // Getting result and variables file names
00167   if (!input->result)
00168     {
00169       input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171       if (!input->result)
00172         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173     }
00174   if (!input->variables)
00175     {
00176       input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178       if (!input->variables)
00179         input->variables =
00180           (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183   // Opening simulator program name
00184   input->simulator =
00185     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186   if (!input->simulator)
00187     {
00188       input_error (_("Bad simulator program"));
00189       goto exit_on_error;
00190     }
00191
00192   // Opening evaluator program name
00193   input->evaluator =
00194     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196   // Obtaining pseudo-random numbers generator seed
00197   input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_SEED,
00199                                       DEFAULT_RANDOM_SEED, &error_code);
00200   if (error_code)
00201     {
00202       input_error (_("Bad pseudo-random numbers generator seed"));
00203       goto exit_on_error;
00204     }
00205
00206   // Opening algorithm
00207   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209     {
00210       input->algorithm = ALGORITHM_MONTE_CARLO;
00211
00212       // Obtaining simulations number
00213       input->nsimulations
00214         = xml_node_get_int (node, (const xmlChar *)
      LABEL_NSIMULATIONS,
00215                             &error_code);
00216       if (error_code)
00217         {
00218           input_error (_("Bad simulations number"));
00219           goto exit_on_error;
00220         }
00221     }
00222   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223     input->algorithm = ALGORITHM_SWEEP;
00224   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226       input->algorithm = ALGORITHM_GENETIC;
00227
00228       // Obtaining population
00229       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231           input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00233                                  &error_code);
00234           if (error_code || input->nsimulations < 3)
00235             {
00236               input_error (_("Invalid population number"));
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (_("No population number"));
00243           goto exit_on_error;
00244         }
00245
00246       // Obtaining generations
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
```

```
00248         {
00249           input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251                                  &error_code);
00252           if (error_code || !input->niterations)
00253             {
00254               input_error (_("Invalid generations number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           input_error (_("No generations number"));
00261           goto exit_on_error;
00262         }
00263
00264       // Obtaining mutation probability
00265       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266         {
00267           input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                                   &error_code);
00270           if (error_code || input->mutation_ratio < 0.
00271               || input->mutation_ratio >= 1.)
00272             {
00273               input_error (_("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
00277       else
00278         {
00279           input_error (_("No mutation probability"));
00280           goto exit_on_error;
00281         }
00282
00283       // Obtaining reproduction probability
00284       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285         {
00286           input->reproduction_ratio
00287             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                                   &error_code);
00289           if (error_code || input->reproduction_ratio < 0.
00290               || input->reproduction_ratio >= 1.0)
00291             {
00292               input_error (_("Invalid reproduction probability"));
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (_("No reproduction probability"));
00299           goto exit_on_error;
00300         }
00301
00302       // Obtaining adaptation probability
00303       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304         {
00305           input->adaptation_ratio
00306             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                                   &error_code);
00308           if (error_code || input->adaptation_ratio < 0.
00309               || input->adaptation_ratio >= 1.)
00310             {
00311               input_error (_("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (_("No adaptation probability"));
00318           goto exit_on_error;
00319         }
00320
00321       // Checking survivals
00322       i = input->mutation_ratio * input->nsimulations;
00323       i += input->reproduction_ratio * input->nsimulations;
00324       i += input->adaptation_ratio * input->nsimulations;
00325       if (i > input->nsimulations - 2)
00326         {
00327           input_error
00328             (_("No enough survival entities to reproduce the population"));
00329           goto exit_on_error;
00330         }
00331     }
00332   else
00333     {
00334       input_error (_("Unknown algorithm"));
```

```
00335       goto exit_on_error;
00336     }
00337   xmlFree (buffer);
00338   buffer = NULL;
00339
00340   if (input->algorithm == ALGORITHM_MONTE_CARLO
00341       || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344       // Obtaining iterations number
00345       input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00347                              &error_code);
00348       if (error_code == 1)
00349         input->niterations = 1;
00350       else if (error_code)
00351         {
00352           input_error (_("Bad iterations number"));
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining best number
00357       input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_NBEST,
00359                                           1, &error_code);
00360       if (error_code || !input->nbest)
00361         {
00362           input_error (_("Invalid best number"));
00363           goto exit_on_error;
00364         }
00365       if (input->nbest > input->nsimulations)
00366         {
00367           input_error (_("Best number higher than simulations number"));
00368           goto exit_on_error;
00369         }
00370
00371       // Obtaining tolerance
00372       input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                            (const xmlChar *) LABEL_TOLERANCE,
00375                                            0., &error_code);
00376       if (error_code || input->tolerance < 0.)
00377         {
00378           input_error (_("Invalid tolerance"));
00379           goto exit_on_error;
00380         }
00381
00382       // Getting direction search method parameters
00383       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384         {
00385           input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00387                                &error_code);
00388           if (error_code)
00389             {
00390               input_error (_("Invalid steps number"));
00391               goto exit_on_error;
00392             }
00393           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->direction = DIRECTION_METHOD_COORDINATES;
00396           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397             {
00398               input->direction = DIRECTION_METHOD_RANDOM;
00399               input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00401                                      &error_code);
00402               if (error_code || !input->nestimates)
00403                 {
00404                   input_error (_("Invalid estimates number"));
00405                   goto exit_on_error;
00406                 }
00407             }
00408           else
00409             {
00410               input_error
00411                 (_("Unknown method to estimate the direction search"));
00412               goto exit_on_error;
00413             }
00414           xmlFree (buffer);
00415           buffer = NULL;
00416           input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                                (const xmlChar *)
```

```
00419                                                 LABEL_RELAXATION,
00420                                                 DEFAULT_RELAXATION,
00421                                                 &error_code);
00422           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00423             {
00424               input_error (_("Invalid relaxation parameter"));
00425               goto exit_on_error;
00426             }
00427         }
00428       else
00429         input->nsteps = 0;
00430     }
00431   // Obtaining the threshold
00432   input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00434                                      0., &error_code);
00435   if (error_code)
00436     {
00437       input_error (_("Invalid threshold"));
00438       goto exit_on_error;
00439     }
00440
00441   // Reading the experimental data
00442   for (child = node->children; child; child = child->next)
00443     {
00444       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448                input->nexperiments);
00449 #endif
00450       input->experiment = (Experiment *)
00451         g_realloc (input->experiment,
00452                    (1 + input->nexperiments) * sizeof (Experiment));
00453       if (!input->nexperiments)
00454         {
00455           if (!experiment_open_xml (input->experiment, child, 0))
00456             goto exit_on_error;
00457         }
00458       else
00459         {
00460           if (!experiment_open_xml (input->experiment + input->
      nexperiments,
00461                                     child, input->experiment->ninputs))
00462             goto exit_on_error;
00463         }
00464       ++input->nexperiments;
00465 #if DEBUG_INPUT
00466       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00467                input->nexperiments);
00468 #endif
00469     }
00470   if (!input->nexperiments)
00471     {
00472       input_error (_("No optimization experiments"));
00473       goto exit_on_error;
00474     }
00475   buffer = NULL;
00476
00477   // Reading the variables data
00478   for (; child; child = child->next)
00479     {
00480 #if DEBUG_INPUT
00481       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00482 #endif
00483       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00484         {
00485           snprintf (buffer2, 64, "%s %u: %s",
00486                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00487           input_error (buffer2);
00488           goto exit_on_error;
00489         }
00490       input->variable = (Variable *)
00491         g_realloc (input->variable,
00492                    (1 + input->nvariables) * sizeof (Variable));
00493       if (!variable_open_xml (input->variable + input->
      nvariables, child,
00494                               input->algorithm, input->nsteps))
00495         goto exit_on_error;
00496       ++input->nvariables;
00497     }
00498   if (!input->nvariables)
00499     {
00500       input_error (_("No optimization variables"));
00501       goto exit_on_error;
```

```
00502     }
00503   buffer = NULL;
00504
00505   // Obtaining the error norm
00506   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507     {
00508       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510         input->norm = ERROR_NORM_EUCLIDIAN;
00511       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512         input->norm = ERROR_NORM_MAXIMUM;
00513       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514         {
00515           input->norm = ERROR_NORM_P;
00516           input->p
00517             =
00518             xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00519           if (!error_code)
00520             {
00521               input_error (_("Bad P parameter"));
00522               goto exit_on_error;
00523             }
00524         }
00525       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526         input->norm = ERROR_NORM_TAXICAB;
00527       else
00528         {
00529           input_error (_("Unknown error norm"));
00530           goto exit_on_error;
00531         }
00532       xmlFree (buffer);
00533     }
00534   else
00535     input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537   // Closing the XML document
00538   xmlFreeDoc (doc);
00539
00540 #if DEBUG_INPUT
00541   fprintf (stderr, "input_open_xml: end\n");
00542 #endif
00543   return 1;
00544
00545 exit_on_error:
00546   xmlFree (buffer);
00547   xmlFreeDoc (doc);
00548 #if DEBUG_INPUT
00549   fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551   return 0;
00552 }
00553
00561 int
00562 input_open_json (JsonParser * parser)
00563 {
00564   JsonNode *node, *child;
00565   JsonObject *object;
00566   JsonArray *array;
00567   const char *buffer;
00568   int error_code;
00569   unsigned int i, n;
00570
00571 #if DEBUG_INPUT
00572   fprintf (stderr, "input_open_json: start\n");
00573 #endif
00574
00575   // Resetting input data
00576   input->type = INPUT_TYPE_JSON;
00577
00578   // Getting the root node
00579 #if DEBUG_INPUT
00580   fprintf (stderr, "input_open_json: getting the root node\n");
00581 #endif
00582   node = json_parser_get_root (parser);
00583   object = json_node_get_object (node);
00584
00585   // Getting result and variables file names
00586   if (!input->result)
00587     {
00588       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589       if (!buffer)
00590         buffer = result_name;
00591       input->result = g_strdup (buffer);
00592     }
00593   else
00594     input->result = g_strdup (result_name);
00595   if (!input->variables)
```

```
00596     {
00597       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598       if (!buffer)
00599         buffer = variables_name;
00600       input->variables = g_strdup (buffer);
00601     }
00602   else
00603     input->variables = g_strdup (variables_name);
00604
00605   // Opening simulator program name
00606   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607   if (!buffer)
00608     {
00609       input_error (_("Bad simulator program"));
00610       goto exit_on_error;
00611     }
00612   input->simulator = g_strdup (buffer);
00613
00614   // Opening evaluator program name
00615   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616   if (buffer)
00617     input->evaluator = g_strdup (buffer);
00618
00619   // Obtaining pseudo-random numbers generator seed
00620   input->seed
00621     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00622                                          DEFAULT_RANDOM_SEED, &error_code);
00623   if (error_code)
00624     {
00625       input_error (_("Bad pseudo-random numbers generator seed"));
00626       goto exit_on_error;
00627     }
00628
00629   // Opening algorithm
00630   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633       input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635       // Obtaining simulations number
00636       input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00638       if (error_code)
00639         {
00640           input_error (_("Bad simulations number"));
00641           goto exit_on_error;
00642         }
00643     }
00644   else if (!strcmp (buffer, LABEL_SWEEP))
00645     input->algorithm = ALGORITHM_SWEEP;
00646   else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648       input->algorithm = ALGORITHM_GENETIC;
00649
00650       // Obtaining population
00651       if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653           input->nsimulations
00654             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00655           if (error_code || input->nsimulations < 3)
00656             {
00657               input_error (_("Invalid population number"));
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           input_error (_("No population number"));
00664           goto exit_on_error;
00665         }
00666
00667       // Obtaining generations
00668       if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670           input->niterations
00671             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00672           if (error_code || !input->niterations)
00673             {
00674               input_error (_("Invalid generations number"));
00675               goto exit_on_error;
00676             }
00677         }
00678       else
```

```
00679         {
00680           input_error (_("No generations number"));
00681           goto exit_on_error;
00682         }
00683
00684       // Obtaining mutation probability
00685       if (json_object_get_member (object, LABEL_MUTATION))
00686         {
00687           input->mutation_ratio
00688             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00689           if (error_code || input->mutation_ratio < 0.
00690               || input->mutation_ratio >= 1.)
00691             {
00692               input_error (_("Invalid mutation probability"));
00693               goto exit_on_error;
00694             }
00695         }
00696       else
00697         {
00698           input_error (_("No mutation probability"));
00699           goto exit_on_error;
00700         }
00701
00702       // Obtaining reproduction probability
00703       if (json_object_get_member (object, LABEL_REPRODUCTION))
00704         {
00705           input->reproduction_ratio
00706             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00707           if (error_code || input->reproduction_ratio < 0.
00708               || input->reproduction_ratio >= 1.0)
00709             {
00710               input_error (_("Invalid reproduction probability"));
00711               goto exit_on_error;
00712             }
00713         }
00714       else
00715         {
00716           input_error (_("No reproduction probability"));
00717           goto exit_on_error;
00718         }
00719
00720       // Obtaining adaptation probability
00721       if (json_object_get_member (object, LABEL_ADAPTATION))
00722         {
00723           input->adaptation_ratio
00724             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00725           if (error_code || input->adaptation_ratio < 0.
00726               || input->adaptation_ratio >= 1.)
00727             {
00728               input_error (_("Invalid adaptation probability"));
00729               goto exit_on_error;
00730             }
00731         }
00732       else
00733         {
00734           input_error (_("No adaptation probability"));
00735           goto exit_on_error;
00736         }
00737
00738       // Checking survivals
00739       i = input->mutation_ratio * input->nsimulations;
00740       i += input->reproduction_ratio * input->nsimulations;
00741       i += input->adaptation_ratio * input->nsimulations;
00742       if (i > input->nsimulations - 2)
00743         {
00744           input_error
00745             (_("No enough survival entities to reproduce the population"));
00746           goto exit_on_error;
00747         }
00748     }
00749   else
00750     {
00751       input_error (_("Unknown algorithm"));
00752       goto exit_on_error;
00753     }
00754
00755   if (input->algorithm == ALGORITHM_MONTE_CARLO
00756       || input->algorithm == ALGORITHM_SWEEP)
00757     {
00758
00759       // Obtaining iterations number
00760       input->niterations
00761         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
```

```
00762       if (error_code == 1)
00763         input->niterations = 1;
00764       else if (error_code)
00765         {
00766           input_error (_("Bad iterations number"));
00767           goto exit_on_error;
00768         }
00769
00770       // Obtaining best number
00771       input->nbest
00772         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00773                                              &error_code);
00774       if (error_code || !input->nbest)
00775         {
00776           input_error (_("Invalid best number"));
00777           goto exit_on_error;
00778         }
00779
00780       // Obtaining tolerance
00781       input->tolerance
00782         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00783                                               &error_code);
00784       if (error_code || input->tolerance < 0.)
00785         {
00786           input_error (_("Invalid tolerance"));
00787           goto exit_on_error;
00788         }
00789
00790       // Getting direction search method parameters
00791       if (json_object_get_member (object, LABEL_NSTEPS))
00792         {
00793           input->nsteps
00794             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00795           if (error_code)
00796             {
00797               input_error (_("Invalid steps number"));
00798               goto exit_on_error;
00799             }
00800           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00801           if (!strcmp (buffer, LABEL_COORDINATES))
00802             input->direction = DIRECTION_METHOD_COORDINATES;
00803           else if (!strcmp (buffer, LABEL_RANDOM))
00804             {
00805               input->direction = DIRECTION_METHOD_RANDOM;
00806               input->nestimates
00807                 =
00808                 json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00809               if (error_code || !input->nestimates)
00810                 {
00811                   input_error (_("Invalid estimates number"));
00812                   goto exit_on_error;
00813                 }
00814             }
00815           else
00816             {
00817               input_error
00818                 (_("Unknown method to estimate the direction search"));
00819               goto exit_on_error;
00820             }
00821           input->relaxation
00822             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00823                                                   DEFAULT_RELAXATION,
00824                                                   &error_code);
00825           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00826             {
00827               input_error (_("Invalid relaxation parameter"));
00828               goto exit_on_error;
00829             }
00830         }
00831       else
00832         input->nsteps = 0;
00833     }
00834   // Obtaining the threshold
00835   input->threshold
00836     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00837                                           &error_code);
00838   if (error_code)
00839     {
00840       input_error (_("Invalid threshold"));
00841       goto exit_on_error;
00842     }
```

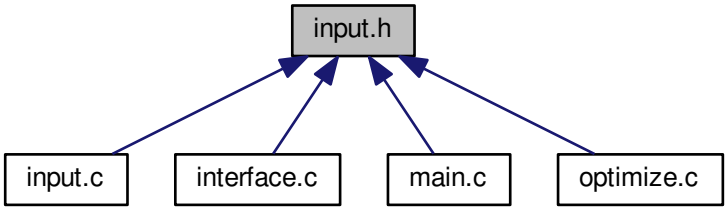
```
00843
00844   // Reading the experimental data
00845   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846   n = json_array_get_length (array);
00847   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00848   for (i = 0; i < n; ++i)
00849     {
00850 #if DEBUG_INPUT
00851       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00852                input->nexperiments);
00853 #endif
00854       child = json_array_get_element (array, i);
00855       if (!input->nexperiments)
00856         {
00857           if (!experiment_open_json (input->experiment, child, 0))
00858             goto exit_on_error;
00859         }
00860       else
00861         {
00862           if (!experiment_open_json (input->experiment + input->
      nexperiments,
00863                                      child, input->experiment->ninputs))
00864             goto exit_on_error;
00865         }
00866       ++input->nexperiments;
00867 #if DEBUG_INPUT
00868       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00869                input->nexperiments);
00870 #endif
00871     }
00872   if (!input->nexperiments)
00873     {
00874       input_error (_("No optimization experiments"));
00875       goto exit_on_error;
00876     }
00877
00878   // Reading the variables data
00879   array = json_object_get_array_member (object, LABEL_VARIABLES);
00880   n = json_array_get_length (array);
00881   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00882   for (i = 0; i < n; ++i)
00883     {
00884 #if DEBUG_INPUT
00885       fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00886 #endif
00887       child = json_array_get_element (array, i);
00888       if (!variable_open_json (input->variable + input->
      nvariables, child,
00889                                input->algorithm, input->nsteps))
00890         goto exit_on_error;
00891       ++input->nvariables;
00892     }
00893   if (!input->nvariables)
00894     {
00895       input_error (_("No optimization variables"));
00896       goto exit_on_error;
00897     }
00898
00899   // Obtaining the error norm
00900   if (json_object_get_member (object, LABEL_NORM))
00901     {
00902       buffer = json_object_get_string_member (object, LABEL_NORM);
00903       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904         input->norm = ERROR_NORM_EUCLIDIAN;
00905       else if (!strcmp (buffer, LABEL_MAXIMUM))
00906         input->norm = ERROR_NORM_MAXIMUM;
00907       else if (!strcmp (buffer, LABEL_P))
00908         {
00909           input->norm = ERROR_NORM_P;
00910           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00911           if (!error_code)
00912             {
00913               input_error (_("Bad P parameter"));
00914               goto exit_on_error;
00915             }
00916         }
00917       else if (!strcmp (buffer, LABEL_TAXICAB))
00918         input->norm = ERROR_NORM_TAXICAB;
00919       else
00920         {
00921           input_error (_("Unknown error norm"));
00922           goto exit_on_error;
00923         }
00924     }
00925   else
```

```
00926     input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928   // Closing the JSON document
00929   g_object_unref (parser);
00930
00931 #if DEBUG_INPUT
00932   fprintf (stderr, "input_open_json: end\n");
00933 #endif
00934   return 1;
00935
00936 exit_on_error:
00937   g_object_unref (parser);
00938 #if DEBUG_INPUT
00939   fprintf (stderr, "input_open_json: end\n");
00940 #endif
00941   return 0;
00942 }
00943
00951 int
00952 input_open (char *filename)
00953 {
00954   xmlDoc *doc;
00955   JsonParser *parser;
00956
00957 #if DEBUG_INPUT
00958   fprintf (stderr, "input_open: start\n");
00959 #endif
00960
00961   // Resetting input data
00962   input_new ();
00963
00964   // Opening input file
00965 #if DEBUG_INPUT
00966   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00967   fprintf (stderr, "input_open: trying XML format\n");
00968 #endif
00969   doc = xmlParseFile (filename);
00970   if (!doc)
00971     {
00972 #if DEBUG_INPUT
00973       fprintf (stderr, "input_open: trying JSON format\n");
00974 #endif
00975       parser = json_parser_new ();
00976       if (!json_parser_load_from_file (parser, filename, NULL))
00977         {
00978           input_error (_("Unable to parse the input file"));
00979           goto exit_on_error;
00980         }
00981       if (!input_open_json (parser))
00982         goto exit_on_error;
00983     }
00984   else if (!input_open_xml (doc))
00985     goto exit_on_error;
00986
00987   // Getting the working directory
00988   input->directory = g_path_get_dirname (filename);
00989   input->name = g_path_get_basename (filename);
00990
00991 #if DEBUG_INPUT
00992   fprintf (stderr, "input_open: end\n");
00993 #endif
00994   return 1;
00995
00996 exit_on_error:
00997   show_error (error_message);
00998   g_free (error_message);
00999   input_free ();
01000 #if DEBUG_INPUT
01001   fprintf (stderr, "input_open: end\n");
01002 #endif
01003   return 0;
01004 }
```

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 ()

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

  Function to free the memory of the input file data.
- void input_error (char *message)

  Function to print an error message opening an Input struct.
- int input_open_xml (xmlDoc *doc)

  Function to open the input file in XML format.
- int input_open_json (JsonParser *parser)

  Function to open the input file in JSON format.
- int input_open (char *filename)

  Function to open the input file.

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

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 124 of file input.c.

```
00125 {
00126   char buffer[64];
00127   snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128   error_message = g_strdup (buffer);
00129 }
```

4.9.3.2 input_open()

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

Function to open the input file.

Parameters

filename	Input data file name.

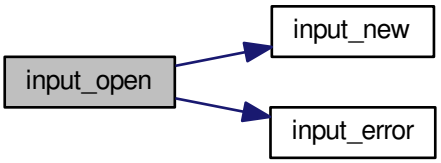
Returns

1_on_success, 0_on_error.

Definition at line 952 of file input.c.

```
00953 {
00954   xmlDoc *doc;
00955   JsonParser *parser;
00956
00957 #if DEBUG_INPUT
00958   fprintf (stderr, "input_open: start\n");
00959 #endif
00960
00961   // Resetting input data
00962   input_new ();
00963
00964   // Opening input file
00965 #if DEBUG_INPUT
00966   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00967   fprintf (stderr, "input_open: trying XML format\n");
00968 #endif
00969   doc = xmlParseFile (filename);
00970   if (!doc)
00971     {
00972 #if DEBUG_INPUT
00973       fprintf (stderr, "input_open: trying JSON format\n");
00974 #endif
00975       parser = json_parser_new ();
00976       if (!json_parser_load_from_file (parser, filename, NULL))
00977         {
00978           input_error (_("Unable to parse the input file"));
00979           goto exit_on_error;
00980         }
00981       if (!input_open_json (parser))
00982         goto exit_on_error;
00983     }
00984   else if (!input_open_xml (doc))
00985     goto exit_on_error;
00986
00987   // Getting the working directory
00988   input->directory = g_path_get_dirname (filename);
00989   input->name = g_path_get_basename (filename);
00990
00991 #if DEBUG_INPUT
00992   fprintf (stderr, "input_open: end\n");
00993 #endif
00994   return 1;
00995
00996 exit_on_error:
00997   show_error (error_message);
00998   g_free (error_message);
00999   input_free ();
01000 #if DEBUG_INPUT
01001   fprintf (stderr, "input_open: end\n");
01002 #endif
01003   return 0;
01004 }
```

Here is the call graph for this function:



4.9.3.3 input_open_json()

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

Function to open the input file in JSON format.

Parameters

parser	JsonParser struct.

Returns

1_on_success, 0_on_error.

Definition at line 562 of file input.c.

```
00563 {
00564   JsonNode *node, *child;
00565   JsonObject *object;
00566   JsonArray *array;
00567   const char *buffer;
00568   int error_code;
00569   unsigned int i, n;
00570
00571 #if DEBUG_INPUT
00572   fprintf (stderr, "input_open_json: start\n");
00573 #endif
00574
00575   // Resetting input data
00576   input->type = INPUT_TYPE_JSON;
00577
00578   // Getting the root node
00579 #if DEBUG_INPUT
00580   fprintf (stderr, "input_open_json: getting the root node\n");
00581 #endif
00582   node = json_parser_get_root (parser);
00583   object = json_node_get_object (node);
00584
00585   // Getting result and variables file names
00586   if (!input->result)
00587     {
00588       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589       if (!buffer)
00590         buffer = result_name;
00591       input->result = g_strdup (buffer);
00592     }
00593   else
00594     input->result = g_strdup (result_name);
00595   if (!input->variables)
00596     {
00597       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598       if (!buffer)
00599         buffer = variables_name;
00600       input->variables = g_strdup (buffer);
00601     }
00602   else
00603     input->variables = g_strdup (variables_name);
00604
00605   // Opening simulator program name
00606   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607   if (!buffer)
00608     {
00609       input_error (_("Bad simulator program"));
00610       goto exit_on_error;
00611     }
00612   input->simulator = g_strdup (buffer);
00613
00614   // Opening evaluator program name
00615   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616   if (buffer)
00617     input->evaluator = g_strdup (buffer);
00618
00619   // Obtaining pseudo-random numbers generator seed
00620   input->seed
00621     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00622                                          DEFAULT_RANDOM_SEED, &error_code);
00623   if (error_code)
00624     {
00625       input_error (_("Bad pseudo-random numbers generator seed"));
00626       goto exit_on_error;
00627     }
```

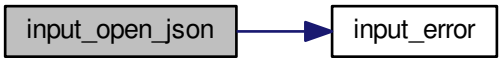
```
00628
00629   // Opening algorithm
00630   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633       input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635       // Obtaining simulations number
00636       input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00638       if (error_code)
00639         {
00640           input_error (_("Bad simulations number"));
00641           goto exit_on_error;
00642         }
00643     }
00644   else if (!strcmp (buffer, LABEL_SWEEP))
00645     input->algorithm = ALGORITHM_SWEEP;
00646   else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648       input->algorithm = ALGORITHM_GENETIC;
00649
00650       // Obtaining population
00651       if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653           input->nsimulations
00654             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00655           if (error_code || input->nsimulations < 3)
00656             {
00657               input_error (_("Invalid population number"));
00658               goto exit_on_error;
00659             }
00660         }
00661       else
00662         {
00663           input_error (_("No population number"));
00664           goto exit_on_error;
00665         }
00666
00667       // Obtaining generations
00668       if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670           input->niterations
00671             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00672           if (error_code || !input->niterations)
00673             {
00674               input_error (_("Invalid generations number"));
00675               goto exit_on_error;
00676             }
00677         }
00678       else
00679         {
00680           input_error (_("No generations number"));
00681           goto exit_on_error;
00682         }
00683
00684       // Obtaining mutation probability
00685       if (json_object_get_member (object, LABEL_MUTATION))
00686         {
00687           input->mutation_ratio
00688             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00689           if (error_code || input->mutation_ratio < 0.
00690               || input->mutation_ratio >= 1.)
00691             {
00692               input_error (_("Invalid mutation probability"));
00693               goto exit_on_error;
00694             }
00695         }
00696       else
00697         {
00698           input_error (_("No mutation probability"));
00699           goto exit_on_error;
00700         }
00701
00702       // Obtaining reproduction probability
00703       if (json_object_get_member (object, LABEL_REPRODUCTION))
00704         {
00705           input->reproduction_ratio
00706             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00707           if (error_code || input->reproduction_ratio < 0.
00708               || input->reproduction_ratio >= 1.0)
00709             {
```

```
00710               input_error (_("Invalid reproduction probability"));
00711               goto exit_on_error;
00712             }
00713         }
00714       else
00715         {
00716           input_error (_("No reproduction probability"));
00717           goto exit_on_error;
00718         }
00719
00720       // Obtaining adaptation probability
00721       if (json_object_get_member (object, LABEL_ADAPTATION))
00722         {
00723           input->adaptation_ratio
00724             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00725           if (error_code || input->adaptation_ratio < 0.
00726               || input->adaptation_ratio >= 1.)
00727             {
00728               input_error (_("Invalid adaptation probability"));
00729               goto exit_on_error;
00730             }
00731         }
00732       else
00733         {
00734           input_error (_("No adaptation probability"));
00735           goto exit_on_error;
00736         }
00737
00738       // Checking survivals
00739       i = input->mutation_ratio * input->nsimulations;
00740       i += input->reproduction_ratio * input->
      nsimulations;
00741       i += input->adaptation_ratio * input->
      nsimulations;
00742       if (i > input->nsimulations - 2)
00743         {
00744           input_error
00745             (_("No enough survival entities to reproduce the population"));
00746           goto exit_on_error;
00747         }
00748     }
00749   else
00750     {
00751       input_error (_("Unknown algorithm"));
00752       goto exit_on_error;
00753     }
00754
00755   if (input->algorithm == ALGORITHM_MONTE_CARLO
00756       || input->algorithm == ALGORITHM_SWEEP)
00757     {
00758
00759       // Obtaining iterations number
00760       input->niterations
00761         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00762       if (error_code == 1)
00763         input->niterations = 1;
00764       else if (error_code)
00765         {
00766           input_error (_("Bad iterations number"));
00767           goto exit_on_error;
00768         }
00769
00770       // Obtaining best number
00771       input->nbest
00772         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00773                                              &error_code);
00774       if (error_code || !input->nbest)
00775         {
00776           input_error (_("Invalid best number"));
00777           goto exit_on_error;
00778         }
00779
00780       // Obtaining tolerance
00781       input->tolerance
00782         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00783                                               &error_code);
00784       if (error_code || input->tolerance < 0.)
00785         {
00786           input_error (_("Invalid tolerance"));
00787           goto exit_on_error;
00788         }
00789
00790       // Getting direction search method parameters
```

```
00791       if (json_object_get_member (object, LABEL_NSTEPS))
00792         {
00793           input->nsteps
00794             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00795           if (error_code)
00796             {
00797               input_error (_("Invalid steps number"));
00798               goto exit_on_error;
00799             }
00800           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00801           if (!strcmp (buffer, LABEL_COORDINATES))
00802             input->direction = DIRECTION_METHOD_COORDINATES;
00803           else if (!strcmp (buffer, LABEL_RANDOM))
00804             {
00805               input->direction = DIRECTION_METHOD_RANDOM;
00806               input->nestimates
00807                 =
00808                   json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00809               if (error_code || !input->nestimates)
00810                 {
00811                   input_error (_("Invalid estimates number"));
00812                   goto exit_on_error;
00813                 }
00814             }
00815           else
00816             {
00817               input_error
00818                 (_("Unknown method to estimate the direction search"));
00819               goto exit_on_error;
00820             }
00821           input->relaxation
00822             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00823                                                   DEFAULT_RELAXATION,
00824                                                   &error_code);
00825           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00826             {
00827               input_error (_("Invalid relaxation parameter"));
00828               goto exit_on_error;
00829             }
00830         }
00831       else
00832         input->nsteps = 0;
00833     }
00834   // Obtaining the threshold
00835   input->threshold
00836     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00837                                           &error_code);
00838   if (error_code)
00839     {
00840       input_error (_("Invalid threshold"));
00841       goto exit_on_error;
00842     }
00843
00844   // Reading the experimental data
00845   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846   n = json_array_get_length (array);
00847   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00848   for (i = 0; i < n; ++i)
00849     {
00850 #if DEBUG_INPUT
00851       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00852                input->nexperiments);
00853 #endif
00854       child = json_array_get_element (array, i);
00855       if (!input->nexperiments)
00856         {
00857           if (!experiment_open_json (input->experiment, child, 0))
00858             goto exit_on_error;
00859         }
00860       else
00861         {
00862           if (!experiment_open_json (input->experiment +
      input->nexperiments,
00863                                      child, input->experiment->
      ninputs))
00864             goto exit_on_error;
00865         }
00866       ++input->nexperiments;
00867 #if DEBUG_INPUT
00868       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00869                input->nexperiments);
00870 #endif
```

```
00871     }
00872   if (!input->nexperiments)
00873     {
00874       input_error (_("No optimization experiments"));
00875       goto exit_on_error;
00876     }
00877
00878   // Reading the variables data
00879   array = json_object_get_array_member (object, LABEL_VARIABLES);
00880   n = json_array_get_length (array);
00881   input->variable = (Variable *) g_malloc (n * sizeof (
      Variable));
00882   for (i = 0; i < n; ++i)
00883     {
00884 #if DEBUG_INPUT
00885       fprintf (stderr, "input_open_json: nvariables=%u\n", input->
      nvariables);
00886 #endif
00887       child = json_array_get_element (array, i);
00888       if (!variable_open_json (input->variable +
      input->nvariables, child,
00889                                input->algorithm, input->
      nsteps))
00890         goto exit_on_error;
00891       ++input->nvariables;
00892     }
00893   if (!input->nvariables)
00894     {
00895       input_error (_("No optimization variables"));
00896       goto exit_on_error;
00897     }
00898
00899   // Obtaining the error norm
00900   if (json_object_get_member (object, LABEL_NORM))
00901     {
00902       buffer = json_object_get_string_member (object, LABEL_NORM);
00903       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904         input->norm = ERROR_NORM_EUCLIDIAN;
00905       else if (!strcmp (buffer, LABEL_MAXIMUM))
00906         input->norm = ERROR_NORM_MAXIMUM;
00907       else if (!strcmp (buffer, LABEL_P))
00908         {
00909           input->norm = ERROR_NORM_P;
00910           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00911           if (!error_code)
00912             {
00913               input_error (_("Bad P parameter"));
00914               goto exit_on_error;
00915             }
00916         }
00917       else if (!strcmp (buffer, LABEL_TAXICAB))
00918         input->norm = ERROR_NORM_TAXICAB;
00919       else
00920         {
00921           input_error (_("Unknown error norm"));
00922           goto exit_on_error;
00923         }
00924     }
00925   else
00926     input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928   // Closing the JSON document
00929   g_object_unref (parser);
00930
00931 #if DEBUG_INPUT
00932   fprintf (stderr, "input_open_json: end\n");
00933 #endif
00934   return 1;
00935
00936 exit_on_error:
00937   g_object_unref (parser);
00938 #if DEBUG_INPUT
00939   fprintf (stderr, "input_open_json: end\n");
00940 #endif
00941   return 0;
00942 }
```

Here is the call graph for this function:



4.9.3.4 input_open_xml()

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

Function to open the input file in XML format.

Parameters

doc	xmlDoc struct.

Returns

1_on_success, 0_on_error.

Definition at line 139 of file input.c.

```
00140 {
00141   char buffer2[64];
00142   xmlNode *node, *child;
00143   xmlChar *buffer;
00144   int error_code;
00145   unsigned int i;
00146
00147 #if DEBUG_INPUT
00148   fprintf (stderr, "input_open_xml: start\n");
00149 #endif
00150
00151   // Resetting input data
00152   buffer = NULL;
00153   input->type = INPUT_TYPE_XML;
00154
00155   // Getting the root node
00156 #if DEBUG_INPUT
00157   fprintf (stderr, "input_open_xml: getting the root node\n");
00158 #endif
00159   node = xmlDocGetRootElement (doc);
00160   if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162       input_error (_("Bad root XML node"));
00163       goto exit_on_error;
00164     }
00165
00166   // Getting result and variables file names
00167   if (!input->result)
00168     {
00169       input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171       if (!input->result)
```

```
00172         input->result = (char *) xmlStrdup ((const xmlChar *)
      result_name);
00173     }
00174   if (!input->variables)
00175     {
00176       input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178       if (!input->variables)
00179         input->variables =
00180           (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183   // Opening simulator program name
00184   input->simulator =
00185     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186   if (!input->simulator)
00187     {
00188       input_error (_("Bad simulator program"));
00189       goto exit_on_error;
00190     }
00191
00192   // Opening evaluator program name
00193   input->evaluator =
00194     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196   // Obtaining pseudo-random numbers generator seed
00197   input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_SEED,
00199                                       DEFAULT_RANDOM_SEED, &error_code);
00200   if (error_code)
00201     {
00202       input_error (_("Bad pseudo-random numbers generator seed"));
00203       goto exit_on_error;
00204     }
00205
00206   // Opening algorithm
00207   buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208   if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209     {
00210       input->algorithm = ALGORITHM_MONTE_CARLO;
00211
00212       // Obtaining simulations number
00213       input->nsimulations
00214         = xml_node_get_int (node, (const xmlChar *)
      LABEL_NSIMULATIONS,
00215                             &error_code);
00216       if (error_code)
00217         {
00218           input_error (_("Bad simulations number"));
00219           goto exit_on_error;
00220         }
00221     }
00222   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223     input->algorithm = ALGORITHM_SWEEP;
00224   else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226       input->algorithm = ALGORITHM_GENETIC;
00227
00228       // Obtaining population
00229       if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231           input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *) LABEL_NPOPULATION,
00233                                  &error_code);
00234           if (error_code || input->nsimulations < 3)
00235             {
00236               input_error (_("Invalid population number"));
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (_("No population number"));
00243           goto exit_on_error;
00244         }
00245
00246       // Obtaining generations
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248         {
00249           input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NGENERATIONS,
00251                                  &error_code);
00252           if (error_code || !input->niterations)
00253             {
00254               input_error (_("Invalid generations number"));
00255               goto exit_on_error;
```

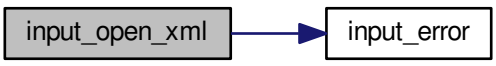
```
00256             }
00257         }
00258       else
00259         {
00260           input_error (_("No generations number"));
00261           goto exit_on_error;
00262         }
00263
00264       // Obtaining mutation probability
00265       if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266         {
00267           input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *) LABEL_MUTATION,
00269                                   &error_code);
00270           if (error_code || input->mutation_ratio < 0.
00271               || input->mutation_ratio >= 1.)
00272             {
00273               input_error (_("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
00277       else
00278         {
00279           input_error (_("No mutation probability"));
00280           goto exit_on_error;
00281         }
00282
00283       // Obtaining reproduction probability
00284       if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285         {
00286           input->reproduction_ratio
00287             = xml_node_get_float (node, (const xmlChar *) LABEL_REPRODUCTION,
00288                                   &error_code);
00289           if (error_code || input->reproduction_ratio < 0.
00290               || input->reproduction_ratio >= 1.0)
00291             {
00292               input_error (_("Invalid reproduction probability"));
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (_("No reproduction probability"));
00299           goto exit_on_error;
00300         }
00301
00302       // Obtaining adaptation probability
00303       if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304         {
00305           input->adaptation_ratio
00306             = xml_node_get_float (node, (const xmlChar *) LABEL_ADAPTATION,
00307                                   &error_code);
00308           if (error_code || input->adaptation_ratio < 0.
00309               || input->adaptation_ratio >= 1.)
00310             {
00311               input_error (_("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (_("No adaptation probability"));
00318           goto exit_on_error;
00319         }
00320
00321       // Checking survivals
00322       i = input->mutation_ratio * input->nsimulations;
00323       i += input->reproduction_ratio * input->
      nsimulations;
00324       i += input->adaptation_ratio * input->
      nsimulations;
00325       if (i > input->nsimulations - 2)
00326         {
00327           input_error
00328             (_("No enough survival entities to reproduce the population"));
00329           goto exit_on_error;
00330         }
00331     }
00332   else
00333     {
00334       input_error (_("Unknown algorithm"));
00335       goto exit_on_error;
00336     }
00337   xmlFree (buffer);
00338   buffer = NULL;
00339
00340   if (input->algorithm == ALGORITHM_MONTE_CARLO
```

```
00341       || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344       // Obtaining iterations number
00345       input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NITERATIONS,
00347                              &error_code);
00348       if (error_code == 1)
00349         input->niterations = 1;
00350       else if (error_code)
00351         {
00352           input_error (_("Bad iterations number"));
00353           goto exit_on_error;
00354         }
00355
00356       // Obtaining best number
00357       input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_NBEST,
00359                                           1, &error_code);
00360       if (error_code || !input->nbest)
00361         {
00362           input_error (_("Invalid best number"));
00363           goto exit_on_error;
00364         }
00365       if (input->nbest > input->nsimulations)
00366         {
00367           input_error (_("Best number higher than simulations number"));
00368           goto exit_on_error;
00369         }
00370
00371       // Obtaining tolerance
00372       input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                            (const xmlChar *) LABEL_TOLERANCE,
00375                                            0., &error_code);
00376       if (error_code || input->tolerance < 0.)
00377         {
00378           input_error (_("Invalid tolerance"));
00379           goto exit_on_error;
00380         }
00381
00382       // Getting direction search method parameters
00383       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384         {
00385           input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *) LABEL_NSTEPS,
00387                                &error_code);
00388           if (error_code)
00389             {
00390               input_error (_("Invalid steps number"));
00391               goto exit_on_error;
00392             }
00393           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->direction = DIRECTION_METHOD_COORDINATES;
00396           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397             {
00398               input->direction = DIRECTION_METHOD_RANDOM;
00399               input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NESTIMATES,
00401                                      &error_code);
00402               if (error_code || !input->nestimates)
00403                 {
00404                   input_error (_("Invalid estimates number"));
00405                   goto exit_on_error;
00406                 }
00407             }
00408           else
00409             {
00410               input_error
00411                 (_("Unknown method to estimate the direction search"));
00412               goto exit_on_error;
00413             }
00414           xmlFree (buffer);
00415           buffer = NULL;
00416           input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                                (const xmlChar *)
00419                                                LABEL_RELAXATION,
00420                                                DEFAULT_RELAXATION,
00421                                                &error_code);
00422           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00423             {
```

```
00424               input_error (_("Invalid relaxation parameter"));
00425               goto exit_on_error;
00426             }
00427         }
00428       else
00429         input->nsteps = 0;
00430     }
00431   // Obtaining the threshold
00432   input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00434                                      0., &error_code);
00435   if (error_code)
00436     {
00437       input_error (_("Invalid threshold"));
00438       goto exit_on_error;
00439     }
00440
00441   // Reading the experimental data
00442   for (child = node->children; child; child = child->next)
00443     {
00444       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448                input->nexperiments);
00449 #endif
00450       input->experiment = (Experiment *)
00451         g_realloc (input->experiment,
00452                    (1 + input->nexperiments) * sizeof (
      Experiment));
00453       if (!input->nexperiments)
00454         {
00455           if (!experiment_open_xml (input->experiment, child, 0))
00456             goto exit_on_error;
00457         }
00458       else
00459         {
00460           if (!experiment_open_xml (input->experiment +
      input->nexperiments,
00461                                     child, input->experiment->
      ninputs))
00462             goto exit_on_error;
00463         }
00464       ++input->nexperiments;
00465 #if DEBUG_INPUT
00466       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00467                input->nexperiments);
00468 #endif
00469     }
00470   if (!input->nexperiments)
00471     {
00472       input_error (_("No optimization experiments"));
00473       goto exit_on_error;
00474     }
00475   buffer = NULL;
00476
00477   // Reading the variables data
00478   for (; child; child = child->next)
00479     {
00480 #if DEBUG_INPUT
00481       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00482 #endif
00483       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00484         {
00485           snprintf (buffer2, 64, "%s %u: %s",
00486                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00487           input_error (buffer2);
00488           goto exit_on_error;
00489         }
00490       input->variable = (Variable *)
00491         g_realloc (input->variable,
00492                    (1 + input->nvariables) * sizeof (Variable));
00493       if (!variable_open_xml (input->variable +
      input->nvariables, child,
00494                               input->algorithm, input->nsteps))
00495         goto exit_on_error;
00496       ++input->nvariables;
00497     }
00498   if (!input->nvariables)
00499     {
00500       input_error (_("No optimization variables"));
00501       goto exit_on_error;
00502     }
00503   buffer = NULL;
00504
00505   // Obtaining the error norm
```

```
00506   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507     {
00508       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510         input->norm = ERROR_NORM_EUCLIDIAN;
00511       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512         input->norm = ERROR_NORM_MAXIMUM;
00513       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514         {
00515           input->norm = ERROR_NORM_P;
00516           input->p
00517             =
00518             xml_node_get_float (node, (const xmlChar *) LABEL_P, &error_code);
00519           if (!error_code)
00520             {
00521               input_error (_("Bad P parameter"));
00522               goto exit_on_error;
00523             }
00524         }
00525       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526         input->norm = ERROR_NORM_TAXICAB;
00527       else
00528         {
00529           input_error (_("Unknown error norm"));
00530           goto exit_on_error;
00531         }
00532       xmlFree (buffer);
00533     }
00534   else
00535     input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537   // Closing the XML document
00538   xmlFreeDoc (doc);
00539
00540 #if DEBUG_INPUT
00541   fprintf (stderr, "input_open_xml: end\n");
00542 #endif
00543   return 1;
00544
00545 exit_on_error:
00546   xmlFree (buffer);
00547   xmlFreeDoc (doc);
00548 #if DEBUG_INPUT
00549   fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551   return 0;
00552 }
```

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 2012-2016, 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
```

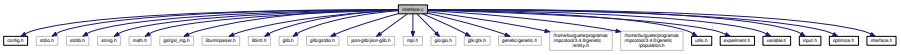
```
00013   1. Redistributions of source code must retain the above copyright notice,
00014      this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017      this list of conditions and the following disclaimer 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)

  Function to save the direction search method data in a XML node.
- void input_save_direction_json (JsonNode *node)

  Function to save the direction search method data in a JSON node.
- void input_save_xml (xmlDoc *doc)

  Function to save the input file in XML format.
- void input_save_json (JsonGenerator *generator)

  Function to save the input file in JSON format.
- void input_save (char *filename)

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

  Function to open the options dialog.

- void running_new ()

  Function to open the running dialog.
- unsigned int window_get_algorithm ()

  Function to get the stochastic algorithm number.
- unsigned int window_get_direction ()

  Function to get the direction search method number.
- unsigned int window_get_norm ()

  Function to get the norm method number.
- void window_save_direction ()

  Function to save the direction search method data in the input file.
- int window_save ()

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

  Function to run a optimization.
- void window_help ()

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

  Function to show an about dialog.
- void window_update_direction ()

  Function to update direction search method widgets view in the main window.
- void window_update ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Function to update the variable rangemax in the main window.

- void window_rangeminabs_variable ()

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

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

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

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

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

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

  Function to open the main window.

Variables

- const char * logo [ ]

  Logo pixmap.
- Options options [1]

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

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

  Window struct to define the main interface window.

4.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file interface.c.

4.11.2 Function Documentation

4.11.2.1 input_save()

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

Function to save the input file.

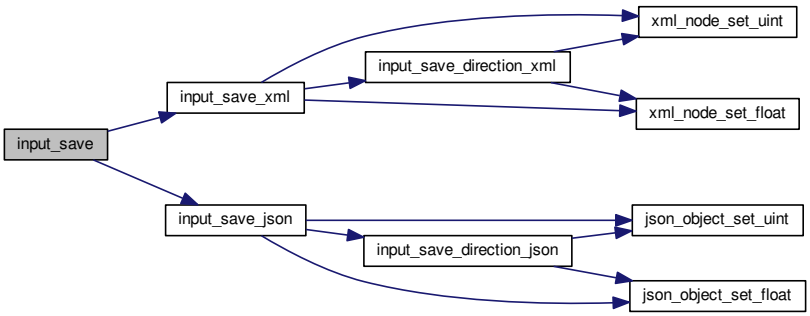
Parameters

filename	Input file name.

Definition at line 579 of file interface.c.

```
00580 {
00581   xmlDoc *doc;
00582   JsonGenerator *generator;
00583
00584 #if DEBUG_INTERFACE
00585   fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588   // Getting the input file directory
00589   input->name = g_path_get_basename (filename);
00590   input->directory = g_path_get_dirname (filename);
00591
00592   if (input->type == INPUT_TYPE_XML)
00593     {
00594       // Opening the input file
00595       doc = xmlNewDoc ((const xmlChar *) "1.0");
00596       input_save_xml (doc);
00597
00598       // Saving the XML file
00599       xmlSaveFormatFile (filename, doc, 1);
00600
00601       // Freeing memory
00602       xmlFreeDoc (doc);
00603     }
00604   else
00605     {
00606       // Opening the input file
00607       generator = json_generator_new ();
00608       json_generator_set_pretty (generator, TRUE);
00609       input_save_json (generator);
00610
00611       // Saving the JSON file
00612       json_generator_to_file (generator, filename, NULL);
00613
00614       // Freeing memory
00615       g_object_unref (generator);
00616     }
00617
00618 #if DEBUG_INTERFACE
00619   fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }
```

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.

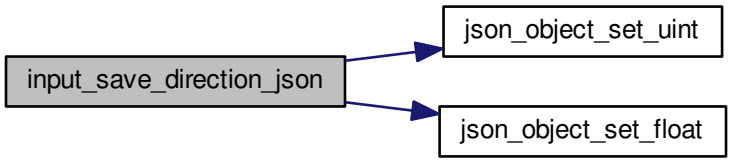
Parameters

node	JSON node.

Definition at line 207 of file interface.c.

```
00208 {
00209   JsonObject *object;
00210 #if DEBUG_INTERFACE
00211   fprintf (stderr, "input_save_direction_json: start\n");
00212 #endif
00213   object = json_node_get_object (node);
00214   if (input->nsteps)
00215     {
00216       json_object_set_uint (object, LABEL_NSTEPS,
      input->nsteps);
00217       if (input->relaxation != DEFAULT_RELAXATION)
00218         json_object_set_float (object, LABEL_RELAXATION,
      input->relaxation);
00219       switch (input->direction)
00220         {
00221         case DIRECTION_METHOD_COORDINATES:
00222           json_object_set_string_member (object, LABEL_DIRECTION,
00223                                          LABEL_COORDINATES);
00224           break;
00225         default:
00226           json_object_set_string_member (object, LABEL_DIRECTION,
00227                                          LABEL_RANDOM);
00228           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00229         }
00230     }
00231 #if DEBUG_INTERFACE
00232   fprintf (stderr, "input_save_direction_json: end\n");
00233 #endif
00234 }
```

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.

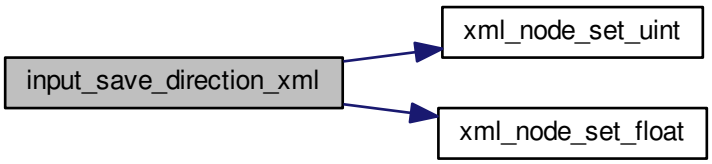
Parameters

node	XML node.

Definition at line 171 of file interface.c.

```
00172 {
00173 #if DEBUG_INTERFACE
00174   fprintf (stderr, "input_save_direction_xml: start\n");
00175 #endif
00176   if (input->nsteps)
00177     {
00178       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00179       if (input->relaxation != DEFAULT_RELAXATION)
00180         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00181                             input->relaxation);
00182       switch (input->direction)
00183         {
00184         case DIRECTION_METHOD_COORDINATES:
00185           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00186                       (const xmlChar *) LABEL_COORDINATES);
00187           break;
00188         default:
00189           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00190                       (const xmlChar *) LABEL_RANDOM);
00191           xml_node_set_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00192                              input->nestimates);
00193         }
00194     }
00195 #if DEBUG_INTERFACE
00196   fprintf (stderr, "input_save_direction_xml: end\n");
00197 #endif
00198 }
```

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.

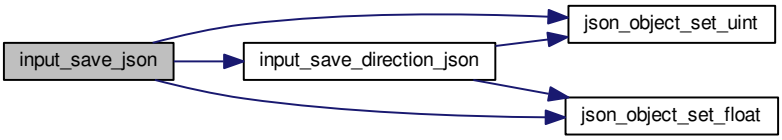
Definition at line 416 of file interface.c.

```
00417 {
00418   unsigned int i, j;
00419   char *buffer;
00420   JsonNode *node, *child;
00421   JsonObject *object, *object2;
00422   JsonArray *array;
00423   GFile *file, *file2;
00424
00425 #if DEBUG_INTERFACE
00426   fprintf (stderr, "input_save_json: start\n");
00427 #endif
00428
00429   // Setting root JSON node
00430   node = json_node_new (JSON_NODE_OBJECT);
00431   object = json_node_get_object (node);
00432   json_generator_set_root (generator, node);
00433
00434   // Adding properties to the root JSON node
00435   if (strcmp (input->result, result_name))
00436     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00437   if (strcmp (input->variables, variables_name))
00438     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00439                                    input->variables);
00440   file = g_file_new_for_path (input->directory);
00441   file2 = g_file_new_for_path (input->simulator);
00442   buffer = g_file_get_relative_path (file, file2);
00443   g_object_unref (file2);
00444   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00445   g_free (buffer);
00446   if (input->evaluator)
00447     {
00448       file2 = g_file_new_for_path (input->evaluator);
00449       buffer = g_file_get_relative_path (file, file2);
00450       g_object_unref (file2);
00451       if (strlen (buffer))
00452         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00453       g_free (buffer);
00454     }
00455   if (input->seed != DEFAULT_RANDOM_SEED)
00456     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00457
00458   // Setting the algorithm
00459   buffer = (char *) g_slice_alloc (64);
00460   switch (input->algorithm)
00461     {
00462     case ALGORITHM_MONTE_CARLO:
00463       json_object_set_string_member (object, LABEL_ALGORITHM,
00464                                      LABEL_MONTE_CARLO);
00465       snprintf (buffer, 64, "%u", input->nsimulations);
00466       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00467       snprintf (buffer, 64, "%u", input->niterations);
00468       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00469       snprintf (buffer, 64, "%.3lg", input->tolerance);
00470       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00471       snprintf (buffer, 64, "%u", input->nbest);
00472       json_object_set_string_member (object, LABEL_NBEST, buffer);
00473       input_save_direction_json (node);
00474       break;
00475     case ALGORITHM_SWEEP:
00476       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00477       snprintf (buffer, 64, "%u", input->niterations);
00478       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00479       snprintf (buffer, 64, "%.3lg", input->tolerance);
00480       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00481       snprintf (buffer, 64, "%u", input->nbest);
00482       json_object_set_string_member (object, LABEL_NBEST, buffer);
00483       input_save_direction_json (node);
00484       break;
00485     default:
00486       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_GENETIC);
00487       snprintf (buffer, 64, "%u", input->nsimulations);
00488       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
```

```
00489       snprintf (buffer, 64, "%u", input->niterations);
00490       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00491       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00492       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00493       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00494       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00495       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00496       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00497       break;
00498     }
00499   g_slice_free1 (64, buffer);
00500   if (input->threshold != 0.)
00501     json_object_set_float (object, LABEL_THRESHOLD,
      input->threshold);
00502
00503   // Setting the experimental data
00504   array = json_array_new ();
00505   for (i = 0; i < input->nexperiments; ++i)
00506     {
00507       child = json_node_new (JSON_NODE_OBJECT);
00508       object = json_node_get_object (child);
00509       json_object_set_string_member (object2, LABEL_NAME,
00510                                      input->experiment[i].name);
00511       if (input->experiment[i].weight != 1.)
00512         json_object_set_float (object2, LABEL_WEIGHT,
00513                                input->experiment[i].weight);
00514       for (j = 0; j < input->experiment->ninputs; ++j)
00515         json_object_set_string_member (object2, template[j],
00516                                        input->experiment[i].
      template[j]);
00517       json_array_add_element (array, child);
00518     }
00519   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00520
00521   // Setting the variables data
00522   array = json_array_new ();
00523   for (i = 0; i < input->nvariables; ++i)
00524     {
00525       child = json_node_new (JSON_NODE_OBJECT);
00526       object = json_node_get_object (child);
00527       json_object_set_string_member (object2, LABEL_NAME,
00528                                      input->variable[i].name);
00529       json_object_set_float (object2, LABEL_MINIMUM,
00530                              input->variable[i].rangemin);
00531       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532         json_object_set_float (object2,
      LABEL_ABSOLUTE_MINIMUM,
00533                                input->variable[i].rangeminabs);
00534       json_object_set_float (object2, LABEL_MAXIMUM,
00535                              input->variable[i].rangemax);
00536       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
      LABEL_ABSOLUTE_MAXIMUM,
00538                                input->variable[i].rangemaxabs);
00539       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00540         json_object_set_uint (object2, LABEL_PRECISION,
00541                               input->variable[i].precision);
00542       if (input->algorithm == ALGORITHM_SWEEP)
00543         json_object_set_uint (object2, LABEL_NSWEEPS,
00544                               input->variable[i].nsweeps);
00545       else if (input->algorithm == ALGORITHM_GENETIC)
00546         json_object_set_uint (object2, LABEL_NBITS,
      input->variable[i].nbits);
00547       if (input->nsteps)
00548         json_object_set_float (object, LABEL_STEP,
      input->variable[i].step);
00549       json_array_add_element (array, child);
00550     }
00551   json_object_set_array_member (object, LABEL_VARIABLES, array);
00552
00553   // Saving the error norm
00554   switch (input->norm)
00555     {
00556     case ERROR_NORM_MAXIMUM:
00557       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00558       break;
00559     case ERROR_NORM_P:
00560       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00561       json_object_set_float (object, LABEL_P, input->
      p);
00562       break;
00563     case ERROR_NORM_TAXICAB:
00564       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00565     }
00566
00567 #if DEBUG_INTERFACE
```

```
00568   fprintf (stderr, "input_save_json: end\n");
00569 #endif
00570 }
```

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.

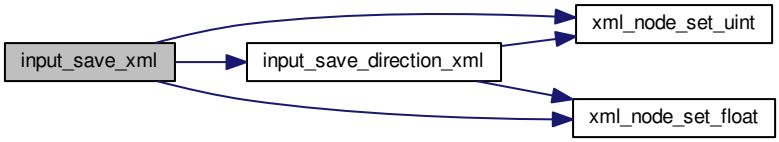
Definition at line 243 of file interface.c.

```
00244 {
00245   unsigned int i, j;
00246   char *buffer;
00247   xmlNode *node, *child;
00248   GFile *file, *file2;
00249
00250 #if DEBUG_INTERFACE
00251   fprintf (stderr, "input_save_xml: start\n");
00252 #endif
00253
00254   // Setting root XML node
00255   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00256   xmlDocSetRootElement (doc, node);
00257
00258   // Adding properties to the root XML node
00259   if (xmlStrcmp
00260       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00261     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00262                 (xmlChar *) input->result);
00263   if (xmlStrcmp
00264       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00265     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00266                 (xmlChar *) input->variables);
00267   file = g_file_new_for_path (input->directory);
00268   file2 = g_file_new_for_path (input->simulator);
00269   buffer = g_file_get_relative_path (file, file2);
00270   g_object_unref (file2);
00271   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00272   g_free (buffer);
00273   if (input->evaluator)
00274     {
```

```
00275       file2 = g_file_new_for_path (input->evaluator);
00276       buffer = g_file_get_relative_path (file, file2);
00277       g_object_unref (file2);
00278       if (xmlStrlen ((xmlChar *) buffer))
00279         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00280                     (xmlChar *) buffer);
00281       g_free (buffer);
00282     }
00283   if (input->seed != DEFAULT_RANDOM_SEED)
00284     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00285
00286   // Setting the algorithm
00287   buffer = (char *) g_slice_alloc (64);
00288   switch (input->algorithm)
00289     {
00290     case ALGORITHM_MONTE_CARLO:
00291       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00292                   (const xmlChar *) LABEL_MONTE_CARLO);
00293       snprintf (buffer, 64, "%u", input->nsimulations);
00294       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00295                   (xmlChar *) buffer);
00296       snprintf (buffer, 64, "%u", input->niterations);
00297       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00298                   (xmlChar *) buffer);
00299       snprintf (buffer, 64, "%.3lg", input->tolerance);
00300       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00301                   (xmlChar *) buffer);
00302       snprintf (buffer, 64, "%u", input->nbest);
00303       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304       input_save_direction_xml (node);
00305       break;
00306     case ALGORITHM_SWEEP:
00307       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                   (const xmlChar *) LABEL_SWEEP);
00309       snprintf (buffer, 64, "%u", input->niterations);
00310       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                   (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%.3lg", input->tolerance);
00313       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00314                   (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,
00335                   (xmlChar *) buffer);
00336       break;
00337     }
00338   g_slice_free1 (64, buffer);
00339   if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00341                         input->threshold);
00342
00343   // Setting the experimental data
00344   for (i = 0; i < input->nexperiments; ++i)
00345     {
00346       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00347       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00348                   (xmlChar *) input->experiment[i].name);
00349       if (input->experiment[i].weight != 1.)
00350         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00351                             input->experiment[i].weight);
00352       for (j = 0; j < input->experiment->ninputs; ++j)
00353         xmlSetProp (child, (const xmlChar *) template[j],
00354                     (xmlChar *) input->experiment[i].template[j]);
00355     }
00356
00357   // Setting the variables data
00358   for (i = 0; i < input->nvariables; ++i)
```

```
00359     {
00360       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00361       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00362                   (xmlChar *) input->variable[i].name);
00363       xmlNodeSetFloat (child, (const xmlChar *)
    LABEL_MINIMUM,
00364                        input->variable[i].rangemin);
00365       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00366         xmlNodeSetFloat (child, (const xmlChar *)
    LABEL_ABSOLUTE_MINIMUM,
00367                          input->variable[i].rangeminabs);
00368       xmlNodeSetFloat (child, (const xmlChar *)
    LABEL_MAXIMUM,
00369                        input->variable[i].rangemax);
00370       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00371         xmlNodeSetFloat (child, (const xmlChar *)
    LABEL_ABSOLUTE_MAXIMUM,
00372                          input->variable[i].rangemaxabs);
00373       if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00374         xmlNodeSetUint (child, (const xmlChar *)
    LABEL_PRECISION,
00375                         input->variable[i].precision);
00376       if (input->algorithm == ALGORITHM_SWEEP)
00377         xmlNodeSetUint (child, (const xmlChar *)
    LABEL_NSWEEPS,
00378                         input->variable[i].nsweeps);
00379       else if (input->algorithm == ALGORITHM_GENETIC)
00380         xmlNodeSetUint (child, (const xmlChar *) LABEL_NBITS,
00381                         input->variable[i].nbits);
00382       if (input->nsteps)
00383         xmlNodeSetFloat (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       xmlNodeSetFloat (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 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:

4.11.2.7 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:

4.11.2.8 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

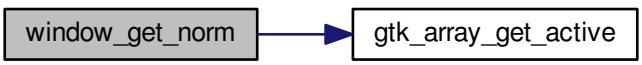
Returns

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:



4.11.2.9 window_new()

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

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2108 of file interface.c.

```
02109 {
02110   unsigned int i;
02111   char *buffer, *buffer2, buffer3[64];
02112   char *label_algorithm[NALGORITHMS] = {
02113     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114   };
02115   char *tip_algorithm[NALGORITHMS] = {
02116     _("Monte-Carlo brute force algorithm"),
02117     _("Sweep brute force algorithm"),
02118     _("Genetic algorithm")
02119   };
02120   char *label_direction[NDIRECTIONS] = {
02121     _("_Coordinates descent"), _("_Random")
02122   };
02123   char *tip_direction[NDIRECTIONS] = {
02124     _("Coordinates direction estimate method"),
02125     _("Random direction estimate method")
02126   };
02127   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128   char *tip_norm[NNORMS] = {
02129     _("Euclidean error norm (L2)"),
02130     _("Maximum error norm (L)"),
02131     _("P error norm (Lp)"),
02132     _("Taxicab error norm (L1)")
02133   };
02134
02135 #if DEBUG_INTERFACE
02136   fprintf (stderr, "window_new: start\n");
02137 #endif
02138
02139   // Creating the window
02140   window->window = main_window
02141     = (GtkWindow *) gtk_application_window_new (application);
02142
02143   // Finish when closing the window
02144   g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148   // Setting the window title
02149   gtk_window_set_title (window->window, "MPCOTool");
02150
02151   // Creating the open button
02152   window->button_open = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("document-open",
02154                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157   // Creating the save button
02158   window->button_save = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("document-save",
02160                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02162                     NULL);
02163
02164   // Creating the run button
02165   window->button_run = (GtkToolButton *) gtk_tool_button_new
02166     (gtk_image_new_from_icon_name ("system-run",
02167                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02168   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02169
02170   // Creating the options button
02171   window->button_options = (GtkToolButton *) gtk_tool_button_new
```

```
02172     (gtk_image_new_from_icon_name ("preferences-system",
02173                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02174      _("Options"));
02175   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02176
02177   // Creating the help button
02178   window->button_help = (GtkToolButton *) gtk_tool_button_new
02179     (gtk_image_new_from_icon_name ("help-browser",
02180                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02181   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02182
02183   // Creating the about button
02184   window->button_about = (GtkToolButton *) gtk_tool_button_new
02185     (gtk_image_new_from_icon_name ("help-about",
02186                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02187   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02188
02189   // Creating the exit button
02190   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02191     (gtk_image_new_from_icon_name ("application-exit",
02192                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02193   g_signal_connect_swapped (window->button_exit, "clicked",
02194                             G_CALLBACK (g_application_quit),
02195                             G_APPLICATION (application));
02196
02197   // Creating the buttons bar
02198   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02199   gtk_toolbar_insert
02200     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02201   gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02203   gtk_toolbar_insert
02204     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
02205   gtk_toolbar_insert
02206     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02207   gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02209   gtk_toolbar_insert
02210     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02211   gtk_toolbar_insert
02212     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02213   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02214
02215   // Creating the simulator program label and entry
02216   window->label_simulator
02217     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02218   window->button_simulator = (GtkFileChooserButton *)
02219     gtk_file_chooser_button_new (_("Simulator program"),
02220                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02221   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02222                                _("Simulator program executable file"));
02223   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02224
02225   // Creating the evaluator program label and entry
02226   window->check_evaluator = (GtkCheckButton *)
02227     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02228   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02229   window->button_evaluator = (GtkFileChooserButton *)
02230     gtk_file_chooser_button_new (_("Evaluator program"),
02231                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02232   gtk_widget_set_tooltip_text
02233     (GTK_WIDGET (window->button_evaluator),
02234      _("Optional evaluator program executable file"));
02235
02236   // Creating the results files labels and entries
02237   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02238   window->entry_result = (GtkEntry *) gtk_entry_new ();
02239   gtk_widget_set_tooltip_text
02240     (GTK_WIDGET (window->entry_result), _("Best results file"));
02241   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02242   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02245
02246   // Creating the files grid and attaching widgets
02247   window->grid_files = (GtkGrid *) gtk_grid_new ();
02248   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
```

```
      label_simulator),
02249                     0, 0, 1, 1);
02250   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02251                     1, 0, 1, 1);
02252   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02253                     0, 1, 1, 1);
02254   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02255                     1, 1, 1, 1);
02256   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02257                     0, 2, 1, 1);
02258   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02259                     1, 2, 1, 1);
02260   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02261                     0, 3, 1, 1);
02262   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02263                     1, 3, 1, 1);
02264
02265   // Creating the algorithm properties
02266   window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268   window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270   gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272      _("Number of simulations to perform for each iteration"));
02273   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274   window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276   window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278   gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280   g_signal_connect
02281     (window->spin_iterations, "value-changed",
      window_update, NULL);
02282   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284   window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286   gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288      _("Tolerance to set the variable interval on the next iteration"));
02289   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290   window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294      _("Number of best simulations used to set the variable interval "
02295        "on the next iteration"));
02296   window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298   window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300   gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302      _("Number of population for the genetic algorithm"));
02303   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304   window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306   window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308   gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310      _("Number of generations for the genetic algorithm"));
02311   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312   window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314   gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317   window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319   window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321   gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324   window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326   window->spin_adaptation
```

```
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328   gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332   window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335   gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338   window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                      GTK_WIDGET (window->spin_threshold));
02342 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                         GTK_ALIGN_FILL);
02345
02346   // Creating the direction search method properties
02347   window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02349   g_signal_connect (window->check_direction, "clicked",
    window_update, NULL);
02350   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02351   window->button_direction[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02355   g_signal_connect (window->button_direction[0], "clicked",
    window_update,
02356                     NULL);
02357   for (i = 0; ++i < NDIRECTIONS;)
02358     {
02359       window->button_direction[i] = (GtkRadioButton *)
02360         gtk_radio_button_new_with_mnemonic
02361         (gtk_radio_button_get_group (window->button_direction[0]),
02362          label_direction[i]);
02363       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02364                                    tip_direction[i]);
02365       gtk_grid_attach (window->grid_direction,
02366                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02367       g_signal_connect (window->button_direction[i], "clicked",
02368                         window_update, NULL);
02369     }
02370   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02371   window->spin_steps = (GtkSpinButton *)
02372     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02373   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02374   window->label_estimates
02375     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02376   window->spin_estimates = (GtkSpinButton *)
02377     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02378   window->label_relaxation
02379     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02380   window->spin_relaxation = (GtkSpinButton *)
02381     gtk_spin_button_new_with_range (0., 2., 0.001);
02382   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->label_steps),
02383                    0, NDIRECTIONS, 1, 1);
02384   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->spin_steps),
02385                    1, NDIRECTIONS, 1, 1);
02386   gtk_grid_attach (window->grid_direction,
02387                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02388                    1, 1);
02389   gtk_grid_attach (window->grid_direction,
02390                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02391                    1);
02392   gtk_grid_attach (window->grid_direction,
02393                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394                    1, 1);
02395   gtk_grid_attach (window->grid_direction,
02396                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397                    1, 1);
02398
02399   // Creating the array of algorithms
02400   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401   window->button_algorithm[0] = (GtkRadioButton *)
02402     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404                                tip_algorithm[0]);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407   g_signal_connect (window->button_algorithm[0], "clicked",
02408                     window_set_algorithm, NULL);
02409   for (i = 0; ++i < NALGORITHMS;)
```

```
02410     {
02411       window->button_algorithm[i] = (GtkRadioButton *)
02412         gtk_radio_button_new_with_mnemonic
02413         (gtk_radio_button_get_group (window->button_algorithm[0]),
02414          label_algorithm[i]);
02415       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416                                    tip_algorithm[i]);
02417       gtk_grid_attach (window->grid_algorithm,
02418                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419       g_signal_connect (window->button_algorithm[i], "clicked",
02420                         window_set_algorithm, NULL);
02421     }
02422   gtk_grid_attach (window->grid_algorithm,
02423                    GTK_WIDGET (window->label_simulations), 0,
02424                    NALGORITHMS, 1, 1);
02425   gtk_grid_attach (window->grid_algorithm,
02426                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                    1);
02428   gtk_grid_attach (window->grid_algorithm,
02429                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                    1, 1);
02431   gtk_grid_attach (window->grid_algorithm,
02432                    GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                    1, 1);
02434   gtk_grid_attach (window->grid_algorithm,
02435                    GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                    1, 1);
02437   gtk_grid_attach (window->grid_algorithm,
02438                    GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                    1);
02440   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests),
02441                    0, NALGORITHMS + 3, 1, 1);
02442   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02443                    NALGORITHMS + 3, 1, 1);
02444   gtk_grid_attach (window->grid_algorithm,
02445                    GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02446                    1, 1);
02447   gtk_grid_attach (window->grid_algorithm,
02448                    GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02449                    1, 1);
02450   gtk_grid_attach (window->grid_algorithm,
02451                    GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02452                    1, 1);
02453   gtk_grid_attach (window->grid_algorithm,
02454                    GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02455                    1, 1);
02456   gtk_grid_attach (window->grid_algorithm,
02457                    GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02458                    1);
02459   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02460                    1, NALGORITHMS + 6, 1, 1);
02461   gtk_grid_attach (window->grid_algorithm,
02462                    GTK_WIDGET (window->label_reproduction), 0,
02463                    NALGORITHMS + 7, 1, 1);
02464   gtk_grid_attach (window->grid_algorithm,
02465                    GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02466                    1, 1);
02467   gtk_grid_attach (window->grid_algorithm,
02468                    GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02469                    1, 1);
02470   gtk_grid_attach (window->grid_algorithm,
02471                    GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02472                    1, 1);
02473   gtk_grid_attach (window->grid_algorithm,
02474                    GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475                    2, 1);
02476   gtk_grid_attach (window->grid_algorithm,
02477                    GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478                    2, 1);
02479   gtk_grid_attach (window->grid_algorithm,
02480                    GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481                    1, 1);
02482   gtk_grid_attach (window->grid_algorithm,
02483                    GTK_WIDGET (window->scrolled_threshold), 1,
02484                    NALGORITHMS + 11, 1, 1);
02485   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487                      GTK_WIDGET (window->grid_algorithm));
02488
02489   // Creating the variable widgets
02490   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491   gtk_widget_set_tooltip_text
02492     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493   window->id_variable = g_signal_connect
```

```
02494     (window->combo_variable, "changed", window_set_variable, NULL);
02495   window->button_add_variable
02496     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497                                                    GTK_ICON_SIZE_BUTTON);
02498   g_signal_connect
02499     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02500   gtk_widget_set_tooltip_text
02501     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02502   window->button_remove_variable
02503     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02504                                                    GTK_ICON_SIZE_BUTTON);
02505   g_signal_connect
02506     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02507   gtk_widget_set_tooltip_text
02508     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02509   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02510   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02511   gtk_widget_set_tooltip_text
02512     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02513   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02514   window->id_variable_label = g_signal_connect
02515     (window->entry_variable, "changed",
      window_label_variable, NULL);
02516   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02517   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02518     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_min),
02521      _("Minimum initial value of the variable"));
02522   window->scrolled_min
02523     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02524   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02525                      GTK_WIDGET (window->spin_min));
02526   g_signal_connect (window->spin_min, "value-changed",
02527                     window_rangemin_variable, NULL);
02528   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02529   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02530     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02531   gtk_widget_set_tooltip_text
02532     (GTK_WIDGET (window->spin_max),
02533      _("Maximum initial value of the variable"));
02534   window->scrolled_max
02535     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02537                      GTK_WIDGET (window->spin_max));
02538   g_signal_connect (window->spin_max, "value-changed",
02539                     window_rangemax_variable, NULL);
02540   window->check_minabs = (GtkCheckButton *)
02541     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02542   g_signal_connect (window->check_minabs, "toggled",
      window_update, NULL);
02543   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02544     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02545   gtk_widget_set_tooltip_text
02546     (GTK_WIDGET (window->spin_minabs),
02547      _("Minimum allowed value of the variable"));
02548   window->scrolled_minabs
02549     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02550   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02551                      GTK_WIDGET (window->spin_minabs));
02552   g_signal_connect (window->spin_minabs, "value-changed",
02553                     window_rangeminabs_variable, NULL);
02554   window->check_maxabs = (GtkCheckButton *)
02555     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02556   g_signal_connect (window->check_maxabs, "toggled",
      window_update, NULL);
02557   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02558     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02559   gtk_widget_set_tooltip_text
02560     (GTK_WIDGET (window->spin_maxabs),
02561      _("Maximum allowed value of the variable"));
02562   window->scrolled_maxabs
02563     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565                      GTK_WIDGET (window->spin_maxabs));
02566   g_signal_connect (window->spin_maxabs, "value-changed",
02567                     window_rangemaxabs_variable, NULL);
02568   window->label_precision
02569     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570   window->spin_precision = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572   gtk_widget_set_tooltip_text
02573     (GTK_WIDGET (window->spin_precision),
02574      _("Number of precision floating point digits\n"
02575        "0 is for integer numbers"));
```

```
02576   g_signal_connect (window->spin_precision, "value-changed",
02577                     window_precision_variable, NULL);
02578   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579   window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582                                _("Number of steps sweeping the variable"));
02583   g_signal_connect (window->spin_sweeps, "value-changed",
02584                     window_update_variable, NULL);
02585   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586   window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588   gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591   g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
    ;
02593   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596   gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598      _("Initial step size for the direction search method"));
02599   window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602                      GTK_WIDGET (window->spin_step));
02603   g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608   gtk_grid_attach (window->grid_variable,
02609                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610   gtk_grid_attach (window->grid_variable,
02611                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612   gtk_grid_attach (window->grid_variable,
02613                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614   gtk_grid_attach (window->grid_variable,
02615                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616   gtk_grid_attach (window->grid_variable,
02617                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618   gtk_grid_attach (window->grid_variable,
02619                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620   gtk_grid_attach (window->grid_variable,
02621                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622   gtk_grid_attach (window->grid_variable,
02623                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624   gtk_grid_attach (window->grid_variable,
02625                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626   gtk_grid_attach (window->grid_variable,
02627                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628   gtk_grid_attach (window->grid_variable,
02629                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630   gtk_grid_attach (window->grid_variable,
02631                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632   gtk_grid_attach (window->grid_variable,
02633                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634   gtk_grid_attach (window->grid_variable,
02635                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636   gtk_grid_attach (window->grid_variable,
02637                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638   gtk_grid_attach (window->grid_variable,
02639                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640   gtk_grid_attach (window->grid_variable,
02641                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642   gtk_grid_attach (window->grid_variable,
02643                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644   gtk_grid_attach (window->grid_variable,
02645                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646   gtk_grid_attach (window->grid_variable,
02647                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                      GTK_WIDGET (window->grid_variable));
02651
02652   // Creating the experiment widgets
02653   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                                _("Experiment selector"));
02656   window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed",
    window_set_experiment, NULL);
02658   window->button_add_experiment
02659     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660                                                    GTK_ICON_SIZE_BUTTON);
```

```
02661   g_signal_connect
02662     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02663   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664                                _("Add experiment"));
02665   window->button_remove_experiment
02666     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667                                                    GTK_ICON_SIZE_BUTTON);
02668   g_signal_connect (window->button_remove_experiment, "clicked",
02669                     window_remove_experiment, NULL);
02670   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02671                                _("Remove experiment"));
02672   window->label_experiment
02673     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674   window->button_experiment = (GtkFileChooserButton *)
02675     gtk_file_chooser_button_new (_("Experimental data file"),
02676                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02677   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678                                _("Experimental data file"));
02679   window->id_experiment_name
02680     = g_signal_connect (window->button_experiment, "selection-changed",
02681                         window_name_experiment, NULL);
02682   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684   window->spin_weight
02685     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686   gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->spin_weight),
02688      _("Weight factor to build the objective function"));
02689   g_signal_connect
02690     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02691   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692   gtk_grid_attach (window->grid_experiment,
02693                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694   gtk_grid_attach (window->grid_experiment,
02695                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696   gtk_grid_attach (window->grid_experiment,
02697                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02698   gtk_grid_attach (window->grid_experiment,
02699                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700   gtk_grid_attach (window->grid_experiment,
02701                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702   gtk_grid_attach (window->grid_experiment,
02703                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704   gtk_grid_attach (window->grid_experiment,
02705                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706   for (i = 0; i < MAX_NINPUTS; ++i)
02707     {
02708       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709       window->check_template[i] = (GtkCheckButton *)
02710         gtk_check_button_new_with_label (buffer3);
02711       window->id_template[i]
02712         = g_signal_connect (window->check_template[i], "toggled",
02713                             window_inputs_experiment, NULL);
02714       gtk_grid_attach (window->grid_experiment,
02715                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716                        1);
02717       window->button_template[i] =
02718         (GtkFileChooserButton *)
02719         gtk_file_chooser_button_new (_("Input template"),
02720                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02721       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722                                    _("Experimental input template file"));
02723       window->id_input[i] =
02724         g_signal_connect_swapped (window->button_template[i],
02725                                   "selection-changed",
02726                                   (void (*)) window_template_experiment,
02727                                   (void *) (size_t) i);
02728       gtk_grid_attach (window->grid_experiment,
02729                        GTK_WIDGET (window->button_template[i]),
02730                        1, 3 + i, 3, 1);
02731     }
02732   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734                      GTK_WIDGET (window->grid_experiment));
02735
02736   // Creating the error norm widgets
02737   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740                      GTK_WIDGET (window->grid_norm));
02741   window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
```

```
02744                               tip_norm[0]);
02745   gtk_grid_attach (window->grid_norm,
02746                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02748   for (i = 0; ++i < NNORMS;)
02749     {
02750       window->button_norm[i] = (GtkRadioButton *)
02751         gtk_radio_button_new_with_mnemonic
02752         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02753       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02754                                    tip_norm[i]);
02755       gtk_grid_attach (window->grid_norm,
02756                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02757       g_signal_connect (window->button_norm[i], "clicked",
      window_update,
02758                         NULL);
02759     }
02760   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02761   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1,
02762                    1);
02763   window->spin_p =
02764     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02765                                                       G_MAXDOUBLE, 0.01);
02766   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02767                                _("P parameter for the P error norm"));
02768   window->scrolled_p =
02769     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02770   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02771                      GTK_WIDGET (window->spin_p));
02772   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02773   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02774   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      scrolled_p),
02775                    1, 2, 1, 2);
02776
02777   // Creating the grid and attaching the widgets to the grid
02778   window->grid = (GtkGrid *) gtk_grid_new ();
02779   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02780                    1);
02781   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02782   gtk_grid_attach (window->grid,
02783                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02784   gtk_grid_attach (window->grid,
02785                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02786   gtk_grid_attach (window->grid,
02787                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02788   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02789   gtk_container_add (GTK_CONTAINER (window->window),
02790                      GTK_WIDGET (window->grid));
02791
02792   // Setting the window logo
02793   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02794   gtk_window_set_icon (window->window, window->logo);
02795
02796   // Showing the window
02797   gtk_widget_show_all (GTK_WIDGET (window->window));
02798
02799   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02800 #if GTK_MINOR_VERSION >= 16
02801   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02802   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02803   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02804   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02805   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808                                40);
02809 #endif
02810
02811   // Reading initial example
02812   input_new ();
02813   buffer2 = g_get_current_dir ();
02814   buffer =
02815     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02816   g_free (buffer2);
02817   window_read (buffer);
02818   g_free (buffer);
02819
02820 #if DEBUG_INTERFACE
02821   fprintf (stderr, "window_new: start\n");
02822 #endif
02823 }
```

4.11.2.10 window_read()

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

Function to read the input data of a file.

Parameters

filename	File name.

Returns

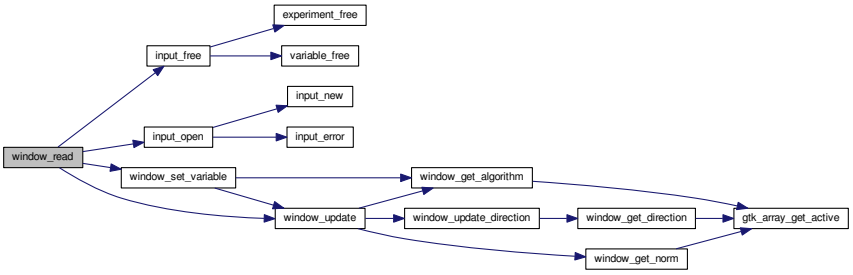
1 on succes, 0 on error.

Definition at line 1904 of file interface.c.

```
01905 {
01906   unsigned int i;
01907   char *buffer;
01908 #if DEBUG_INTERFACE
01909   fprintf (stderr, "window_read: start\n");
01910 #endif
01911
01912   // Reading new input file
01913   input_free ();
01914   if (!input_open (filename))
01915     {
01916 #if DEBUG_INTERFACE
01917       fprintf (stderr, "window_read: end\n");
01918 #endif
01919       return 0;
01920     }
01921
01922   // Setting GTK+ widgets data
01923   gtk_entry_set_text (window->entry_result, input->result);
01924   gtk_entry_set_text (window->entry_variables, input->
      variables);
01925   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01926   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01927                                  (window->button_simulator), buffer);
01928   g_free (buffer);
01929   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01930                                 (size_t) input->evaluator);
01931   if (input->evaluator)
01932     {
01933       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01934       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01935                                      (window->button_evaluator), buffer);
01936       g_free (buffer);
01937     }
01938   gtk_toggle_button_set_active
01939     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01940   switch (input->algorithm)
01941     {
01942     case ALGORITHM_MONTE_CARLO:
01943       gtk_spin_button_set_value (window->spin_simulations,
01944                                  (gdouble) input->nsimulations);
01945     case ALGORITHM_SWEEP:
01946       gtk_spin_button_set_value (window->spin_iterations,
01947                                  (gdouble) input->niterations);
01948       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01949       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01950       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01951                                     (window->check_direction),
      input->nsteps);
01952       if (input->nsteps)
01953         {
01954           gtk_toggle_button_set_active
```

```
01955             (GTK_TOGGLE_BUTTON (window->button_direction
01956                                 [input->direction]), TRUE);
01957           gtk_spin_button_set_value (window->spin_steps,
01958                                      (gdouble) input->nsteps);
01959           gtk_spin_button_set_value (window->spin_relaxation,
01960                                      (gdouble) input->relaxation);
01961           switch (input->direction)
01962             {
01963             case DIRECTION_METHOD_RANDOM:
01964               gtk_spin_button_set_value (window->spin_estimates,
01965                                          (gdouble) input->nestimates);
01966             }
01967         }
01968       break;
01969     default:
01970       gtk_spin_button_set_value (window->spin_population,
01971                                  (gdouble) input->nsimulations);
01972       gtk_spin_button_set_value (window->spin_generations,
01973                                  (gdouble) input->niterations);
01974       gtk_spin_button_set_value (window->spin_mutation,
01975                                  input->mutation_ratio);
01976       gtk_spin_button_set_value (window->spin_reproduction,
01977                                  input->reproduction_ratio);
01978       gtk_spin_button_set_value (window->spin_adaptation,
01979                                  input->adaptation_ratio);
01980     }
01981   gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983   gtk_spin_button_set_value (window->spin_p, input->p);
01984   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01985   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01986   g_signal_handler_block (window->button_experiment,
01987                           window->id_experiment_name);
01988   gtk_combo_box_text_remove_all (window->combo_experiment);
01989   for (i = 0; i < input->nexperiments; ++i)
01990     gtk_combo_box_text_append_text (window->combo_experiment,
01991                                     input->experiment[i].name);
01992   g_signal_handler_unblock
01993     (window->button_experiment, window->
      id_experiment_name);
01994   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01995   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01996   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01997   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01998   gtk_combo_box_text_remove_all (window->combo_variable);
01999   for (i = 0; i < input->nvariables; ++i)
02000     gtk_combo_box_text_append_text (window->combo_variable,
02001                                     input->variable[i].name);
02002   g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02005   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006   window_set_variable ();
02007   window_update ();
02008
02009 #if DEBUG_INTERFACE
02010   fprintf (stderr, "window_read: end\n");
02011 #endif
02012   return 1;
02013 }
```

Here is the call graph for this function:



4.11.2.11 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file interface.c.

```
00826 {
00827   GtkFileChooserDialog *dlg;
00828   GtkFileFilter *filter1, *filter2;
00829   char *buffer;
00830
00831 #if DEBUG_INTERFACE
00832   fprintf (stderr, "window_save: start\n");
00833 #endif
00834
00835   // Opening the saving dialog
00836   dlg = (GtkFileChooserDialog *)
00837     gtk_file_chooser_dialog_new (_("Save file"),
00838                                  window->window,
00839                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                  _("_Cancel"),
00841                                  GTK_RESPONSE_CANCEL,
00842                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00843   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                   TRUE);
00845   buffer = g_build_filename (input->directory, input->name, NULL);
00846   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847   g_free (buffer);
00848
00849   // Adding XML filter
00850   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851   gtk_file_filter_set_name (filter1, "XML");
00852   gtk_file_filter_add_pattern (filter1, "*.xml");
00853   gtk_file_filter_add_pattern (filter1, "*.XML");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856   // Adding JSON filter
00857   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858   gtk_file_filter_set_name (filter2, "JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.json");
00860   gtk_file_filter_add_pattern (filter2, "*.JSON");
00861   gtk_file_filter_add_pattern (filter2, "*.js");
00862   gtk_file_filter_add_pattern (filter2, "*.JS");
00863   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
```

```
00864
00865   if (input->type == INPUT_TYPE_XML)
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867   else
00868     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870   // If OK response then saving
00871   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873       // Setting input file type
00874       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875       buffer = (char *) gtk_file_filter_get_name (filter1);
00876       if (!strcmp (buffer, "XML"))
00877         input->type = INPUT_TYPE_XML;
00878       else
00879         input->type = INPUT_TYPE_JSON;
00880
00881       // Adding properties to the root XML node
00882       input->simulator = gtk_file_chooser_get_filename
00883         (GTK_FILE_CHOOSER (window->button_simulator));
00884       if (gtk_toggle_button_get_active
00885           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886         input->evaluator = gtk_file_chooser_get_filename
00887           (GTK_FILE_CHOOSER (window->button_evaluator));
00888       else
00889         input->evaluator = NULL;
00890       if (input->type == INPUT_TYPE_XML)
00891         {
00892           input->result
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->entry_result));
00895           input->variables
00896             = (char *) xmlStrdup ((const xmlChar *)
00897                                   gtk_entry_get_text
00898                                   (window->entry_variables));
00899         }
00900       else
00901         {
00902           input->result =
00903             g_strdup (gtk_entry_get_text (window->entry_result));
00904           input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907
00908       // Setting the algorithm
00909       switch (window_get_algorithm ())
00910         {
00911         case ALGORITHM_MONTE_CARLO:
00912           input->algorithm = ALGORITHM_MONTE_CARLO;
00913           input->nsimulations
00914             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915           input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917           input->tolerance =
00918             gtk_spin_button_get_value (window->spin_tolerance);
00919           input->nbest =
00920             gtk_spin_button_get_value_as_int (window->spin_bests);
00921           window_save_direction ();
00922           break;
00923         case ALGORITHM_SWEEP:
00924           input->algorithm = ALGORITHM_SWEEP;
00925           input->niterations
00926             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927           input->tolerance =
00928             gtk_spin_button_get_value (window->spin_tolerance);
00929           input->nbest =
00930             gtk_spin_button_get_value_as_int (window->spin_bests);
00931           window_save_direction ();
00932           break;
00933         default:
00934           input->algorithm = ALGORITHM_GENETIC;
00935           input->nsimulations
00936             = gtk_spin_button_get_value_as_int (window->spin_population);
00937           input->niterations
00938             = gtk_spin_button_get_value_as_int (window->spin_generations);
00939           input->mutation_ratio
00940             = gtk_spin_button_get_value (window->spin_mutation);
00941           input->reproduction_ratio
00942             = gtk_spin_button_get_value (window->spin_reproduction);
00943           input->adaptation_ratio
00944             = gtk_spin_button_get_value (window->spin_adaptation);
00945           break;
00946         }
00947       input->norm = window_get_norm ();
00948       input->p = gtk_spin_button_get_value (window->spin_p);
00949       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
```

```
00950
00951       // Saving the XML file
00952       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953       input_save (buffer);
00954
00955       // Closing and freeing memory
00956       g_free (buffer);
00957       gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959       fprintf (stderr, "window_save: end\n");
00960 #endif
00961       return 1;
00962     }
00963
00964   // Closing and freeing memory
00965   gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967   fprintf (stderr, "window_save: end\n");
00968 #endif
00969   return 0;
00970 }
```

4.11.2.12 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 1542 of file interface.c.

```
01543 {
01544   unsigned int i, j;
01545   char *buffer;
01546   GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548   fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550   i = (size_t) data;
01551   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552   file1
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555   file2 = g_file_new_for_path (input->directory);
01556   buffer = g_file_get_relative_path (file2, file1);
01557   if (input->type == INPUT_TYPE_XML)
01558     input->experiment[j].template[i] =
01559       (char *) xmlStrdup ((xmlChar *) buffer);
01560   else
01561     input->experiment[j].template[i] = g_strdup (buffer);
01562   g_free (buffer);
01563   g_object_unref (file2);
01564   g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566   fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }
```

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 2012-2016, 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
00170 void
00171 input_save_direction_xml (xmlNode * node)
00172 {
00173 #if DEBUG_INTERFACE
00174   fprintf (stderr, "input_save_direction_xml: start\n");
00175 #endif
00176   if (input->nsteps)
00177     {
00178       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00179       if (input->relaxation != DEFAULT_RELAXATION)
00180         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00181                             input->relaxation);
00182       switch (input->direction)
00183         {
00184         case DIRECTION_METHOD_COORDINATES:
00185           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00186                       (const xmlChar *) LABEL_COORDINATES);
00187           break;
00188         default:
00189           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00190                       (const xmlChar *) LABEL_RANDOM);
00191           xml_node_set_uint (node, (const xmlChar *)
```

```
      LABEL_NESTIMATES,
00192                              input->nestimates);
00193         }
00194     }
00195 #if DEBUG_INTERFACE
00196   fprintf (stderr, "input_save_direction_xml: end\n");
00197 #endif
00198 }
00199
00206 void
00207 input_save_direction_json (JsonNode * node)
00208 {
00209   JsonObject *object;
00210 #if DEBUG_INTERFACE
00211   fprintf (stderr, "input_save_direction_json: start\n");
00212 #endif
00213   object = json_node_get_object (node);
00214   if (input->nsteps)
00215     {
00216       json_object_set_uint (object, LABEL_NSTEPS,
      input->nsteps);
00217       if (input->relaxation != DEFAULT_RELAXATION)
00218         json_object_set_float (object, LABEL_RELAXATION,
      input->relaxation);
00219       switch (input->direction)
00220         {
00221         case DIRECTION_METHOD_COORDINATES:
00222           json_object_set_string_member (object, LABEL_DIRECTION,
00223                                          LABEL_COORDINATES);
00224           break;
00225         default:
00226           json_object_set_string_member (object, LABEL_DIRECTION,
00227                                          LABEL_RANDOM);
00228           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00229         }
00230     }
00231 #if DEBUG_INTERFACE
00232   fprintf (stderr, "input_save_direction_json: end\n");
00233 #endif
00234 }
00235
00242 void
00243 input_save_xml (xmlDoc * doc)
00244 {
00245   unsigned int i, j;
00246   char *buffer;
00247   xmlNode *node, *child;
00248   GFile *file, *file2;
00249
00250 #if DEBUG_INTERFACE
00251   fprintf (stderr, "input_save_xml: start\n");
00252 #endif
00253
00254   // Setting root XML node
00255   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00256   xmlDocSetRootElement (doc, node);
00257
00258   // Adding properties to the root XML node
00259   if (xmlStrcmp
00260       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00261     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00262                 (xmlChar *) input->result);
00263   if (xmlStrcmp
00264       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00265     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00266                 (xmlChar *) input->variables);
00267   file = g_file_new_for_path (input->directory);
00268   file2 = g_file_new_for_path (input->simulator);
00269   buffer = g_file_get_relative_path (file, file2);
00270   g_object_unref (file2);
00271   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00272   g_free (buffer);
00273   if (input->evaluator)
00274     {
00275       file2 = g_file_new_for_path (input->evaluator);
00276       buffer = g_file_get_relative_path (file, file2);
00277       g_object_unref (file2);
00278       if (xmlStrlen ((xmlChar *) buffer))
00279         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00280                     (xmlChar *) buffer);
00281       g_free (buffer);
00282     }
00283   if (input->seed != DEFAULT_RANDOM_SEED)
00284     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
```

```
00285
00286   // Setting the algorithm
00287   buffer = (char *) g_slice_alloc (64);
00288   switch (input->algorithm)
00289     {
00290     case ALGORITHM_MONTE_CARLO:
00291       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00292                   (const xmlChar *) LABEL_MONTE_CARLO);
00293       snprintf (buffer, 64, "%u", input->nsimulations);
00294       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00295                   (xmlChar *) buffer);
00296       snprintf (buffer, 64, "%u", input->niterations);
00297       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00298                   (xmlChar *) buffer);
00299       snprintf (buffer, 64, "%.3lg", input->tolerance);
00300       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00301                   (xmlChar *) buffer);
00302       snprintf (buffer, 64, "%u", input->nbest);
00303       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304       input_save_direction_xml (node);
00305       break;
00306     case ALGORITHM_SWEEP:
00307       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                   (const xmlChar *) LABEL_SWEEP);
00309       snprintf (buffer, 64, "%u", input->niterations);
00310       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                   (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%.3lg", input->tolerance);
00313       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00314                   (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,
00335                   (xmlChar *) buffer);
00336       break;
00337     }
00338   g_slice_free1 (64, buffer);
00339   if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
    LABEL_THRESHOLD,
00341                         input->threshold);
00342
00343   // Setting the experimental data
00344   for (i = 0; i < input->nexperiments; ++i)
00345     {
00346       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00347       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00348                   (xmlChar *) input->experiment[i].name);
00349       if (input->experiment[i].weight != 1.)
00350         xml_node_set_float (child, (const xmlChar *)
    LABEL_WEIGHT,
00351                             input->experiment[i].weight);
00352       for (j = 0; j < input->experiment->ninputs; ++j)
00353         xmlSetProp (child, (const xmlChar *) template[j],
00354                     (xmlChar *) input->experiment[i].template[j]);
00355     }
00356
00357   // Setting the variables data
00358   for (i = 0; i < input->nvariables; ++i)
00359     {
00360       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00361       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00362                   (xmlChar *) input->variable[i].name);
00363       xml_node_set_float (child, (const xmlChar *)
    LABEL_MINIMUM,
00364                           input->variable[i].rangemin);
00365       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00366         xml_node_set_float (child, (const xmlChar *)
    LABEL_ABSOLUTE_MINIMUM,
00367                             input->variable[i].rangeminabs);
```

```
00368       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00369                           input->variable[i].rangemax);
00370       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00371         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00374         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00375                            input->variable[i].precision);
00376       if (input->algorithm == ALGORITHM_SWEEP)
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
00415 void
00416 input_save_json (JsonGenerator * generator)
00417 {
00418   unsigned int i, j;
00419   char *buffer;
00420   JsonNode *node, *child;
00421   JsonObject *object, *object2;
00422   JsonArray *array;
00423   GFile *file, *file2;
00424
00425 #if DEBUG_INTERFACE
00426   fprintf (stderr, "input_save_json: start\n");
00427 #endif
00428
00429   // Setting root JSON node
00430   node = json_node_new (JSON_NODE_OBJECT);
00431   object = json_node_get_object (node);
00432   json_generator_set_root (generator, node);
00433
00434   // Adding properties to the root JSON node
00435   if (strcmp (input->result, result_name))
00436     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00437   if (strcmp (input->variables, variables_name))
00438     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00439                                    input->variables);
00440   file = g_file_new_for_path (input->directory);
00441   file2 = g_file_new_for_path (input->simulator);
00442   buffer = g_file_get_relative_path (file, file2);
00443   g_object_unref (file2);
00444   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00445   g_free (buffer);
00446   if (input->evaluator)
00447     {
00448       file2 = g_file_new_for_path (input->evaluator);
00449       buffer = g_file_get_relative_path (file, file2);
00450       g_object_unref (file2);
00451       if (strlen (buffer))
00452         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
```

```
00453       g_free (buffer);
00454     }
00455   if (input->seed != DEFAULT_RANDOM_SEED)
00456     json_object_set_uint (object, LABEL_SEED,
    input->seed);
00457
00458   // Setting the algorithm
00459   buffer = (char *) g_slice_alloc (64);
00460   switch (input->algorithm)
00461     {
00462     case ALGORITHM_MONTE_CARLO:
00463       json_object_set_string_member (object, LABEL_ALGORITHM,
00464                                      LABEL_MONTE_CARLO);
00465       snprintf (buffer, 64, "%u", input->nsimulations);
00466       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00467       snprintf (buffer, 64, "%u", input->niterations);
00468       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00469       snprintf (buffer, 64, "%.3lg", input->tolerance);
00470       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00471       snprintf (buffer, 64, "%u", input->nbest);
00472       json_object_set_string_member (object, LABEL_NBEST, buffer);
00473       input_save_direction_json (node);
00474       break;
00475     case ALGORITHM_SWEEP:
00476       json_object_set_string_member (object, LABEL_ALGORITHM,
    LABEL_SWEEP);
00477       snprintf (buffer, 64, "%u", input->niterations);
00478       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00479       snprintf (buffer, 64, "%.3lg", input->tolerance);
00480       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00481       snprintf (buffer, 64, "%u", input->nbest);
00482       json_object_set_string_member (object, LABEL_NBEST, buffer);
00483       input_save_direction_json (node);
00484       break;
00485     default:
00486       json_object_set_string_member (object, LABEL_ALGORITHM,
    LABEL_GENETIC);
00487       snprintf (buffer, 64, "%u", input->nsimulations);
00488       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00489       snprintf (buffer, 64, "%u", input->niterations);
00490       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00491       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00492       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00493       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00494       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00495       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00496       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00497       break;
00498     }
00499   g_slice_free1 (64, buffer);
00500   if (input->threshold != 0.)
00501     json_object_set_float (object, LABEL_THRESHOLD,
    input->threshold);
00502
00503   // Setting the experimental data
00504   array = json_array_new ();
00505   for (i = 0; i < input->nexperiments; ++i)
00506     {
00507       child = json_node_new (JSON_NODE_OBJECT);
00508       object = json_node_get_object (child);
00509       json_object_set_string_member (object2, LABEL_NAME,
00510                                      input->experiment[i].name);
00511       if (input->experiment[i].weight != 1.)
00512         json_object_set_float (object2, LABEL_WEIGHT,
00513                                input->experiment[i].weight);
00514       for (j = 0; j < input->experiment->ninputs; ++j)
00515         json_object_set_string_member (object2, template[j],
00516                                        input->experiment[i].
    template[j]);
00517       json_array_add_element (array, child);
00518     }
00519   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00520
00521   // Setting the variables data
00522   array = json_array_new ();
00523   for (i = 0; i < input->nvariables; ++i)
00524     {
00525       child = json_node_new (JSON_NODE_OBJECT);
00526       object = json_node_get_object (child);
00527       json_object_set_string_member (object2, LABEL_NAME,
00528                                      input->variable[i].name);
00529       json_object_set_float (object2, LABEL_MINIMUM,
00530                              input->variable[i].rangemin);
00531       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532         json_object_set_float (object2,
    LABEL_ABSOLUTE_MINIMUM,
00533                                input->variable[i].rangeminabs);
```

```
00534       json_object_set_float (object2, LABEL_MAXIMUM,
00535                              input->variable[i].rangemax);
00536       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
      LABEL_ABSOLUTE_MAXIMUM,
00538                                input->variable[i].rangemaxabs);
00539       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00540         json_object_set_uint (object2, LABEL_PRECISION,
00541                               input->variable[i].precision);
00542       if (input->algorithm == ALGORITHM_SWEEP)
00543         json_object_set_uint (object2, LABEL_NSWEEPS,
00544                               input->variable[i].nsweeps);
00545       else if (input->algorithm == ALGORITHM_GENETIC)
00546         json_object_set_uint (object2, LABEL_NBITS,
      input->variable[i].nbits);
00547       if (input->nsteps)
00548         json_object_set_float (object, LABEL_STEP,
      input->variable[i].step);
00549       json_array_add_element (array, child);
00550     }
00551   json_object_set_array_member (object, LABEL_VARIABLES, array);
00552
00553   // Saving the error norm
00554   switch (input->norm)
00555     {
00556     case ERROR_NORM_MAXIMUM:
00557       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00558       break;
00559     case ERROR_NORM_P:
00560       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00561       json_object_set_float (object, LABEL_P, input->
      p);
00562       break;
00563     case ERROR_NORM_TAXICAB:
00564       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00565     }
00566
00567 #if DEBUG_INTERFACE
00568   fprintf (stderr, "input_save_json: end\n");
00569 #endif
00570 }
00571
00578 void
00579 input_save (char *filename)
00580 {
00581   xmlDoc *doc;
00582   JsonGenerator *generator;
00583
00584 #if DEBUG_INTERFACE
00585   fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588   // Getting the input file directory
00589   input->name = g_path_get_basename (filename);
00590   input->directory = g_path_get_dirname (filename);
00591
00592   if (input->type == INPUT_TYPE_XML)
00593     {
00594       // Opening the input file
00595       doc = xmlNewDoc ((const xmlChar *) "1.0");
00596       input_save_xml (doc);
00597
00598       // Saving the XML file
00599       xmlSaveFormatFile (filename, doc, 1);
00600
00601       // Freeing memory
00602       xmlFreeDoc (doc);
00603     }
00604   else
00605     {
00606       // Opening the input file
00607       generator = json_generator_new ();
00608       json_generator_set_pretty (generator, TRUE);
00609       input_save_json (generator);
00610
00611       // Saving the JSON file
00612       json_generator_to_file (generator, filename, NULL);
00613
00614       // Freeing memory
00615       g_object_unref (generator);
00616     }
00617
00618 #if DEBUG_INTERFACE
00619   fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }
```

```
00622
00627 void
00628 options_new ()
00629 {
00630 #if DEBUG_INTERFACE
00631   fprintf (stderr, "options_new: start\n");
00632 #endif
00633   options->label_seed = (GtkLabel *)
00634     gtk_label_new (_("Pseudo-random numbers generator seed"));
00635   options->spin_seed = (GtkSpinButton *)
00636     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00637   gtk_widget_set_tooltip_text
00638     (GTK_WIDGET (options->spin_seed),
00639      _("Seed to init the pseudo-random numbers generator"));
00640   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00641   options->label_threads = (GtkLabel *)
00642     gtk_label_new (_("Threads number for the stochastic algorithm"));
00643   options->spin_threads
00644     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00645   gtk_widget_set_tooltip_text
00646     (GTK_WIDGET (options->spin_threads),
00647      _("Number of threads to perform the calibration/optimization for "
00648        "the stochastic algorithm"));
00649   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00650   options->label_direction = (GtkLabel *)
00651     gtk_label_new (_("Threads number for the direction search method"));
00652   options->spin_direction =
00653     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00654   gtk_widget_set_tooltip_text (GTK_WIDGET (options->spin_direction),
00655                                _
00656                                ("Number of threads to perform the calibration/optimization for "
00657                                 "the direction search method"));
00658   gtk_spin_button_set_value (options->spin_direction,
00659                              (gdouble) nthreads_direction);
00660   options->grid = (GtkGrid *) gtk_grid_new ();
00661   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1,
00662                    1);
00663   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1,
00664                    1);
00665   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00666                    1, 1);
00667   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00668                    1);
00669   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00670                    1, 1);
00671   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00672                    1, 1);
00673   gtk_widget_show_all (GTK_WIDGET (options->grid));
00674   options->dialog = (GtkDialog *)
00675     gtk_dialog_new_with_buttons (_("Options"),
00676                                  window->window,
00677                                  GTK_DIALOG_MODAL,
00678                                  _("_OK"), GTK_RESPONSE_OK,
00679                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00680   gtk_container_add
00681     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00682      GTK_WIDGET (options->grid));
00683   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00684     {
00685       input->seed
00686         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00687       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00688       nthreads_direction
00689         = gtk_spin_button_get_value_as_int (options->spin_direction);
00690     }
00691   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00692 #if DEBUG_INTERFACE
00693   fprintf (stderr, "options_new: end\n");
00694 #endif
00695 }
00696
00701 void
00702 running_new ()
00703 {
00704 #if DEBUG_INTERFACE
00705   fprintf (stderr, "running_new: start\n");
00706 #endif
00707   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00708   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00709   running->grid = (GtkGrid *) gtk_grid_new ();
00710   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00711   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00712   running->dialog = (GtkDialog *)
00713     gtk_dialog_new_with_buttons (_("Calculating"),
00714                                  window->window, GTK_DIALOG_MODAL, NULL,
```

```
00715                                   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
00790 void
00791 window_save_direction ()
00792 {
00793 #if DEBUG_INTERFACE
00794   fprintf (stderr, "window_save_direction: start\n");
00795 #endif
00796   if (gtk_toggle_button_get_active
00797       (GTK_TOGGLE_BUTTON (window->check_direction)))
00798     {
00799       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00800       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00801       switch (window_get_direction ())
00802         {
00803         case DIRECTION_METHOD_COORDINATES:
00804           input->direction = DIRECTION_METHOD_COORDINATES;
00805           break;
00806         default:
00807           input->direction = DIRECTION_METHOD_RANDOM;
00808           input->nestimates
00809             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00810         }
00811     }
00812   else
00813     input->nsteps = 0;
00814 #if DEBUG_INTERFACE
00815   fprintf (stderr, "window_save_direction: end\n");
```

```
00816 #endif
00817 }
00818
00824 int
00825 window_save ()
00826 {
00827   GtkFileChooserDialog *dlg;
00828   GtkFileFilter *filter1, *filter2;
00829   char *buffer;
00830
00831 #if DEBUG_INTERFACE
00832   fprintf (stderr, "window_save: start\n");
00833 #endif
00834
00835   // Opening the saving dialog
00836   dlg = (GtkFileChooserDialog *)
00837     gtk_file_chooser_dialog_new (_("Save file"),
00838                                  window->window,
00839                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                  _("_Cancel"),
00841                                  GTK_RESPONSE_CANCEL,
00842                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00843   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                   TRUE);
00845   buffer = g_build_filename (input->directory, input->name, NULL);
00846   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847   g_free (buffer);
00848
00849   // Adding XML filter
00850   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851   gtk_file_filter_set_name (filter1, "XML");
00852   gtk_file_filter_add_pattern (filter1, "*.xml");
00853   gtk_file_filter_add_pattern (filter1, "*.XML");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856   // Adding JSON filter
00857   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858   gtk_file_filter_set_name (filter2, "JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.json");
00860   gtk_file_filter_add_pattern (filter2, "*.JSON");
00861   gtk_file_filter_add_pattern (filter2, "*.js");
00862   gtk_file_filter_add_pattern (filter2, "*.JS");
00863   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865   if (input->type == INPUT_TYPE_XML)
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867   else
00868     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870   // If OK response then saving
00871   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873       // Setting input file type
00874       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875       buffer = (char *) gtk_file_filter_get_name (filter1);
00876       if (!strcmp (buffer, "XML"))
00877         input->type = INPUT_TYPE_XML;
00878       else
00879         input->type = INPUT_TYPE_JSON;
00880
00881       // Adding properties to the root XML node
00882       input->simulator = gtk_file_chooser_get_filename
00883         (GTK_FILE_CHOOSER (window->button_simulator));
00884       if (gtk_toggle_button_get_active
00885           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886         input->evaluator = gtk_file_chooser_get_filename
00887           (GTK_FILE_CHOOSER (window->button_evaluator));
00888       else
00889         input->evaluator = NULL;
00890       if (input->type == INPUT_TYPE_XML)
00891         {
00892           input->result
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->entry_result));
00895           input->variables
00896             = (char *) xmlStrdup ((const xmlChar *)
00897                                   gtk_entry_get_text
00898                                   (window->entry_variables));
00899         }
00900       else
00901         {
00902           input->result =
00903             g_strdup (gtk_entry_get_text (window->entry_result));
00904           input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907
```

```
00908       // Setting the algorithm
00909       switch (window_get_algorithm ())
00910         {
00911         case ALGORITHM_MONTE_CARLO:
00912           input->algorithm = ALGORITHM_MONTE_CARLO;
00913           input->nsimulations
00914             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915           input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917           input->tolerance =
00918             gtk_spin_button_get_value (window->spin_tolerance);
00919           input->nbest =
00920             gtk_spin_button_get_value_as_int (window->spin_bests);
00921           window_save_direction ();
00922           break;
00923         case ALGORITHM_SWEEP:
00924           input->algorithm = ALGORITHM_SWEEP;
00925           input->niterations
00926             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927           input->tolerance =
00928             gtk_spin_button_get_value (window->spin_tolerance);
00929           input->nbest =
00930             gtk_spin_button_get_value_as_int (window->spin_bests);
00931           window_save_direction ();
00932           break;
00933         default:
00934           input->algorithm = ALGORITHM_GENETIC;
00935           input->nsimulations
00936             = gtk_spin_button_get_value_as_int (window->spin_population);
00937           input->niterations
00938             = gtk_spin_button_get_value_as_int (window->spin_generations);
00939           input->mutation_ratio
00940             = gtk_spin_button_get_value (window->spin_mutation);
00941           input->reproduction_ratio
00942             = gtk_spin_button_get_value (window->spin_reproduction);
00943           input->adaptation_ratio
00944             = gtk_spin_button_get_value (window->spin_adaptation);
00945           break;
00946         }
00947       input->norm = window_get_norm ();
00948       input->p = gtk_spin_button_get_value (window->spin_p);
00949       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00950
00951       // Saving the XML file
00952       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953       input_save (buffer);
00954
00955       // Closing and freeing memory
00956       g_free (buffer);
00957       gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959       fprintf (stderr, "window_save: end\n");
00960 #endif
00961       return 1;
00962     }
00963
00964   // Closing and freeing memory
00965   gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967   fprintf (stderr, "window_save: end\n");
00968 #endif
00969   return 0;
00970 }
00971
00976 void
00977 window_run ()
00978 {
00979   unsigned int i;
00980   char *msg, *msg2, buffer[64], buffer2[64];
00981 #if DEBUG_INTERFACE
00982   fprintf (stderr, "window_run: start\n");
00983 #endif
00984   if (!window_save ())
00985     {
00986 #if DEBUG_INTERFACE
00987       fprintf (stderr, "window_run: end\n");
00988 #endif
00989       return;
00990     }
00991   running_new ();
00992   while (gtk_events_pending ())
00993     gtk_main_iteration ();
00994   optimize_open ();
00995 #if DEBUG_INTERFACE
00996   fprintf (stderr, "window_run: closing running dialog\n");
00997 #endif
```

```
00998   gtk_spinner_stop (running->spinner);
00999   gtk_widget_destroy (GTK_WIDGET (running->dialog));
01000 #if DEBUG_INTERFACE
01001   fprintf (stderr, "window_run: displaying results\n");
01002 #endif
01003   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01004   msg2 = g_strdup (buffer);
01005   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01006     {
01007       snprintf (buffer, 64, "%s = %s\n",
01008                 input->variable[i].name,
01009                 format[input->variable[i].precision]);
01010       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01011       msg = g_strconcat (msg2, buffer2, NULL);
01012       g_free (msg2);
01013     }
01014   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01015             optimize->calculation_time);
01016   msg = g_strconcat (msg2, buffer, NULL);
01017   g_free (msg2);
01018   show_message (_("Best result"), msg, INFO_TYPE);
01019   g_free (msg);
01020 #if DEBUG_INTERFACE
01021   fprintf (stderr, "window_run: freeing memory\n");
01022 #endif
01023   optimize_free ();
01024 #if DEBUG_INTERFACE
01025   fprintf (stderr, "window_run: end\n");
01026 #endif
01027 }
01028
01033 void
01034 window_help ()
01035 {
01036   char *buffer, *buffer2;
01037 #if DEBUG_INTERFACE
01038   fprintf (stderr, "window_help: start\n");
01039 #endif
01040   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01041                               _("user-manual.pdf"), NULL);
01042   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01043   g_free (buffer2);
01044   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
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
01058 void
01059 window_about ()
01060 {
01061   static const gchar *authors[] = {
01062     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01063     "Borja Latorre Garcés <borja.latorre@csic.es>",
01064     NULL
01065   };
01066 #if DEBUG_INTERFACE
01067   fprintf (stderr, "window_about: start\n");
01068 #endif
01069   gtk_show_about_dialog
01070     (window->window,
01071      "program_name", "MPCOTool",
01072      "comments",
01073      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01074        "A software to perform calibrations or optimizations of empirical"
01075        " parameters"),
01076      "authors", authors,
01077      "translator-credits",
01078      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01079      "(english, french and spanish)\n"
01080      "Uğur Çayoğlu (german)",
01081      "version", "3.4.0",
01082      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01083      "logo", window->logo,
01084      "website", "https://github.com/jburguete/mpcotool",
01085      "license-type", GTK_LICENSE_BSD, NULL);
01086 #if DEBUG_INTERFACE
01087   fprintf (stderr, "window_about: end\n");
01088 #endif
01089 }
01090
01096 void
01097 window_update_direction ()
```

```
01098 {
01099 #if DEBUG_INTERFACE
01100   fprintf (stderr, "window_update_direction: start\n");
01101 #endif
01102   gtk_widget_show (GTK_WIDGET (window->check_direction));
01103   if (gtk_toggle_button_get_active
01104       (GTK_TOGGLE_BUTTON (window->check_direction)))
01105     {
01106       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01107       gtk_widget_show (GTK_WIDGET (window->label_step));
01108       gtk_widget_show (GTK_WIDGET (window->spin_step));
01109     }
01110   switch (window_get_direction ())
01111     {
01112     case DIRECTION_METHOD_COORDINATES:
01113       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01114       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01115       break;
01116     default:
01117       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01118       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01119     }
01120 #if DEBUG_INTERFACE
01121   fprintf (stderr, "window_update_direction: end\n");
01122 #endif
01123 }
01124
01129 void
01130 window_update ()
01131 {
01132   unsigned int i;
01133 #if DEBUG_INTERFACE
01134   fprintf (stderr, "window_update: start\n");
01135 #endif
01136   gtk_widget_set_sensitive
01137     (GTK_WIDGET (window->button_evaluator),
01138      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01139                                    (window->check_evaluator)));
01140   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01142   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01144   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01146   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01147   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01148   gtk_widget_hide (GTK_WIDGET (window->label_population));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01150   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01151   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01152   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01153   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01154   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01155   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01156   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01157   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01158   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01159   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01160   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01161   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01162   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01163   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01164   gtk_widget_hide (GTK_WIDGET (window->label_step));
01165   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01166   gtk_widget_hide (GTK_WIDGET (window->label_p));
01167   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01168   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01169   switch (window_get_algorithm ())
01170     {
01171     case ALGORITHM_MONTE_CARLO:
01172       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01173       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01174       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01175       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01176       if (i > 1)
01177         {
01178           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01179           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01180           gtk_widget_show (GTK_WIDGET (window->label_bests));
01181           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01182         }
01183       window_update_direction ();
01184       break;
01185     case ALGORITHM_SWEEP:
01186       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01187       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01188       if (i > 1)
```

```
01189         {
01190           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01191           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01192           gtk_widget_show (GTK_WIDGET (window->label_bests));
01193           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01194         }
01195       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01196       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01197       gtk_widget_show (GTK_WIDGET (window->check_direction));
01198       window_update_direction ();
01199       break;
01200     default:
01201       gtk_widget_show (GTK_WIDGET (window->label_population));
01202       gtk_widget_show (GTK_WIDGET (window->spin_population));
01203       gtk_widget_show (GTK_WIDGET (window->label_generations));
01204       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01205       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01206       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01207       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01208       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01209       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01210       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01211       gtk_widget_show (GTK_WIDGET (window->label_bits));
01212       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01213     }
01214   gtk_widget_set_sensitive
01215     (GTK_WIDGET (window->button_remove_experiment),
    input->nexperiments > 1);
01216   gtk_widget_set_sensitive
01217     (GTK_WIDGET (window->button_remove_variable), input->
    nvariables > 1);
01218   for (i = 0; i < input->experiment->ninputs; ++i)
01219     {
01220       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01221       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01222       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01223       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01224       g_signal_handler_block
01225         (window->check_template[i], window->id_template[i]);
01226       g_signal_handler_block (window->button_template[i],
01227                               window->id_input[i]);
01228       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01229                                     (window->check_template[i]), 1);
01230       g_signal_handler_unblock (window->button_template[i],
01231                                 window->id_input[i]);
01232       g_signal_handler_unblock (window->check_template[i],
01233                                 window->id_template[i]);
01234     }
01235   if (i > 0)
01236     {
01237       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]),
01238                                 1);
01239       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01240                                 gtk_toggle_button_get_active
01241                                 GTK_TOGGLE_BUTTON (window->check_template
01242                                                    [i - 1]));
01243     }
01244   if (i < MAX_NINPUTS)
01245     {
01246       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01247       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01248       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01249       gtk_widget_set_sensitive
01250         (GTK_WIDGET (window->button_template[i]),
01251          gtk_toggle_button_get_active
01252          GTK_TOGGLE_BUTTON (window->check_template[i]));
01253       g_signal_handler_block
01254         (window->check_template[i], window->id_template[i]);
01255       g_signal_handler_block (window->button_template[i],
01256                               window->id_input[i]);
01257       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01258                                     (window->check_template[i]), 0);
01259       g_signal_handler_unblock (window->button_template[i],
01260                                 window->id_input[i]);
01261       g_signal_handler_unblock (window->check_template[i],
01262                                 window->id_template[i]);
01263     }
01264   while (++i < MAX_NINPUTS)
01265     {
01266       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01267       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01268     }
01269   gtk_widget_set_sensitive
01270     (GTK_WIDGET (window->spin_minabs),
01271      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01272   gtk_widget_set_sensitive
01273     (GTK_WIDGET (window->spin_maxabs),
```

```
01274       gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01275   if (window_get_norm () == ERROR_NORM_P)
01276     {
01277       gtk_widget_show (GTK_WIDGET (window->label_p));
01278       gtk_widget_show (GTK_WIDGET (window->spin_p));
01279     }
01280 #if DEBUG_INTERFACE
01281   fprintf (stderr, "window_update: end\n");
01282 #endif
01283 }
01284
01289 void
01290 window_set_algorithm ()
01291 {
01292   int i;
01293 #if DEBUG_INTERFACE
01294   fprintf (stderr, "window_set_algorithm: start\n");
01295 #endif
01296   i = window_get_algorithm ();
01297   switch (i)
01298     {
01299     case ALGORITHM_SWEEP:
01300       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01301       if (i < 0)
01302         i = 0;
01303       gtk_spin_button_set_value (window->spin_sweeps,
01304                                  (gdouble) input->variable[i].
    nsweeps);
01305       break;
01306     case ALGORITHM_GENETIC:
01307       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01308       if (i < 0)
01309         i = 0;
01310       gtk_spin_button_set_value (window->spin_bits,
01311                                  (gdouble) input->variable[i].nbits);
01312     }
01313   window_update ();
01314 #if DEBUG_INTERFACE
01315   fprintf (stderr, "window_set_algorithm: end\n");
01316 #endif
01317 }
01318
01323 void
01324 window_set_experiment ()
01325 {
01326   unsigned int i, j;
01327   char *buffer1, *buffer2;
01328 #if DEBUG_INTERFACE
01329   fprintf (stderr, "window_set_experiment: start\n");
01330 #endif
01331   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01332   gtk_spin_button_set_value (window->spin_weight,
01333                              input->experiment[i].weight);
01334   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01335   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01336   g_free (buffer1);
01337   g_signal_handler_block
01338     (window->button_experiment, window->id_experiment_name);
01339   gtk_file_chooser_set_filename
01340     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01341   g_signal_handler_unblock
01342     (window->button_experiment, window->id_experiment_name);
01343   g_free (buffer2);
01344   for (j = 0; j < input->experiment->ninputs; ++j)
01345     {
01346       g_signal_handler_block (window->button_template[j],
01347                               window->id_input[j]);
01348       buffer2 =
01349         g_build_filename (input->directory, input->experiment[i].
    template[j],
01350                           NULL);
01351       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01352                                      (window->button_template[j]), buffer2);
01353       g_free (buffer2);
01354       g_signal_handler_unblock
01355         (window->button_template[j], window->id_input[j]);
01356     }
01357 #if DEBUG_INTERFACE
01358   fprintf (stderr, "window_set_experiment: end\n");
01359 #endif
01360 }
01361
01366 void
01367 window_remove_experiment ()
01368 {
01369   unsigned int i, j;
01370 #if DEBUG_INTERFACE
```

```
01371   fprintf (stderr, "window_remove_experiment: start\n");
01372 #endif
01373   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01374   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01375   gtk_combo_box_text_remove (window->combo_experiment, i);
01376   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01377   experiment_free (input->experiment + i, input->
      type);
01378   --input->nexperiments;
01379   for (j = i; j < input->nexperiments; ++j)
01380     memcpy (input->experiment + j, input->experiment + j + 1,
01381             sizeof (Experiment));
01382   j = input->nexperiments - 1;
01383   if (i > j)
01384     i = j;
01385   for (j = 0; j < input->experiment->ninputs; ++j)
01386     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01387   g_signal_handler_block
01388     (window->button_experiment, window->id_experiment_name);
01389   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01390   g_signal_handler_unblock
01391     (window->button_experiment, window->id_experiment_name);
01392   for (j = 0; j < input->experiment->ninputs; ++j)
01393     g_signal_handler_unblock (window->button_template[j],
01394                               window->id_input[j]);
01395   window_update ();
01396 #if DEBUG_INTERFACE
01397   fprintf (stderr, "window_remove_experiment: end\n");
01398 #endif
01399 }
01400
01405 void
01406 window_add_experiment ()
01407 {
01408   unsigned int i, j;
01409 #if DEBUG_INTERFACE
01410   fprintf (stderr, "window_add_experiment: start\n");
01411 #endif
01412   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01413   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01414   gtk_combo_box_text_insert_text
01415     (window->combo_experiment, i, input->experiment[i].
      name);
01416   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01417   input->experiment = (Experiment *) g_realloc
01418     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01419   for (j = input->nexperiments - 1; j > i; --j)
01420     memcpy (input->experiment + j + 1, input->experiment + j,
01421             sizeof (Experiment));
01422   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01423   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01424   if (input->type == INPUT_TYPE_XML)
01425     {
01426       input->experiment[j + 1].name
01427         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01428       for (j = 0; j < input->experiment->ninputs; ++j)
01429         input->experiment[i + 1].template[j]
01430           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      template[j]);
01431     }
01432   else
01433     {
01434       input->experiment[j + 1].name = g_strdup (input->
      experiment[j].name);
01435       for (j = 0; j < input->experiment->ninputs; ++j)
01436         input->experiment[i + 1].template[j]
01437           = g_strdup (input->experiment[i].template[j]);
01438     }
01439   ++input->nexperiments;
01440   for (j = 0; j < input->experiment->ninputs; ++j)
01441     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01442   g_signal_handler_block
01443     (window->button_experiment, window->id_experiment_name);
01444   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01445   g_signal_handler_unblock
01446     (window->button_experiment, window->id_experiment_name);
01447   for (j = 0; j < input->experiment->ninputs; ++j)
```

```
01448     g_signal_handler_unblock (window->button_template[j],
01449                               window->id_input[j]);
01450   window_update ();
01451 #if DEBUG_INTERFACE
01452   fprintf (stderr, "window_add_experiment: end\n");
01453 #endif
01454 }
01455
01460 void
01461 window_name_experiment ()
01462 {
01463   unsigned int i;
01464   char *buffer;
01465   GFile *file1, *file2;
01466 #if DEBUG_INTERFACE
01467   fprintf (stderr, "window_name_experiment: start\n");
01468 #endif
01469   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01470   file1
01471     =
01472     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01473   file2 = g_file_new_for_path (input->directory);
01474   buffer = g_file_get_relative_path (file2, file1);
01475   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01476   gtk_combo_box_text_remove (window->combo_experiment, i);
01477   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01478   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01479   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01480   g_free (buffer);
01481   g_object_unref (file2);
01482   g_object_unref (file1);
01483 #if DEBUG_INTERFACE
01484   fprintf (stderr, "window_name_experiment: end\n");
01485 #endif
01486 }
01487
01492 void
01493 window_weight_experiment ()
01494 {
01495   unsigned int i;
01496 #if DEBUG_INTERFACE
01497   fprintf (stderr, "window_weight_experiment: start\n");
01498 #endif
01499   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01500   input->experiment[i].weight =
01501     gtk_spin_button_get_value (window->spin_weight);
01502 #if DEBUG_INTERFACE
01503   fprintf (stderr, "window_weight_experiment: end\n");
01504 #endif
01505 }
01506
01512 void
01513 window_inputs_experiment ()
01514 {
01515   unsigned int j;
01516 #if DEBUG_INTERFACE
01517   fprintf (stderr, "window_inputs_experiment: start\n");
01518 #endif
01519   j = input->experiment->ninputs - 1;
01520   if (j
01521       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01522                                         (window->check_template[j])))
01523     --input->experiment->ninputs;
01524   if (input->experiment->ninputs < MAX_NINPUTS
01525       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01526                                        (window->check_template[j])))
01527     ++input->experiment->ninputs;
01528   window_update ();
01529 #if DEBUG_INTERFACE
01530   fprintf (stderr, "window_inputs_experiment: end\n");
01531 #endif
01532 }
01533
01541 void
01542 window_template_experiment (void *data)
01543 {
01544   unsigned int i, j;
01545   char *buffer;
01546   GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548   fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550   i = (size_t) data;
01551   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552   file1
```

```
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555   file2 = g_file_new_for_path (input->directory);
01556   buffer = g_file_get_relative_path (file2, file1);
01557   if (input->type == INPUT_TYPE_XML)
01558     input->experiment[j].template[i] =
01559       (char *) xmlStrdup ((xmlChar *) buffer);
01560   else
01561     input->experiment[j].template[i] = g_strdup (buffer);
01562   g_free (buffer);
01563   g_object_unref (file2);
01564   g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566   fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }
01569
01574 void
01575 window_set_variable ()
01576 {
01577   unsigned int i;
01578 #if DEBUG_INTERFACE
01579   fprintf (stderr, "window_set_variable: start\n");
01580 #endif
01581   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01582   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01583   gtk_entry_set_text (window->entry_variable, input->variable[i].
      name);
01584   g_signal_handler_unblock (window->entry_variable,
01585                             window->id_variable_label);
01586   gtk_spin_button_set_value (window->spin_min, input->variable[i].
      rangemin);
01587   gtk_spin_button_set_value (window->spin_max, input->variable[i].
      rangemax);
01588   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01589     {
01590       gtk_spin_button_set_value (window->spin_minabs,
01591                                  input->variable[i].rangeminabs);
01592       gtk_toggle_button_set_active
01593         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01594     }
01595   else
01596     {
01597       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01598       gtk_toggle_button_set_active
01599         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01600     }
01601   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01602     {
01603       gtk_spin_button_set_value (window->spin_maxabs,
01604                                  input->variable[i].rangemaxabs);
01605       gtk_toggle_button_set_active
01606         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01607     }
01608   else
01609     {
01610       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01611       gtk_toggle_button_set_active
01612         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01613     }
01614   gtk_spin_button_set_value (window->spin_precision,
01615                              input->variable[i].precision);
01616   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01617   if (input->nsteps)
01618     gtk_spin_button_set_value (window->spin_step, input->variable[i].
      step);
01619 #if DEBUG_INTERFACE
01620   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01621            input->variable[i].precision);
01622 #endif
01623   switch (window_get_algorithm ())
01624     {
01625     case ALGORITHM_SWEEP:
01626       gtk_spin_button_set_value (window->spin_sweeps,
01627                                  (gdouble) input->variable[i].
      nsweeps);
01628 #if DEBUG_INTERFACE
01629       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01630                input->variable[i].nsweeps);
01631 #endif
01632       break;
01633     case ALGORITHM_GENETIC:
01634       gtk_spin_button_set_value (window->spin_bits,
01635                                  (gdouble) input->variable[i].nbits);
01636 #if DEBUG_INTERFACE
```

```
01637       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01638                input->variable[i].nbits);
01639 #endif
01640       break;
01641     }
01642   window_update ();
01643 #if DEBUG_INTERFACE
01644   fprintf (stderr, "window_set_variable: end\n");
01645 #endif
01646 }
01647
01652 void
01653 window_remove_variable ()
01654 {
01655   unsigned int i, j;
01656 #if DEBUG_INTERFACE
01657   fprintf (stderr, "window_remove_variable: start\n");
01658 #endif
01659   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01660   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01661   gtk_combo_box_text_remove (window->combo_variable, i);
01662   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01663   xmlFree (input->variable[i].name);
01664   --input->nvariables;
01665   for (j = i; j < input->nvariables; ++j)
01666     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01667   j = input->nvariables - 1;
01668   if (i > j)
01669     i = j;
01670   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01671   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01672   g_signal_handler_unblock (window->entry_variable,
01673                             window->id_variable_label);
01674   window_update ();
01675 #if DEBUG_INTERFACE
01676   fprintf (stderr, "window_remove_variable: end\n");
01677 #endif
01678 }
01679
01684 void
01685 window_add_variable ()
01686 {
01687   unsigned int i, j;
01688 #if DEBUG_INTERFACE
01689   fprintf (stderr, "window_add_variable: start\n");
01690 #endif
01691   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01692   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01693   gtk_combo_box_text_insert_text (window->combo_variable, i,
01694                                   input->variable[i].name);
01695   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01696   input->variable = (Variable *) g_realloc
01697     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01698   for (j = input->nvariables - 1; j > i; --j)
01699     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01700   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01701   if (input->type == INPUT_TYPE_XML)
01702     input->variable[j + 1].name
01703       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01704   else
01705     input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
01706   ++input->nvariables;
01707   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01708   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01709   g_signal_handler_unblock (window->entry_variable,
01710                             window->id_variable_label);
01711   window_update ();
01712 #if DEBUG_INTERFACE
01713   fprintf (stderr, "window_add_variable: end\n");
01714 #endif
01715 }
01716
01721 void
01722 window_label_variable ()
01723 {
01724   unsigned int i;
```

```
01725   const char *buffer;
01726 #if DEBUG_INTERFACE
01727   fprintf (stderr, "window_label_variable: start\n");
01728 #endif
01729   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01730   buffer = gtk_entry_get_text (window->entry_variable);
01731   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01732   gtk_combo_box_text_remove (window->combo_variable, i);
01733   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01734   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01735   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01736 #if DEBUG_INTERFACE
01737   fprintf (stderr, "window_label_variable: end\n");
01738 #endif
01739 }
01740
01745 void
01746 window_precision_variable ()
01747 {
01748   unsigned int i;
01749 #if DEBUG_INTERFACE
01750   fprintf (stderr, "window_precision_variable: start\n");
01751 #endif
01752   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01753   input->variable[i].precision
01754     =
01755     (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01756   gtk_spin_button_set_digits (window->spin_min, input->variable[i].
      precision);
01757   gtk_spin_button_set_digits (window->spin_max, input->variable[i].
      precision);
01758   gtk_spin_button_set_digits (window->spin_minabs,
01759                               input->variable[i].precision);
01760   gtk_spin_button_set_digits (window->spin_maxabs,
01761                               input->variable[i].precision);
01762 #if DEBUG_INTERFACE
01763   fprintf (stderr, "window_precision_variable: end\n");
01764 #endif
01765 }
01766
01771 void
01772 window_rangemin_variable ()
01773 {
01774   unsigned int i;
01775 #if DEBUG_INTERFACE
01776   fprintf (stderr, "window_rangemin_variable: start\n");
01777 #endif
01778   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01779   input->variable[i].rangemin = gtk_spin_button_get_value (window->
      spin_min);
01780 #if DEBUG_INTERFACE
01781   fprintf (stderr, "window_rangemin_variable: end\n");
01782 #endif
01783 }
01784
01789 void
01790 window_rangemax_variable ()
01791 {
01792   unsigned int i;
01793 #if DEBUG_INTERFACE
01794   fprintf (stderr, "window_rangemax_variable: start\n");
01795 #endif
01796   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01797   input->variable[i].rangemax = gtk_spin_button_get_value (window->
      spin_max);
01798 #if DEBUG_INTERFACE
01799   fprintf (stderr, "window_rangemax_variable: end\n");
01800 #endif
01801 }
01802
01807 void
01808 window_rangeminabs_variable ()
01809 {
01810   unsigned int i;
01811 #if DEBUG_INTERFACE
01812   fprintf (stderr, "window_rangeminabs_variable: start\n");
01813 #endif
01814   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01815   input->variable[i].rangeminabs
01816     = gtk_spin_button_get_value (window->spin_minabs);
01817 #if DEBUG_INTERFACE
01818   fprintf (stderr, "window_rangeminabs_variable: end\n");
01819 #endif
01820 }
01821
```

```
01826 void
01827 window_rangemaxabs_variable ()
01828 {
01829   unsigned int i;
01830 #if DEBUG_INTERFACE
01831   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01832 #endif
01833   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01834   input->variable[i].rangemaxabs
01835     = gtk_spin_button_get_value (window->spin_maxabs);
01836 #if DEBUG_INTERFACE
01837   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01838 #endif
01839 }
01840
01845 void
01846 window_step_variable ()
01847 {
01848   unsigned int i;
01849 #if DEBUG_INTERFACE
01850   fprintf (stderr, "window_step_variable: start\n");
01851 #endif
01852   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01853   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01854 #if DEBUG_INTERFACE
01855   fprintf (stderr, "window_step_variable: end\n");
01856 #endif
01857 }
01858
01863 void
01864 window_update_variable ()
01865 {
01866   int i;
01867 #if DEBUG_INTERFACE
01868   fprintf (stderr, "window_update_variable: start\n");
01869 #endif
01870   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01871   if (i < 0)
01872     i = 0;
01873   switch (window_get_algorithm ())
01874     {
01875     case ALGORITHM_SWEEP:
01876       input->variable[i].nsweeps
01877         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01878 #if DEBUG_INTERFACE
01879       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01880                input->variable[i].nsweeps);
01881 #endif
01882       break;
01883     case ALGORITHM_GENETIC:
01884       input->variable[i].nbits
01885         = gtk_spin_button_get_value_as_int (window->spin_bits);
01886 #if DEBUG_INTERFACE
01887       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01888                input->variable[i].nbits);
01889 #endif
01890     }
01891 #if DEBUG_INTERFACE
01892   fprintf (stderr, "window_update_variable: end\n");
01893 #endif
01894 }
01895
01903 int
01904 window_read (char *filename)
01905 {
01906   unsigned int i;
01907   char *buffer;
01908 #if DEBUG_INTERFACE
01909   fprintf (stderr, "window_read: start\n");
01910 #endif
01911
01912   // Reading new input file
01913   input_free ();
01914   if (!input_open (filename))
01915     {
01916 #if DEBUG_INTERFACE
01917       fprintf (stderr, "window_read: end\n");
01918 #endif
01919       return 0;
01920     }
01921
01922   // Setting GTK+ widgets data
01923   gtk_entry_set_text (window->entry_result, input->result);
01924   gtk_entry_set_text (window->entry_variables, input->
      variables);
01925   buffer = g_build_filename (input->directory, input->
```

```
simulator, NULL);
01926   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01927                                  (window->button_simulator), buffer);
01928   g_free (buffer);
01929   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01930                                 (size_t) input->evaluator);
01931   if (input->evaluator)
01932     {
01933       buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01934       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01935                                      (window->button_evaluator), buffer);
01936       g_free (buffer);
01937     }
01938   gtk_toggle_button_set_active
01939     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01940   switch (input->algorithm)
01941     {
01942     case ALGORITHM_MONTE_CARLO:
01943       gtk_spin_button_set_value (window->spin_simulations,
01944                                  (gdouble) input->nsimulations);
01945     case ALGORITHM_SWEEP:
01946       gtk_spin_button_set_value (window->spin_iterations,
01947                                  (gdouble) input->niterations);
01948       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01949       gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01950       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01951                                     (window->check_direction),
input->nsteps);
01952       if (input->nsteps)
01953         {
01954           gtk_toggle_button_set_active
01955             (GTK_TOGGLE_BUTTON (window->button_direction
01956                                 [input->direction]), TRUE);
01957           gtk_spin_button_set_value (window->spin_steps,
01958                                      (gdouble) input->nsteps);
01959           gtk_spin_button_set_value (window->spin_relaxation,
01960                                      (gdouble) input->relaxation);
01961           switch (input->direction)
01962             {
01963             case DIRECTION_METHOD_RANDOM:
01964               gtk_spin_button_set_value (window->spin_estimates,
01965                                          (gdouble) input->nestimates);
01966             }
01967         }
01968       break;
01969     default:
01970       gtk_spin_button_set_value (window->spin_population,
01971                                  (gdouble) input->nsimulations);
01972       gtk_spin_button_set_value (window->spin_generations,
01973                                  (gdouble) input->niterations);
01974       gtk_spin_button_set_value (window->spin_mutation,
01975                                  input->mutation_ratio);
01976       gtk_spin_button_set_value (window->spin_reproduction,
01977                                  input->reproduction_ratio);
01978       gtk_spin_button_set_value (window->spin_adaptation,
01979                                  input->adaptation_ratio);
01980     }
01981   gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983   gtk_spin_button_set_value (window->spin_p, input->p);
01984   gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985   g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986   g_signal_handler_block (window->button_experiment,
01987                           window->id_experiment_name);
01988   gtk_combo_box_text_remove_all (window->combo_experiment);
01989   for (i = 0; i < input->nexperiments; ++i)
01990     gtk_combo_box_text_append_text (window->combo_experiment,
01991                                     input->experiment[i].name);
01992   g_signal_handler_unblock
01993     (window->button_experiment, window->id_experiment_name);
01994   g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01995   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01996   g_signal_handler_block (window->combo_variable, window->
id_variable);
01997   g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01998   gtk_combo_box_text_remove_all (window->combo_variable);
01999   for (i = 0; i < input->nvariables; ++i)
02000     gtk_combo_box_text_append_text (window->combo_variable,
02001                                     input->variable[i].name);
```

```
02002   g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02005   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006   window_set_variable ();
02007   window_update ();
02008
02009 #if DEBUG_INTERFACE
02010   fprintf (stderr, "window_read: end\n");
02011 #endif
02012   return 1;
02013 }
02014
02019 void
02020 window_open ()
02021 {
02022   GtkFileChooserDialog *dlg;
02023   GtkFileFilter *filter;
02024   char *buffer, *directory, *name;
02025
02026 #if DEBUG_INTERFACE
02027   fprintf (stderr, "window_open: start\n");
02028 #endif
02029
02030   // Saving a backup of the current input file
02031   directory = g_strdup (input->directory);
02032   name = g_strdup (input->name);
02033
02034   // Opening dialog
02035   dlg = (GtkFileChooserDialog *)
02036     gtk_file_chooser_dialog_new (_("Open input file"),
02037                                  window->window,
02038                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02039                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02040                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02041
02042   // Adding XML filter
02043   filter = (GtkFileFilter *) gtk_file_filter_new ();
02044   gtk_file_filter_set_name (filter, "XML");
02045   gtk_file_filter_add_pattern (filter, "*.xml");
02046   gtk_file_filter_add_pattern (filter, "*.XML");
02047   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02048
02049   // Adding JSON filter
02050   filter = (GtkFileFilter *) gtk_file_filter_new ();
02051   gtk_file_filter_set_name (filter, "JSON");
02052   gtk_file_filter_add_pattern (filter, "*.json");
02053   gtk_file_filter_add_pattern (filter, "*.JSON");
02054   gtk_file_filter_add_pattern (filter, "*.js");
02055   gtk_file_filter_add_pattern (filter, "*.JS");
02056   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02057
02058   // If OK saving
02059   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02060     {
02061
02062       // Traying to open the input file
02063       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02064       if (!window_read (buffer))
02065         {
02066 #if DEBUG_INTERFACE
02067           fprintf (stderr, "window_open: error reading input file\n");
02068 #endif
02069           g_free (buffer);
02070
02071           // Reading backup file on error
02072           buffer = g_build_filename (directory, name, NULL);
02073           if (!input_open (buffer))
02074             {
02075
02076               // Closing on backup file reading error
02077 #if DEBUG_INTERFACE
02078               fprintf (stderr, "window_read: error reading backup file\n");
02079 #endif
02080               g_free (buffer);
02081               break;
02082             }
02083           g_free (buffer);
02084         }
02085       else
02086         {
02087           g_free (buffer);
02088           break;
02089         }
02090     }
02091
```

```
02092   // Freeing and closing
02093   g_free (name);
02094   g_free (directory);
02095   gtk_widget_destroy (GTK_WIDGET (dlg));
02096 #if DEBUG_INTERFACE
02097   fprintf (stderr, "window_open: end\n");
02098 #endif
02099 }
02100
02107 void
02108 window_new (GtkApplication * application)
02109 {
02110   unsigned int i;
02111   char *buffer, *buffer2, buffer3[64];
02112   char *label_algorithm[NALGORITHMS] = {
02113     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114   };
02115   char *tip_algorithm[NALGORITHMS] = {
02116     _("Monte-Carlo brute force algorithm"),
02117     _("Sweep brute force algorithm"),
02118     _("Genetic algorithm")
02119   };
02120   char *label_direction[NDIRECTIONS] = {
02121     _("_Coordinates descent"), _("_Random")
02122   };
02123   char *tip_direction[NDIRECTIONS] = {
02124     _("Coordinates direction estimate method"),
02125     _("Random direction estimate method")
02126   };
02127   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128   char *tip_norm[NNORMS] = {
02129     _("Euclidean error norm (L2)"),
02130     _("Maximum error norm (L)"),
02131     _("P error norm (Lp)"),
02132     _("Taxicab error norm (L1)")
02133   };
02134
02135 #if DEBUG_INTERFACE
02136   fprintf (stderr, "window_new: start\n");
02137 #endif
02138
02139   // Creating the window
02140   window->window = main_window
02141     = (GtkWindow *) gtk_application_window_new (application);
02142
02143   // Finish when closing the window
02144   g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148   // Setting the window title
02149   gtk_window_set_title (window->window, "MPCOTool");
02150
02151   // Creating the open button
02152   window->button_open = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("document-open",
02154                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157   // Creating the save button
02158   window->button_save = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("document-save",
02160                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02162                     NULL);
02163
02164   // Creating the run button
02165   window->button_run = (GtkToolButton *) gtk_tool_button_new
02166     (gtk_image_new_from_icon_name ("system-run",
02167                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02168   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02169
02170   // Creating the options button
02171   window->button_options = (GtkToolButton *) gtk_tool_button_new
02172     (gtk_image_new_from_icon_name ("preferences-system",
02173                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02174      _("Options"));
02175   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02176
02177   // Creating the help button
02178   window->button_help = (GtkToolButton *) gtk_tool_button_new
02179     (gtk_image_new_from_icon_name ("help-browser",
02180                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02181   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02182
02183   // Creating the about button
```

```
02184   window->button_about = (GtkToolButton *) gtk_tool_button_new
02185     (gtk_image_new_from_icon_name ("help-about",
02186                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02187   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02188
02189   // Creating the exit button
02190   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02191     (gtk_image_new_from_icon_name ("application-exit",
02192                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02193   g_signal_connect_swapped (window->button_exit, "clicked",
02194                             G_CALLBACK (g_application_quit),
02195                             G_APPLICATION (application));
02196
02197   // Creating the buttons bar
02198   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02199   gtk_toolbar_insert
02200     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02201   gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02203   gtk_toolbar_insert
02204     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02205   gtk_toolbar_insert
02206     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02207   gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02209   gtk_toolbar_insert
02210     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02211   gtk_toolbar_insert
02212     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02213   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02214
02215   // Creating the simulator program label and entry
02216   window->label_simulator
02217     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02218   window->button_simulator = (GtkFileChooserButton *)
02219     gtk_file_chooser_button_new (_("Simulator program"),
02220                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02221   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02222                                _("Simulator program executable file"));
02223   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02224
02225   // Creating the evaluator program label and entry
02226   window->check_evaluator = (GtkCheckButton *)
02227     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02228   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02229   window->button_evaluator = (GtkFileChooserButton *)
02230     gtk_file_chooser_button_new (_("Evaluator program"),
02231                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02232   gtk_widget_set_tooltip_text
02233     (GTK_WIDGET (window->button_evaluator),
02234      _("Optional evaluator program executable file"));
02235
02236   // Creating the results files labels and entries
02237   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02238   window->entry_result = (GtkEntry *) gtk_entry_new ();
02239   gtk_widget_set_tooltip_text
02240     (GTK_WIDGET (window->entry_result), _("Best results file"));
02241   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02242   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02245
02246   // Creating the files grid and attaching widgets
02247   window->grid_files = (GtkGrid *) gtk_grid_new ();
02248   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02249                    0, 0, 1, 1);
02250   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02251                    1, 0, 1, 1);
02252   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02253                    0, 1, 1, 1);
02254   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02255                    1, 1, 1, 1);
02256   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02257                    0, 2, 1, 1);
02258   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02259                    1, 2, 1, 1);
02260   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02261                    0, 3, 1, 1);
02262   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
```

```
    entry_variables),
02263                     1, 3, 1, 1);
02264
02265   // Creating the algorithm properties
02266   window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268   window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270   gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272      _("Number of simulations to perform for each iteration"));
02273   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274   window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276   window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278   gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280   g_signal_connect
02281     (window->spin_iterations, "value-changed", window_update, NULL);
02282   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284   window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286   gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288      _("Tolerance to set the variable interval on the next iteration"));
02289   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290   window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294      _("Number of best simulations used to set the variable interval "
02295        "on the next iteration"));
02296   window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298   window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300   gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302      _("Number of population for the genetic algorithm"));
02303   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304   window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306   window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308   gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310      _("Number of generations for the genetic algorithm"));
02311   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312   window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314   gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317   window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319   window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321   gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324   window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326   window->spin_adaptation
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328   gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332   window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335   gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338   window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                      GTK_WIDGET (window->spin_threshold));
02342 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                         GTK_ALIGN_FILL);
02345
02346   // Creating the direction search method properties
02347   window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
```

```
02349   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02350   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02351   window->button_direction[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02355   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02356                     NULL);
02357   for (i = 0; ++i < NDIRECTIONS;)
02358     {
02359       window->button_direction[i] = (GtkRadioButton *)
02360         gtk_radio_button_new_with_mnemonic
02361         (gtk_radio_button_get_group (window->button_direction[0]),
02362          label_direction[i]);
02363       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02364                                    tip_direction[i]);
02365       gtk_grid_attach (window->grid_direction,
02366                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02367       g_signal_connect (window->button_direction[i], "clicked",
02368                         window_update, NULL);
02369     }
02370   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02371   window->spin_steps = (GtkSpinButton *)
02372     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02373   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02374   window->label_estimates
02375     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02376   window->spin_estimates = (GtkSpinButton *)
02377     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02378   window->label_relaxation
02379     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02380   window->spin_relaxation = (GtkSpinButton *)
02381     gtk_spin_button_new_with_range (0., 2., 0.001);
02382   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02383                    0, NDIRECTIONS, 1, 1);
02384   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02385                    1, NDIRECTIONS, 1, 1);
02386   gtk_grid_attach (window->grid_direction,
02387                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02388                    1, 1);
02389   gtk_grid_attach (window->grid_direction,
02390                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02391                    1);
02392   gtk_grid_attach (window->grid_direction,
02393                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394                    1, 1);
02395   gtk_grid_attach (window->grid_direction,
02396                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397                    1, 1);
02398
02399   // Creating the array of algorithms
02400   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401   window->button_algorithm[0] = (GtkRadioButton *)
02402     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404                                tip_algorithm[0]);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407   g_signal_connect (window->button_algorithm[0], "clicked",
02408                     window_set_algorithm, NULL);
02409   for (i = 0; ++i < NALGORITHMS;)
02410     {
02411       window->button_algorithm[i] = (GtkRadioButton *)
02412         gtk_radio_button_new_with_mnemonic
02413         (gtk_radio_button_get_group (window->button_algorithm[0]),
02414          label_algorithm[i]);
02415       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416                                    tip_algorithm[i]);
02417       gtk_grid_attach (window->grid_algorithm,
02418                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419       g_signal_connect (window->button_algorithm[i], "clicked",
02420                         window_set_algorithm, NULL);
02421     }
02422   gtk_grid_attach (window->grid_algorithm,
02423                    GTK_WIDGET (window->label_simulations), 0,
02424                    NALGORITHMS, 1, 1);
02425   gtk_grid_attach (window->grid_algorithm,
02426                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                    1);
02428   gtk_grid_attach (window->grid_algorithm,
02429                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                    1, 1);
02431   gtk_grid_attach (window->grid_algorithm,
```

```
02432                   GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                   1, 1);
02434   gtk_grid_attach (window->grid_algorithm,
02435                   GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                   1, 1);
02437   gtk_grid_attach (window->grid_algorithm,
02438                   GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                   1);
02440   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_bests),
02441                   0, NALGORITHMS + 3, 1, 1);
02442   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_bests), 1,
02443                   NALGORITHMS + 3, 1, 1);
02444   gtk_grid_attach (window->grid_algorithm,
02445                   GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02446                   1, 1);
02447   gtk_grid_attach (window->grid_algorithm,
02448                   GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02449                   1, 1);
02450   gtk_grid_attach (window->grid_algorithm,
02451                   GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02452                   1, 1);
02453   gtk_grid_attach (window->grid_algorithm,
02454                   GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02455                   1, 1);
02456   gtk_grid_attach (window->grid_algorithm,
02457                   GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02458                   1);
02459   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_mutation),
02460                   1, NALGORITHMS + 6, 1, 1);
02461   gtk_grid_attach (window->grid_algorithm,
02462                   GTK_WIDGET (window->label_reproduction), 0,
02463                   NALGORITHMS + 7, 1, 1);
02464   gtk_grid_attach (window->grid_algorithm,
02465                   GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02466                   1, 1);
02467   gtk_grid_attach (window->grid_algorithm,
02468                   GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02469                   1, 1);
02470   gtk_grid_attach (window->grid_algorithm,
02471                   GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02472                   1, 1);
02473   gtk_grid_attach (window->grid_algorithm,
02474                   GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475                   2, 1);
02476   gtk_grid_attach (window->grid_algorithm,
02477                   GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478                   2, 1);
02479   gtk_grid_attach (window->grid_algorithm,
02480                   GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481                   1, 1);
02482   gtk_grid_attach (window->grid_algorithm,
02483                   GTK_WIDGET (window->scrolled_threshold), 1,
02484                   NALGORITHMS + 11, 1, 1);
02485   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487                      GTK_WIDGET (window->grid_algorithm));
02488
02489   // Creating the variable widgets
02490   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491   gtk_widget_set_tooltip_text
02492     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493   window->id_variable = g_signal_connect
02494     (window->combo_variable, "changed", window_set_variable, NULL);
02495   window->button_add_variable
02496     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497                                                    GTK_ICON_SIZE_BUTTON);
02498   g_signal_connect
02499     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02500   gtk_widget_set_tooltip_text
02501     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02502   window->button_remove_variable
02503     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02504                                                    GTK_ICON_SIZE_BUTTON);
02505   g_signal_connect
02506     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02507   gtk_widget_set_tooltip_text
02508     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02509   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02510   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02511   gtk_widget_set_tooltip_text
02512     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02513   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
```

```
02514   window->id_variable_label = g_signal_connect
02515     (window->entry_variable, "changed", window_label_variable, NULL);
02516   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02517   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02518     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_min),
02521      _("Minimum initial value of the variable"));
02522   window->scrolled_min
02523     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02524   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02525                      GTK_WIDGET (window->spin_min));
02526   g_signal_connect (window->spin_min, "value-changed",
02527                     window_rangemin_variable, NULL);
02528   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02529   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02530     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02531   gtk_widget_set_tooltip_text
02532     (GTK_WIDGET (window->spin_max),
02533      _("Maximum initial value of the variable"));
02534   window->scrolled_max
02535     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02537                      GTK_WIDGET (window->spin_max));
02538   g_signal_connect (window->spin_max, "value-changed",
02539                     window_rangemax_variable, NULL);
02540   window->check_minabs = (GtkCheckButton *)
02541     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02542   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02543   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02544     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02545   gtk_widget_set_tooltip_text
02546     (GTK_WIDGET (window->spin_minabs),
02547      _("Minimum allowed value of the variable"));
02548   window->scrolled_minabs
02549     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02550   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02551                      GTK_WIDGET (window->spin_minabs));
02552   g_signal_connect (window->spin_minabs, "value-changed",
02553                     window_rangeminabs_variable, NULL);
02554   window->check_maxabs = (GtkCheckButton *)
02555     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02556   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02557   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02558     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02559   gtk_widget_set_tooltip_text
02560     (GTK_WIDGET (window->spin_maxabs),
02561      _("Maximum allowed value of the variable"));
02562   window->scrolled_maxabs
02563     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565                      GTK_WIDGET (window->spin_maxabs));
02566   g_signal_connect (window->spin_maxabs, "value-changed",
02567                     window_rangemaxabs_variable, NULL);
02568   window->label_precision
02569     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570   window->spin_precision = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572   gtk_widget_set_tooltip_text
02573     (GTK_WIDGET (window->spin_precision),
02574      _("Number of precision floating point digits\n"
02575        "0 is for integer numbers"));
02576   g_signal_connect (window->spin_precision, "value-changed",
02577                     window_precision_variable, NULL);
02578   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579   window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582                                _("Number of steps sweeping the variable"));
02583   g_signal_connect (window->spin_sweeps, "value-changed",
02584                     window_update_variable, NULL);
02585   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586   window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588   gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591   g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL);
02593   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596   gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598      _("Initial step size for the direction search method"));
02599   window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
```

```
02601   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602                      GTK_WIDGET (window->spin_step));
02603   g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608   gtk_grid_attach (window->grid_variable,
02609                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610   gtk_grid_attach (window->grid_variable,
02611                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612   gtk_grid_attach (window->grid_variable,
02613                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614   gtk_grid_attach (window->grid_variable,
02615                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616   gtk_grid_attach (window->grid_variable,
02617                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618   gtk_grid_attach (window->grid_variable,
02619                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620   gtk_grid_attach (window->grid_variable,
02621                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622   gtk_grid_attach (window->grid_variable,
02623                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624   gtk_grid_attach (window->grid_variable,
02625                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626   gtk_grid_attach (window->grid_variable,
02627                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628   gtk_grid_attach (window->grid_variable,
02629                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630   gtk_grid_attach (window->grid_variable,
02631                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632   gtk_grid_attach (window->grid_variable,
02633                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634   gtk_grid_attach (window->grid_variable,
02635                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636   gtk_grid_attach (window->grid_variable,
02637                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638   gtk_grid_attach (window->grid_variable,
02639                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640   gtk_grid_attach (window->grid_variable,
02641                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642   gtk_grid_attach (window->grid_variable,
02643                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644   gtk_grid_attach (window->grid_variable,
02645                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646   gtk_grid_attach (window->grid_variable,
02647                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                      GTK_WIDGET (window->grid_variable));
02651
02652   // Creating the experiment widgets
02653   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                                _("Experiment selector"));
02656   window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
02658   window->button_add_experiment
02659     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660                                                    GTK_ICON_SIZE_BUTTON);
02661   g_signal_connect
02662     (window->button_add_experiment, "clicked",
       window_add_experiment, NULL);
02663   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664                                _("Add experiment"));
02665   window->button_remove_experiment
02666     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667                                                    GTK_ICON_SIZE_BUTTON);
02668   g_signal_connect (window->button_remove_experiment, "clicked",
02669                     window_remove_experiment, NULL);
02670   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02671                                _("Remove experiment"));
02672   window->label_experiment
02673     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674   window->button_experiment = (GtkFileChooserButton *)
02675     gtk_file_chooser_button_new (_("Experimental data file"),
02676                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02677   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678                                _("Experimental data file"));
02679   window->id_experiment_name
02680     = g_signal_connect (window->button_experiment, "selection-changed",
02681                         window_name_experiment, NULL);
02682   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684   window->spin_weight
02685     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
```

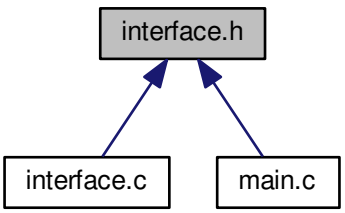
```
02686   gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->spin_weight),
02688      _("Weight factor to build the objective function"));
02689   g_signal_connect
02690     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
02691   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692   gtk_grid_attach (window->grid_experiment,
02693                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694   gtk_grid_attach (window->grid_experiment,
02695                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696   gtk_grid_attach (window->grid_experiment,
02697                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02698   gtk_grid_attach (window->grid_experiment,
02699                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700   gtk_grid_attach (window->grid_experiment,
02701                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702   gtk_grid_attach (window->grid_experiment,
02703                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704   gtk_grid_attach (window->grid_experiment,
02705                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706   for (i = 0; i < MAX_NINPUTS; ++i)
02707     {
02708       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709       window->check_template[i] = (GtkCheckButton *)
02710         gtk_check_button_new_with_label (buffer3);
02711       window->id_template[i]
02712         = g_signal_connect (window->check_template[i], "toggled",
02713                             window_inputs_experiment, NULL);
02714       gtk_grid_attach (window->grid_experiment,
02715                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716                        1);
02717       window->button_template[i] =
02718         (GtkFileChooserButton *)
02719         gtk_file_chooser_button_new (_("Input template"),
02720                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02721       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722                                    _("Experimental input template file"));
02723       window->id_input[i] =
02724         g_signal_connect_swapped (window->button_template[i],
02725                                   "selection-changed",
02726                                   (void (*)) window_template_experiment,
02727                                   (void *) (size_t) i);
02728       gtk_grid_attach (window->grid_experiment,
02729                        GTK_WIDGET (window->button_template[i]),
02730                        1, 3 + i, 3, 1);
02731     }
02732   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734                      GTK_WIDGET (window->grid_experiment));
02735
02736   // Creating the error norm widgets
02737   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740                      GTK_WIDGET (window->grid_norm));
02741   window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02744                                tip_norm[0]);
02745   gtk_grid_attach (window->grid_norm,
02746                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02748   for (i = 0; ++i < NNORMS;)
02749     {
02750       window->button_norm[i] = (GtkRadioButton *)
02751         gtk_radio_button_new_with_mnemonic
02752         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02753       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02754                                    tip_norm[i]);
02755       gtk_grid_attach (window->grid_norm,
02756                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02757       g_signal_connect (window->button_norm[i], "clicked",
      window_update,
02758                         NULL);
02759     }
02760   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02761   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1,
02762                    1);
02763   window->spin_p =
02764     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02765                                                       G_MAXDOUBLE, 0.01);
02766   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02767                                _("P parameter for the P error norm"));
02768   window->scrolled_p =
02769     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02770   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
```

```
02771                     GTK_WIDGET (window->spin_p));
02772   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02773   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02774   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02775                    1, 2, 1, 2);
02776
02777   // Creating the grid and attaching the widgets to the grid
02778   window->grid = (GtkGrid *) gtk_grid_new ();
02779   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02780                    1);
02781   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02782   gtk_grid_attach (window->grid,
02783                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02784   gtk_grid_attach (window->grid,
02785                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02786   gtk_grid_attach (window->grid,
02787                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02788   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02789   gtk_container_add (GTK_CONTAINER (window->window),
02790                      GTK_WIDGET (window->grid));
02791
02792   // Setting the window logo
02793   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02794   gtk_window_set_icon (window->window, window->logo);
02795
02796   // Showing the window
02797   gtk_widget_show_all (GTK_WIDGET (window->window));
02798
02799   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02800 #if GTK_MINOR_VERSION >= 16
02801   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02802   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02803   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02804   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02805   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808                                40);
02809 #endif
02810
02811   // Reading initial example
02812   input_new ();
02813   buffer2 = g_get_current_dir ();
02814   buffer =
02815     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02816   g_free (buffer2);
02817   window_read (buffer);
02818   g_free (buffer);
02819
02820 #if DEBUG_INTERFACE
02821   fprintf (stderr, "window_new: start\n");
02822 #endif
02823 }
```

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)

  Function to get the active GtkRadioButton.
- void input_save (char *filename)

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

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

  Function to open the running dialog.
- unsigned int window_get_algorithm ()

  Function to get the stochastic algorithm number.
- unsigned int window_get_direction ()

  Function to get the direction search method number.
- unsigned int window_get_norm ()

  Function to get the norm method number.
- void window_save_direction ()

  Function to save the direction search method data in the input file.
- int window_save ()

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

  Function to run a optimization.
- void window_help ()

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

  Function to update direction search method widgets view in the main window.
- void window_update ()

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

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

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

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

Function to add an experiment in the main window.

- void window_name_experiment ()

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Function to open the main window.

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

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.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 567 of file utils.c.

```
00568 {
00569   unsigned int i;
00570   for (i = 0; i < n; ++i)
00571     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572       break;
00573   return i;
00574 }
```

4.13.2.2 input_save()

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

Function to save the input file.

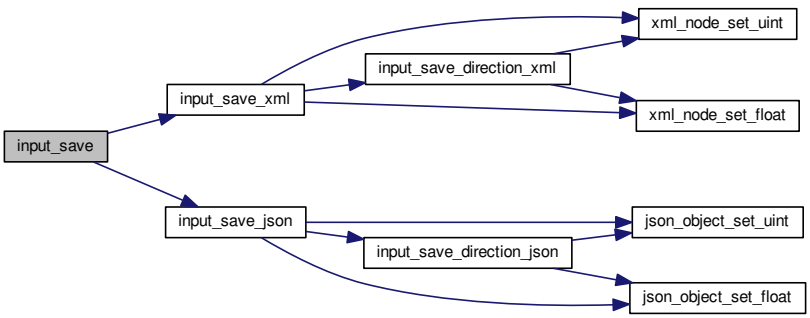
Parameters

filename	Input file name.

Definition at line 579 of file interface.c.

```
00580 {
00581   xmlDoc *doc;
00582   JsonGenerator *generator;
00583
00584 #if DEBUG_INTERFACE
00585   fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588   // Getting the input file directory
00589   input->name = g_path_get_basename (filename);
00590   input->directory = g_path_get_dirname (filename);
00591
00592   if (input->type == INPUT_TYPE_XML)
00593     {
00594       // Opening the input file
00595       doc = xmlNewDoc ((const xmlChar *) "1.0");
00596       input_save_xml (doc);
00597
00598       // Saving the XML file
00599       xmlSaveFormatFile (filename, doc, 1);
00600
00601       // Freeing memory
00602       xmlFreeDoc (doc);
00603     }
00604   else
00605     {
00606       // Opening the input file
00607       generator = json_generator_new ();
00608       json_generator_set_pretty (generator, TRUE);
00609       input_save_json (generator);
00610
00611       // Saving the JSON file
00612       json_generator_to_file (generator, filename, NULL);
00613
00614       // Freeing memory
00615       g_object_unref (generator);
00616     }
00617
00618 #if DEBUG_INTERFACE
00619   fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }
```

Here is the call graph for this function:



4.13.2.3 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:

4.13.2.4 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:

4.13.2.5 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

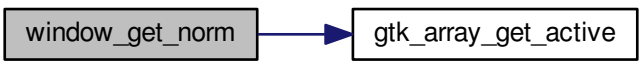
Returns

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:



4.13.2.6 window_new()

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

Function to open the main window.

Parameters

application	GtkApplication struct.

Definition at line 2108 of file interface.c.

```
02109 {
02110   unsigned int i;
02111   char *buffer, *buffer2, buffer3[64];
02112   char *label_algorithm[NALGORITHMS] = {
02113     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114   };
02115   char *tip_algorithm[NALGORITHMS] = {
02116     _("Monte-Carlo brute force algorithm"),
02117     _("Sweep brute force algorithm"),
02118     _("Genetic algorithm")
02119   };
02120   char *label_direction[NDIRECTIONS] = {
02121     _("_Coordinates descent"), _("_Random")
02122   };
02123   char *tip_direction[NDIRECTIONS] = {
02124     _("Coordinates direction estimate method"),
02125     _("Random direction estimate method")
02126   };
02127   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128   char *tip_norm[NNORMS] = {
02129     _("Euclidean error norm (L2)"),
02130     _("Maximum error norm (L)"),
02131     _("P error norm (Lp)"),
02132     _("Taxicab error norm (L1)")
02133   };
02134
02135 #if DEBUG_INTERFACE
02136   fprintf (stderr, "window_new: start\n");
02137 #endif
02138
02139   // Creating the window
02140   window->window = main_window
02141     = (GtkWindow *) gtk_application_window_new (application);
02142
02143   // Finish when closing the window
02144   g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148   // Setting the window title
02149   gtk_window_set_title (window->window, "MPCOTool");
02150
02151   // Creating the open button
02152   window->button_open = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("document-open",
02154                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157   // Creating the save button
02158   window->button_save = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("document-save",
02160                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02162                     NULL);
02163
02164   // Creating the run button
02165   window->button_run = (GtkToolButton *) gtk_tool_button_new
02166     (gtk_image_new_from_icon_name ("system-run",
02167                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02168   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02169
02170   // Creating the options button
02171   window->button_options = (GtkToolButton *) gtk_tool_button_new
```

```
02172     (gtk_image_new_from_icon_name ("preferences-system",
02173                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02174      _("Options"));
02175   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02176
02177   // Creating the help button
02178   window->button_help = (GtkToolButton *) gtk_tool_button_new
02179     (gtk_image_new_from_icon_name ("help-browser",
02180                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02181   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02182
02183   // Creating the about button
02184   window->button_about = (GtkToolButton *) gtk_tool_button_new
02185     (gtk_image_new_from_icon_name ("help-about",
02186                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02187   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02188
02189   // Creating the exit button
02190   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02191     (gtk_image_new_from_icon_name ("application-exit",
02192                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02193   g_signal_connect_swapped (window->button_exit, "clicked",
02194                             G_CALLBACK (g_application_quit),
02195                             G_APPLICATION (application));
02196
02197   // Creating the buttons bar
02198   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02199   gtk_toolbar_insert
02200     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02201   gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02203   gtk_toolbar_insert
02204     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
02205   gtk_toolbar_insert
02206     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02207   gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02209   gtk_toolbar_insert
02210     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02211   gtk_toolbar_insert
02212     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02213   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02214
02215   // Creating the simulator program label and entry
02216   window->label_simulator
02217     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02218   window->button_simulator = (GtkFileChooserButton *)
02219     gtk_file_chooser_button_new (_("Simulator program"),
02220                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02221   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02222                                _("Simulator program executable file"));
02223   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02224
02225   // Creating the evaluator program label and entry
02226   window->check_evaluator = (GtkCheckButton *)
02227     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02228   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02229   window->button_evaluator = (GtkFileChooserButton *)
02230     gtk_file_chooser_button_new (_("Evaluator program"),
02231                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02232   gtk_widget_set_tooltip_text
02233     (GTK_WIDGET (window->button_evaluator),
02234      _("Optional evaluator program executable file"));
02235
02236   // Creating the results files labels and entries
02237   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02238   window->entry_result = (GtkEntry *) gtk_entry_new ();
02239   gtk_widget_set_tooltip_text
02240     (GTK_WIDGET (window->entry_result), _("Best results file"));
02241   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02242   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02245
02246   // Creating the files grid and attaching widgets
02247   window->grid_files = (GtkGrid *) gtk_grid_new ();
02248   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
```

```
      label_simulator),
02249                     0, 0, 1, 1);
02250   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02251                     1, 0, 1, 1);
02252   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02253                     0, 1, 1, 1);
02254   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02255                     1, 1, 1, 1);
02256   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02257                     0, 2, 1, 1);
02258   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02259                     1, 2, 1, 1);
02260   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02261                     0, 3, 1, 1);
02262   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02263                     1, 3, 1, 1);
02264
02265   // Creating the algorithm properties
02266   window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268   window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270   gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272      _("Number of simulations to perform for each iteration"));
02273   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274   window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276   window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278   gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280   g_signal_connect
02281     (window->spin_iterations, "value-changed",
      window_update, NULL);
02282   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284   window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286   gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288      _("Tolerance to set the variable interval on the next iteration"));
02289   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290   window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294      _("Number of best simulations used to set the variable interval "
02295        "on the next iteration"));
02296   window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298   window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300   gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302      _("Number of population for the genetic algorithm"));
02303   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304   window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306   window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308   gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310      _("Number of generations for the genetic algorithm"));
02311   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312   window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314   gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317   window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319   window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321   gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324   window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326   window->spin_adaptation
```

```
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328   gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332   window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335   gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338   window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                      GTK_WIDGET (window->spin_threshold));
02342 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                         GTK_ALIGN_FILL);
02345
02346   // Creating the direction search method properties
02347   window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02349   g_signal_connect (window->check_direction, "clicked",
    window_update, NULL);
02350   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02351   window->button_direction[0] = (GtkRadioButton *)
02352     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02355   g_signal_connect (window->button_direction[0], "clicked",
    window_update,
02356                     NULL);
02357   for (i = 0; ++i < NDIRECTIONS;)
02358     {
02359       window->button_direction[i] = (GtkRadioButton *)
02360         gtk_radio_button_new_with_mnemonic
02361         (gtk_radio_button_get_group (window->button_direction[0]),
02362          label_direction[i]);
02363       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02364                                    tip_direction[i]);
02365       gtk_grid_attach (window->grid_direction,
02366                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02367       g_signal_connect (window->button_direction[i], "clicked",
02368                         window_update, NULL);
02369     }
02370   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02371   window->spin_steps = (GtkSpinButton *)
02372     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02373   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02374   window->label_estimates
02375     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02376   window->spin_estimates = (GtkSpinButton *)
02377     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02378   window->label_relaxation
02379     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02380   window->spin_relaxation = (GtkSpinButton *)
02381     gtk_spin_button_new_with_range (0., 2., 0.001);
02382   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->label_steps),
02383                    0, NDIRECTIONS, 1, 1);
02384   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->spin_steps),
02385                    1, NDIRECTIONS, 1, 1);
02386   gtk_grid_attach (window->grid_direction,
02387                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02388                    1, 1);
02389   gtk_grid_attach (window->grid_direction,
02390                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02391                    1);
02392   gtk_grid_attach (window->grid_direction,
02393                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394                    1, 1);
02395   gtk_grid_attach (window->grid_direction,
02396                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397                    1, 1);
02398
02399   // Creating the array of algorithms
02400   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401   window->button_algorithm[0] = (GtkRadioButton *)
02402     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404                                tip_algorithm[0]);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407   g_signal_connect (window->button_algorithm[0], "clicked",
02408                     window_set_algorithm, NULL);
02409   for (i = 0; ++i < NALGORITHMS;)
```

```
02410     {
02411       window->button_algorithm[i] = (GtkRadioButton *)
02412         gtk_radio_button_new_with_mnemonic
02413         (gtk_radio_button_get_group (window->button_algorithm[0]),
02414          label_algorithm[i]);
02415       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416                                    tip_algorithm[i]);
02417       gtk_grid_attach (window->grid_algorithm,
02418                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419       g_signal_connect (window->button_algorithm[i], "clicked",
02420                         window_set_algorithm, NULL);
02421     }
02422   gtk_grid_attach (window->grid_algorithm,
02423                    GTK_WIDGET (window->label_simulations), 0,
02424                    NALGORITHMS, 1, 1);
02425   gtk_grid_attach (window->grid_algorithm,
02426                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                    1);
02428   gtk_grid_attach (window->grid_algorithm,
02429                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                    1, 1);
02431   gtk_grid_attach (window->grid_algorithm,
02432                    GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                    1, 1);
02434   gtk_grid_attach (window->grid_algorithm,
02435                    GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                    1, 1);
02437   gtk_grid_attach (window->grid_algorithm,
02438                    GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                    1);
02440   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests),
02441                    0, NALGORITHMS + 3, 1, 1);
02442   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02443                    NALGORITHMS + 3, 1, 1);
02444   gtk_grid_attach (window->grid_algorithm,
02445                    GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02446                    1, 1);
02447   gtk_grid_attach (window->grid_algorithm,
02448                    GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02449                    1, 1);
02450   gtk_grid_attach (window->grid_algorithm,
02451                    GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02452                    1, 1);
02453   gtk_grid_attach (window->grid_algorithm,
02454                    GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02455                    1, 1);
02456   gtk_grid_attach (window->grid_algorithm,
02457                    GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02458                    1);
02459   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02460                    1, NALGORITHMS + 6, 1, 1);
02461   gtk_grid_attach (window->grid_algorithm,
02462                    GTK_WIDGET (window->label_reproduction), 0,
02463                    NALGORITHMS + 7, 1, 1);
02464   gtk_grid_attach (window->grid_algorithm,
02465                    GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02466                    1, 1);
02467   gtk_grid_attach (window->grid_algorithm,
02468                    GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02469                    1, 1);
02470   gtk_grid_attach (window->grid_algorithm,
02471                    GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02472                    1, 1);
02473   gtk_grid_attach (window->grid_algorithm,
02474                    GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475                    2, 1);
02476   gtk_grid_attach (window->grid_algorithm,
02477                    GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478                    2, 1);
02479   gtk_grid_attach (window->grid_algorithm,
02480                    GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481                    1, 1);
02482   gtk_grid_attach (window->grid_algorithm,
02483                    GTK_WIDGET (window->scrolled_threshold), 1,
02484                    NALGORITHMS + 11, 1, 1);
02485   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487                      GTK_WIDGET (window->grid_algorithm));
02488
02489   // Creating the variable widgets
02490   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491   gtk_widget_set_tooltip_text
02492     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493   window->id_variable = g_signal_connect
```

```
02494     (window->combo_variable, "changed", window_set_variable, NULL);
02495   window->button_add_variable
02496     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497                                                    GTK_ICON_SIZE_BUTTON);
02498   g_signal_connect
02499     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02500   gtk_widget_set_tooltip_text
02501     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02502   window->button_remove_variable
02503     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02504                                                    GTK_ICON_SIZE_BUTTON);
02505   g_signal_connect
02506     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02507   gtk_widget_set_tooltip_text
02508     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02509   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02510   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02511   gtk_widget_set_tooltip_text
02512     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02513   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02514   window->id_variable_label = g_signal_connect
02515     (window->entry_variable, "changed",
      window_label_variable, NULL);
02516   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02517   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02518     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02519   gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_min),
02521      _("Minimum initial value of the variable"));
02522   window->scrolled_min
02523     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02524   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02525                      GTK_WIDGET (window->spin_min));
02526   g_signal_connect (window->spin_min, "value-changed",
02527                     window_rangemin_variable, NULL);
02528   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02529   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02530     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02531   gtk_widget_set_tooltip_text
02532     (GTK_WIDGET (window->spin_max),
02533      _("Maximum initial value of the variable"));
02534   window->scrolled_max
02535     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02536   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02537                      GTK_WIDGET (window->spin_max));
02538   g_signal_connect (window->spin_max, "value-changed",
02539                     window_rangemax_variable, NULL);
02540   window->check_minabs = (GtkCheckButton *)
02541     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02542   g_signal_connect (window->check_minabs, "toggled",
      window_update, NULL);
02543   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02544     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02545   gtk_widget_set_tooltip_text
02546     (GTK_WIDGET (window->spin_minabs),
02547      _("Minimum allowed value of the variable"));
02548   window->scrolled_minabs
02549     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02550   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02551                      GTK_WIDGET (window->spin_minabs));
02552   g_signal_connect (window->spin_minabs, "value-changed",
02553                     window_rangeminabs_variable, NULL);
02554   window->check_maxabs = (GtkCheckButton *)
02555     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02556   g_signal_connect (window->check_maxabs, "toggled",
      window_update, NULL);
02557   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02558     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02559   gtk_widget_set_tooltip_text
02560     (GTK_WIDGET (window->spin_maxabs),
02561      _("Maximum allowed value of the variable"));
02562   window->scrolled_maxabs
02563     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565                      GTK_WIDGET (window->spin_maxabs));
02566   g_signal_connect (window->spin_maxabs, "value-changed",
02567                     window_rangemaxabs_variable, NULL);
02568   window->label_precision
02569     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570   window->spin_precision = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572   gtk_widget_set_tooltip_text
02573     (GTK_WIDGET (window->spin_precision),
02574      _("Number of precision floating point digits\n"
02575        "0 is for integer numbers"));
```

```
02576   g_signal_connect (window->spin_precision, "value-changed",
02577                     window_precision_variable, NULL);
02578   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579   window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582                                _("Number of steps sweeping the variable"));
02583   g_signal_connect (window->spin_sweeps, "value-changed",
02584                     window_update_variable, NULL);
02585   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586   window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588   gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591   g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
    ;
02593   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596   gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598      _("Initial step size for the direction search method"));
02599   window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602                      GTK_WIDGET (window->spin_step));
02603   g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608   gtk_grid_attach (window->grid_variable,
02609                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610   gtk_grid_attach (window->grid_variable,
02611                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612   gtk_grid_attach (window->grid_variable,
02613                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614   gtk_grid_attach (window->grid_variable,
02615                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616   gtk_grid_attach (window->grid_variable,
02617                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618   gtk_grid_attach (window->grid_variable,
02619                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620   gtk_grid_attach (window->grid_variable,
02621                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622   gtk_grid_attach (window->grid_variable,
02623                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624   gtk_grid_attach (window->grid_variable,
02625                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626   gtk_grid_attach (window->grid_variable,
02627                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628   gtk_grid_attach (window->grid_variable,
02629                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630   gtk_grid_attach (window->grid_variable,
02631                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632   gtk_grid_attach (window->grid_variable,
02633                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634   gtk_grid_attach (window->grid_variable,
02635                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636   gtk_grid_attach (window->grid_variable,
02637                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638   gtk_grid_attach (window->grid_variable,
02639                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640   gtk_grid_attach (window->grid_variable,
02641                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642   gtk_grid_attach (window->grid_variable,
02643                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644   gtk_grid_attach (window->grid_variable,
02645                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646   gtk_grid_attach (window->grid_variable,
02647                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                      GTK_WIDGET (window->grid_variable));
02651
02652   // Creating the experiment widgets
02653   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                                _("Experiment selector"));
02656   window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed",
     window_set_experiment, NULL);
02658   window->button_add_experiment
02659     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660                                                    GTK_ICON_SIZE_BUTTON);
```

```
02661   g_signal_connect
02662     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02663   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664                                _("Add experiment"));
02665   window->button_remove_experiment
02666     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667                                                    GTK_ICON_SIZE_BUTTON);
02668   g_signal_connect (window->button_remove_experiment, "clicked",
02669                     window_remove_experiment, NULL);
02670   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02671                                _("Remove experiment"));
02672   window->label_experiment
02673     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674   window->button_experiment = (GtkFileChooserButton *)
02675     gtk_file_chooser_button_new (_("Experimental data file"),
02676                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02677   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678                                _("Experimental data file"));
02679   window->id_experiment_name
02680     = g_signal_connect (window->button_experiment, "selection-changed",
02681                         window_name_experiment, NULL);
02682   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684   window->spin_weight
02685     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686   gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->spin_weight),
02688      _("Weight factor to build the objective function"));
02689   g_signal_connect
02690     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02691   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692   gtk_grid_attach (window->grid_experiment,
02693                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694   gtk_grid_attach (window->grid_experiment,
02695                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696   gtk_grid_attach (window->grid_experiment,
02697                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02698   gtk_grid_attach (window->grid_experiment,
02699                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700   gtk_grid_attach (window->grid_experiment,
02701                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702   gtk_grid_attach (window->grid_experiment,
02703                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704   gtk_grid_attach (window->grid_experiment,
02705                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706   for (i = 0; i < MAX_NINPUTS; ++i)
02707     {
02708       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709       window->check_template[i] = (GtkCheckButton *)
02710         gtk_check_button_new_with_label (buffer3);
02711       window->id_template[i]
02712         = g_signal_connect (window->check_template[i], "toggled",
02713                             window_inputs_experiment, NULL);
02714       gtk_grid_attach (window->grid_experiment,
02715                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716                        1);
02717       window->button_template[i] =
02718         (GtkFileChooserButton *)
02719         gtk_file_chooser_button_new (_("Input template"),
02720                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02721       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722                                    _("Experimental input template file"));
02723       window->id_input[i] =
02724         g_signal_connect_swapped (window->button_template[i],
02725                                   "selection-changed",
02726                                   (void (*)) window_template_experiment,
02727                                   (void *) (size_t) i);
02728       gtk_grid_attach (window->grid_experiment,
02729                        GTK_WIDGET (window->button_template[i]),
02730                        1, 3 + i, 3, 1);
02731     }
02732   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734                      GTK_WIDGET (window->grid_experiment));
02735
02736   // Creating the error norm widgets
02737   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740                      GTK_WIDGET (window->grid_norm));
02741   window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
```

```
02744                                tip_norm[0]);
02745   gtk_grid_attach (window->grid_norm,
02746                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02748   for (i = 0; ++i < NNORMS;)
02749     {
02750       window->button_norm[i] = (GtkRadioButton *)
02751         gtk_radio_button_new_with_mnemonic
02752         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02753       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02754                                    tip_norm[i]);
02755       gtk_grid_attach (window->grid_norm,
02756                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02757       g_signal_connect (window->button_norm[i], "clicked",
      window_update,
02758                         NULL);
02759     }
02760   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02761   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1,
02762                    1);
02763   window->spin_p =
02764     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02765                                                       G_MAXDOUBLE, 0.01);
02766   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02767                                _("P parameter for the P error norm"));
02768   window->scrolled_p =
02769     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02770   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02771                      GTK_WIDGET (window->spin_p));
02772   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02773   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02774   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      scrolled_p),
02775                    1, 2, 1, 2);
02776
02777   // Creating the grid and attaching the widgets to the grid
02778   window->grid = (GtkGrid *) gtk_grid_new ();
02779   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02780                    1);
02781   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02782   gtk_grid_attach (window->grid,
02783                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02784   gtk_grid_attach (window->grid,
02785                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02786   gtk_grid_attach (window->grid,
02787                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02788   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02789   gtk_container_add (GTK_CONTAINER (window->window),
02790                      GTK_WIDGET (window->grid));
02791
02792   // Setting the window logo
02793   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02794   gtk_window_set_icon (window->window, window->logo);
02795
02796   // Showing the window
02797   gtk_widget_show_all (GTK_WIDGET (window->window));
02798
02799   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02800 #if GTK_MINOR_VERSION >= 16
02801   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02802   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02803   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02804   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02805   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808                                40);
02809 #endif
02810
02811   // Reading initial example
02812   input_new ();
02813   buffer2 = g_get_current_dir ();
02814   buffer =
02815     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02816   g_free (buffer2);
02817   window_read (buffer);
02818   g_free (buffer);
02819
02820 #if DEBUG_INTERFACE
02821   fprintf (stderr, "window_new: start\n");
02822 #endif
02823 }
```

4.13.2.7 window_read()

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

Function to read the input data of a file.

Parameters

filename	File name.

Returns

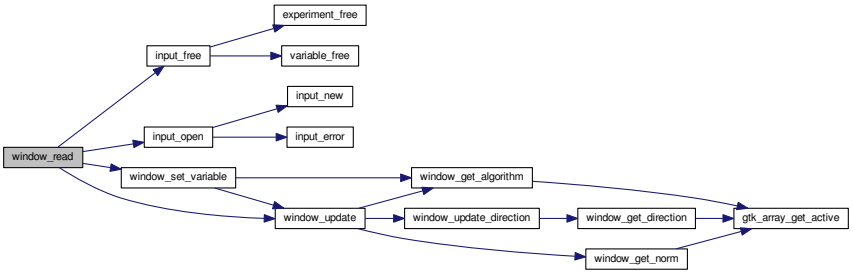
1 on succes, 0 on error.

Definition at line 1904 of file interface.c.

```
01905 {
01906   unsigned int i;
01907   char *buffer;
01908 #if DEBUG_INTERFACE
01909   fprintf (stderr, "window_read: start\n");
01910 #endif
01911
01912   // Reading new input file
01913   input_free ();
01914   if (!input_open (filename))
01915     {
01916 #if DEBUG_INTERFACE
01917       fprintf (stderr, "window_read: end\n");
01918 #endif
01919       return 0;
01920     }
01921
01922   // Setting GTK+ widgets data
01923   gtk_entry_set_text (window->entry_result, input->result);
01924   gtk_entry_set_text (window->entry_variables, input->
      variables);
01925   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01926   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01927                                  (window->button_simulator), buffer);
01928   g_free (buffer);
01929   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01930                                 (size_t) input->evaluator);
01931   if (input->evaluator)
01932     {
01933       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01934       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01935                                      (window->button_evaluator), buffer);
01936       g_free (buffer);
01937     }
01938   gtk_toggle_button_set_active
01939     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01940   switch (input->algorithm)
01941     {
01942     case ALGORITHM_MONTE_CARLO:
01943       gtk_spin_button_set_value (window->spin_simulations,
01944                                  (gdouble) input->nsimulations);
01945     case ALGORITHM_SWEEP:
01946       gtk_spin_button_set_value (window->spin_iterations,
01947                                  (gdouble) input->niterations);
01948       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01949       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01950       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01951                                     (window->check_direction),
      input->nsteps);
01952       if (input->nsteps)
01953         {
01954           gtk_toggle_button_set_active
```

```
01955             (GTK_TOGGLE_BUTTON (window->button_direction
01956                                 [input->direction]), TRUE);
01957           gtk_spin_button_set_value (window->spin_steps,
01958                                      (gdouble) input->nsteps);
01959           gtk_spin_button_set_value (window->spin_relaxation,
01960                                      (gdouble) input->relaxation);
01961           switch (input->direction)
01962             {
01963             case DIRECTION_METHOD_RANDOM:
01964               gtk_spin_button_set_value (window->spin_estimates,
01965                                          (gdouble) input->nestimates);
01966             }
01967         }
01968       break;
01969     default:
01970       gtk_spin_button_set_value (window->spin_population,
01971                                  (gdouble) input->nsimulations);
01972       gtk_spin_button_set_value (window->spin_generations,
01973                                  (gdouble) input->niterations);
01974       gtk_spin_button_set_value (window->spin_mutation,
01975                                  input->mutation_ratio);
01976       gtk_spin_button_set_value (window->spin_reproduction,
01977                                  input->reproduction_ratio);
01978       gtk_spin_button_set_value (window->spin_adaptation,
01979                                  input->adaptation_ratio);
01980     }
01981   gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983   gtk_spin_button_set_value (window->spin_p, input->p);
01984   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01985   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01986   g_signal_handler_block (window->button_experiment,
01987                           window->id_experiment_name);
01988   gtk_combo_box_text_remove_all (window->combo_experiment);
01989   for (i = 0; i < input->nexperiments; ++i)
01990     gtk_combo_box_text_append_text (window->combo_experiment,
01991                                     input->experiment[i].name);
01992   g_signal_handler_unblock
01993     (window->button_experiment, window->
      id_experiment_name);
01994   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01995   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01996   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01997   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01998   gtk_combo_box_text_remove_all (window->combo_variable);
01999   for (i = 0; i < input->nvariables; ++i)
02000     gtk_combo_box_text_append_text (window->combo_variable,
02001                                     input->variable[i].name);
02002   g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
02005   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006   window_set_variable ();
02007   window_update ();
02008
02009 #if DEBUG_INTERFACE
02010   fprintf (stderr, "window_read: end\n");
02011 #endif
02012   return 1;
02013 }
```

Here is the call graph for this function:



4.13.2.8 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file interface.c.

```
00826 {
00827   GtkFileChooserDialog *dlg;
00828   GtkFileFilter *filter1, *filter2;
00829   char *buffer;
00830
00831 #if DEBUG_INTERFACE
00832   fprintf (stderr, "window_save: start\n");
00833 #endif
00834
00835   // Opening the saving dialog
00836   dlg = (GtkFileChooserDialog *)
00837     gtk_file_chooser_dialog_new (_("Save file"),
00838                                  window->window,
00839                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                  _("_Cancel"),
00841                                  GTK_RESPONSE_CANCEL,
00842                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00843   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844                                                   TRUE);
00845   buffer = g_build_filename (input->directory, input->name, NULL);
00846   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847   g_free (buffer);
00848
00849   // Adding XML filter
00850   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851   gtk_file_filter_set_name (filter1, "XML");
00852   gtk_file_filter_add_pattern (filter1, "*.xml");
00853   gtk_file_filter_add_pattern (filter1, "*.XML");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856   // Adding JSON filter
00857   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858   gtk_file_filter_set_name (filter2, "JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.json");
00860   gtk_file_filter_add_pattern (filter2, "*.JSON");
00861   gtk_file_filter_add_pattern (filter2, "*.js");
00862   gtk_file_filter_add_pattern (filter2, "*.JS");
00863   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
```

```
00864
00865   if (input->type == INPUT_TYPE_XML)
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867   else
00868     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870   // If OK response then saving
00871   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872     {
00873       // Setting input file type
00874       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875       buffer = (char *) gtk_file_filter_get_name (filter1);
00876       if (!strcmp (buffer, "XML"))
00877         input->type = INPUT_TYPE_XML;
00878       else
00879         input->type = INPUT_TYPE_JSON;
00880
00881       // Adding properties to the root XML node
00882       input->simulator = gtk_file_chooser_get_filename
00883         (GTK_FILE_CHOOSER (window->button_simulator));
00884       if (gtk_toggle_button_get_active
00885           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886         input->evaluator = gtk_file_chooser_get_filename
00887           (GTK_FILE_CHOOSER (window->button_evaluator));
00888       else
00889         input->evaluator = NULL;
00890       if (input->type == INPUT_TYPE_XML)
00891         {
00892           input->result
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                   gtk_entry_get_text (window->entry_result));
00895           input->variables
00896             = (char *) xmlStrdup ((const xmlChar *)
00897                                   gtk_entry_get_text
00898                                   (window->entry_variables));
00899         }
00900       else
00901         {
00902           input->result =
00903             g_strdup (gtk_entry_get_text (window->entry_result));
00904           input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906         }
00907
00908       // Setting the algorithm
00909       switch (window_get_algorithm ())
00910         {
00911         case ALGORITHM_MONTE_CARLO:
00912           input->algorithm = ALGORITHM_MONTE_CARLO;
00913           input->nsimulations
00914             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915           input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917           input->tolerance =
00918             gtk_spin_button_get_value (window->spin_tolerance);
00919           input->nbest =
00920             gtk_spin_button_get_value_as_int (window->spin_bests);
00921           window_save_direction ();
00922           break;
00923         case ALGORITHM_SWEEP:
00924           input->algorithm = ALGORITHM_SWEEP;
00925           input->niterations
00926             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927           input->tolerance =
00928             gtk_spin_button_get_value (window->spin_tolerance);
00929           input->nbest =
00930             gtk_spin_button_get_value_as_int (window->spin_bests);
00931           window_save_direction ();
00932           break;
00933         default:
00934           input->algorithm = ALGORITHM_GENETIC;
00935           input->nsimulations
00936             = gtk_spin_button_get_value_as_int (window->spin_population);
00937           input->niterations
00938             = gtk_spin_button_get_value_as_int (window->spin_generations);
00939           input->mutation_ratio
00940             = gtk_spin_button_get_value (window->spin_mutation);
00941           input->reproduction_ratio
00942             = gtk_spin_button_get_value (window->spin_reproduction);
00943           input->adaptation_ratio
00944             = gtk_spin_button_get_value (window->spin_adaptation);
00945           break;
00946         }
00947       input->norm = window_get_norm ();
00948       input->p = gtk_spin_button_get_value (window->spin_p);
00949       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
```

```
00950
00951       // Saving the XML file
00952       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953       input_save (buffer);
00954
00955       // Closing and freeing memory
00956       g_free (buffer);
00957       gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959       fprintf (stderr, "window_save: end\n");
00960 #endif
00961       return 1;
00962     }
00963
00964   // Closing and freeing memory
00965   gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967   fprintf (stderr, "window_save: end\n");
00968 #endif
00969   return 0;
00970 }
```

4.13.2.9 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 1542 of file interface.c.

```
01543 {
01544   unsigned int i, j;
01545   char *buffer;
01546   GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548   fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550   i = (size_t) data;
01551   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552   file1
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555   file2 = g_file_new_for_path (input->directory);
01556   buffer = g_file_get_relative_path (file2, file1);
01557   if (input->type == INPUT_TYPE_XML)
01558     input->experiment[j].template[i] =
01559       (char *) xmlStrdup ((xmlChar *) buffer);
01560   else
01561     input->experiment[j].template[i] = g_strdup (buffer);
01562   g_free (buffer);
01563   g_object_unref (file2);
01564   g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566   fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }
```

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 2012-2016, 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 dependency graph for main.c:

Macros

- #define DEBUG_MAIN 0

  Macro to debug main functions.

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 2012-2016, 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
00070 #define DEBUG_MAIN 0
00071
00072
00081 #if EXTERNAL_LIBRARY
00082 int
00083 mpcotool (int argn, char **argc)
00084 #else
00085 int
00086 main (int argn, char **argc)
00087 #endif
00088 {
00089 #if HAVE_GTK
00090   GtkApplication *application;
00091   char *buffer;
00092 #endif
00093
00094   // Starting pseudo-random numbers generator
00095 #if DEBUG_MAIN
00096   fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00097 #endif
00098   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00099
00100   // Allowing spaces in the XML data file
00101 #if DEBUG_MAIN
00102   fprintf (stderr, "main: allowing spaces in the XML data file\n");
00103 #endif
00104   xmlKeepBlanksDefault (0);
00105
00106   // Starting MPI
00107 #if HAVE_MPI
00108 #if DEBUG_MAIN
00109   fprintf (stderr, "main: starting MPI\n");
00110 #endif
00111   MPI_Init (&argn, &argc);
00112   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00113   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00114   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00115 #else
00116   ntasks = 1;
00117 #endif
00118
00119   // Resetting result and variables file names
00120 #if DEBUG_MAIN
```

```
00121   fprintf (stderr, "main: resetting result and variables file names\n");
00122 #endif
00123   input->result = input->variables = NULL;
00124
00125 #if HAVE_GTK
00126
00127   // Getting threads number and pseudo-random numbers generator seed
00128   nthreads_direction = nthreads = cores_number ();
00129   optimize->seed = DEFAULT_RANDOM_SEED;
00130
00131   // Setting local language and international floating point numbers notation
00132   setlocale (LC_ALL, "");
00133   setlocale (LC_NUMERIC, "C");
00134   window->application_directory = g_get_current_dir ();
00135   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
00136   bindtextdomain (PROGRAM_INTERFACE, buffer);
00137   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00138   textdomain (PROGRAM_INTERFACE);
00139
00140   // Initing GTK+
00141 #if !EXTERNAL_LIBRARY
00142   show_pending = process_pending;
00143 #endif
00144   gtk_disable_setlocale ();
00145   application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00146                                      G_APPLICATION_FLAGS_NONE);
00147   g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00148
00149   // Opening the main window
00150   g_application_run (G_APPLICATION (application), 0, NULL);
00151
00152   // Freeing memory
00153   input_free ();
00154   g_free (buffer);
00155   gtk_widget_destroy (GTK_WIDGET (window->window));
00156   g_object_unref (application);
00157   g_free (window->application_directory);
00158
00159 #else
00160
00161   // Checking syntax
00162   if (argn < 2)
00163     {
00164       printf ("The syntax is:\n"
00165               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166               "[variables_file]\n");
00167       return 1;
00168     }
00169
00170   // Getting threads number and pseudo-random numbers generator seed
00171 #if DEBUG_MAIN
00172   fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00173            "generator seed\n");
00174 #endif
00175   nthreads_direction = nthreads = cores_number ();
00176   optimize->seed = DEFAULT_RANDOM_SEED;
00177   if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00178     {
00179       nthreads_direction = nthreads = atoi (argc[2]);
00180       if (!nthreads)
00181         {
00182           printf ("Bad threads number\n");
00183           return 2;
00184         }
00185       argc += 2;
00186       argn -= 2;
00187       if (argn > 2 && !strcmp (argc[1], "-seed"))
00188         {
00189           optimize->seed = atoi (argc[2]);
00190           argc += 2;
00191           argn -= 2;
00192         }
00193     }
00194   else if (argn > 2 && !strcmp (argc[1], "-seed"))
00195     {
00196       optimize->seed = atoi (argc[2]);
00197       argc += 2;
00198       argn -= 2;
00199       if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00200         {
00201           nthreads_direction = nthreads = atoi (argc[2]);
00202           if (!nthreads)
00203             {
00204               printf ("Bad threads number\n");
00205               return 2;
00206             }
```

```
00207           argc += 2;
00208           argn -= 2;
00209         }
00210     }
00211   printf ("nthreads=%u\n", nthreads);
00212   printf ("seed=%lu\n", optimize->seed);
00213
00214   // Checking arguments
00215 #if DEBUG_MAIN
00216   fprintf (stderr, "main: checking arguments\n");
00217 #endif
00218   if (argn > 4 || argn < 2)
00219     {
00220       printf ("The syntax is:\n"
00221               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00222               "[variables_file]\n");
00223       return 1;
00224     }
00225   if (argn > 2)
00226     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00227   if (argn == 4)
00228     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00229
00230   // Making optimization
00231 #if DEBUG_MAIN
00232   fprintf (stderr, "main: making optimization\n");
00233 #endif
00234   if (input_open (argc[1]))
00235     optimize_open ();
00236
00237   // Freeing memory
00238 #if DEBUG_MAIN
00239   fprintf (stderr, "main: freeing memory and closing\n");
00240 #endif
00241   optimize_free ();
00242
00243 #endif
00244
00245   // Closing MPI
00246 #if HAVE_MPI
00247   MPI_Finalize ();
00248 #endif
00249
00250   // Freeing memory
00251   gsl_rng_free (optimize->rng);
00252
00253   // Closing
00254   return 0;
00255 }
```

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 *template)

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

  Function to parse input files, simulating and calculating the \ objective function.
- double optimize_norm_euclidian (unsigned int simulation)

  Function to calculate the Euclidian error norm.
- double optimize_norm_maximum (unsigned int simulation)

  Function to calculate the maximum error norm.
- double optimize_norm_p (unsigned int simulation)

  Function to calculate the P error norm.
- double optimize_norm_taxicab (unsigned int simulation)

  Function to calculate the taxicab error norm.
- void optimize_print ()

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

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

  Function to save the best simulations.
- void optimize_sequential ()

  Function to optimize sequentially.
- void * optimize_thread (ParallelData *data)

  Function to optimize on a thread.
- void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 optimization results.
- void optimize_synchronise ()

  Function to synchronise the optimization results of MPI tasks.
- void optimize_sweep ()

  Function to optimize with the sweep algorithm.
- void optimize_MonteCarlo ()

  Function to optimize with the Monte-Carlo algorithm.
- void optimize_best_direction (unsigned int simulation, double value)

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

- void optimize_direction_sequential (unsigned int simulation)

  Function to estimate the direction search sequentially.
- void * optimize_direction_thread (ParallelData *data)

  Function to estimate the direction search on a thread.
- double optimize_estimate_direction_random (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- void optimize_step_direction (unsigned int simulation)

  Function to do a step of the direction search method.
- void optimize_direction ()

  Function to optimize with a direction search method.
- double optimize_genetic_objective (**Entity** *entity)

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

  Function to optimize with the genetic algorithm.
- void optimize_save_old ()

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

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

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

  Function to do a step of the iterative algorithm.
- void optimize_iterate ()

  Function to iterate the algorithm.
- void optimize_free ()

  Function to free the memory used by the Optimize struct.
- void optimize_open ()

  Function to open and perform a optimization.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- unsigned int nthreads_direction

  Number of threads for the direction search method.
- GMutex mutex [1]

  Mutex struct.
- 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 468 of file optimize.c.

```
00469 {
00470   unsigned int i, j;
00471   double e;
00472 #if DEBUG_OPTIMIZE
00473   fprintf (stderr, "optimize_best: start\n");
00474   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475            optimize->nsaveds, optimize->nbest);
00476 #endif
00477   if (optimize->nsaveds < optimize->nbest
00478       || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480       if (optimize->nsaveds < optimize->nbest)
00481         ++optimize->nsaveds;
00482       optimize->error_best[optimize->nsaveds - 1] = value;
00483       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484       for (i = optimize->nsaveds; --i;)
00485         {
00486           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00487             {
00488               j = optimize->simulation_best[i];
00489               e = optimize->error_best[i];
00490               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
```

```
00491               optimize->error_best[i] = optimize->
      error_best[i - 1];
00492               optimize->simulation_best[i - 1] = j;
00493               optimize->error_best[i - 1] = e;
00494             }
00495           else
00496             break;
00497         }
00498     }
00499 #if DEBUG_OPTIMIZE
00500   fprintf (stderr, "optimize_best: end\n");
00501 #endif
00502 }
```

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 795 of file optimize.c.

```
00796 {
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_best_direction: start\n");
00799   fprintf (stderr,
00800            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801            simulation, value, optimize->error_best[0]);
00802 #endif
00803   if (value < optimize->error_best[0])
00804     {
00805       optimize->error_best[0] = value;
00806       optimize->simulation_best[0] = simulation;
00807 #if DEBUG_OPTIMIZE
00808       fprintf (stderr,
00809                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810                simulation, value);
00811 #endif
00812     }
00813 #if DEBUG_OPTIMIZE
00814   fprintf (stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }
```

4.17.2.3 optimize_direction_sequential()

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

Function to estimate the direction search sequentially.

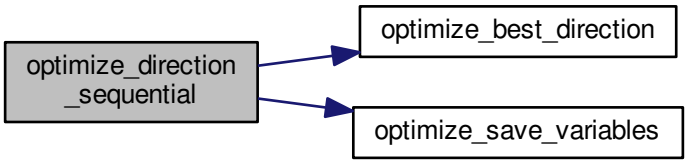
Parameters

simulation	Simulation number.

Definition at line 825 of file optimize.c.

```
00826 {
00827   unsigned int i, j;
00828   double e;
00829 #if DEBUG_OPTIMIZE
00830   fprintf (stderr, "optimize_direction_sequential: start\n");
00831   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832            "nend_direction=%u\n",
00833            optimize->nstart_direction, optimize->
      nend_direction);
00834 #endif
00835   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837       j = simulation + i;
00838       e = optimize_norm (j);
00839       optimize_best_direction (j, e);
00840       optimize_save_variables (j, e);
00841       if (e < optimize->threshold)
00842         {
00843           optimize->stop = 1;
00844           break;
00845         }
00846 #if DEBUG_OPTIMIZE
00847       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00848 #endif
00849     }
00850 #if DEBUG_OPTIMIZE
00851   fprintf (stderr, "optimize_direction_sequential: end\n");
00852 #endif
00853 }
```

Here is the call graph for this function:



4.17.2.4 optimize_direction_thread()

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

Function to estimate the direction search on a thread.

Parameters

data	Function data.

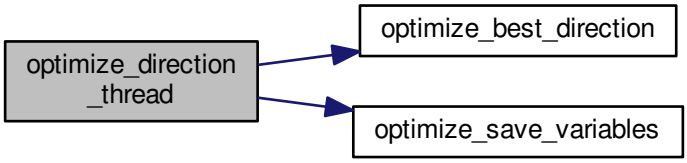
Returns

NULL

Definition at line 863 of file optimize.c.

```
00864 {
00865   unsigned int i, thread;
00866   double e;
00867 #if DEBUG_OPTIMIZE
00868   fprintf (stderr, "optimize_direction_thread: start\n");
00869 #endif
00870   thread = data->thread;
00871 #if DEBUG_OPTIMIZE
00872   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873            thread,
00874            optimize->thread_direction[thread],
00875            optimize->thread_direction[thread + 1]);
00876 #endif
00877   for (i = optimize->thread_direction[thread];
00878        i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880       e = optimize_norm (i);
00881       g_mutex_lock (mutex);
00882       optimize_best_direction (i, e);
00883       optimize_save_variables (i, e);
00884       if (e < optimize->threshold)
00885         optimize->stop = 1;
00886       g_mutex_unlock (mutex);
00887       if (optimize->stop)
00888         break;
00889 #if DEBUG_OPTIMIZE
00890       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00891 #endif
00892     }
00893 #if DEBUG_OPTIMIZE
00894   fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896   g_thread_exit (NULL);
00897   return NULL;
00898 }
```

Here is the call graph for this function:



4.17.2.5 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.

```
00939 {
00940   double x;
00941 #if DEBUG_OPTIMIZE
00942   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00943 #endif
00944   x = optimize->direction[variable];
00945   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947       if (estimate & 1)
00948         x += optimize->step[variable];
00949       else
00950         x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953   fprintf (stderr,
00954            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00955            variable, x);
00956   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00957 #endif
00958   return x;
00959 }
```

4.17.2.6 optimize_estimate_direction_random()

```
double optimize_estimate_direction_random (
             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 910 of file optimize.c.

```
00912 {
00913   double x;
00914 #if DEBUG_OPTIMIZE
00915   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916 #endif
00917   x = optimize->direction[variable]
00918     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
     step[variable];
00919 #if DEBUG_OPTIMIZE
00920   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00921            variable, x);
00922   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00923 #endif
00924   return x;
00925 }
```

4.17.2.7 optimize_genetic_objective()

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

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

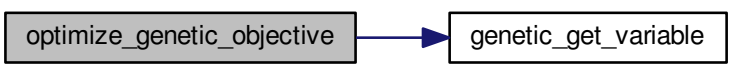
Returns

objective function value.

Definition at line 1104 of file optimize.c.

```
01105 {
01106   unsigned int j;
01107   double objective;
01108   char buffer[64];
01109 #if DEBUG_OPTIMIZE
01110   fprintf (stderr, "optimize_genetic_objective: start\n");
01111 #endif
01112   for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114       optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117   objective = optimize_norm (entity->id);
01118   g_mutex_lock (mutex);
01119   for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122       fprintf (optimize->file_variables, buffer,
01123                genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125   fprintf (optimize->file_variables, "%.14le\n", objective);
01126   g_mutex_unlock (mutex);
01127 #if DEBUG_OPTIMIZE
01128   fprintf (stderr, "optimize_genetic_objective: end\n");
01129 #endif
01130   return objective;
01131 }
```

Here is the call graph for this function:



4.17.2.8 optimize_input()

```
void optimize_input (
             unsigned int simulation,
             char * input,
             GMappedFile * template )
```

Function to write the simulation input file.

Parameters

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

Definition at line 104 of file optimize.c.

```
00105 {
00106   unsigned int i;
00107   char buffer[32], value[32], *buffer2, *buffer3, *content;
00108   FILE *file;
00109   gsize length;
00110   GRegex *regex;
00111
00112 #if DEBUG_OPTIMIZE
00113   fprintf (stderr, "optimize_input: start\n");
00114 #endif
00115
00116   // Checking the file
00117   if (!template)
00118     goto optimize_input_end;
00119
00120   // Opening template
00121   content = g_mapped_file_get_contents (template);
00122   length = g_mapped_file_get_length (template);
00123 #if DEBUG_OPTIMIZE
00124   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125            content);
00126 #endif
00127   file = g_fopen (input, "w");
00128
00129   // Parsing template
00130   for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132 #if DEBUG_OPTIMIZE
00133       fprintf (stderr, "optimize_input: variable=%u\n", i);
00134 #endif
00135       snprintf (buffer, 32, "@variable%u@", i + 1);
00136       regex = g_regex_new (buffer, 0, 0, NULL);
00137       if (i == 0)
00138         {
00139           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                              optimize->label[i], 0, NULL);
00141 #if DEBUG_OPTIMIZE
00142           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143 #endif
00144         }
00145       else
00146         {
00147           length = strlen (buffer3);
00148           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                              optimize->label[i], 0, NULL);
00150           g_free (buffer3);
00151         }
00152       g_regex_unref (regex);
00153       length = strlen (buffer2);
00154       snprintf (buffer, 32, "@value%u@", i + 1);
00155       regex = g_regex_new (buffer, 0, 0, NULL);
00156       snprintf (value, 32, format[optimize->precision[i]],
00157                 optimize->value[simulation * optimize->
      nvariables + i]);
00158
00159 #if DEBUG_OPTIMIZE
00160       fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                          0, NULL);
00164       g_free (buffer2);
00165       g_regex_unref (regex);
00166     }
00167
00168   // Saving input file
00169   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170   g_free (buffer3);
00171   fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175   fprintf (stderr, "optimize_input: end\n");
```

```
00176 #endif
00177   return;
00178 }
```

4.17.2.9 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 591 of file optimize.c.

```
00593 {
00594   unsigned int i, j, k, s[optimize->nbest];
00595   double e[optimize->nbest];
00596 #if DEBUG_OPTIMIZE
00597   fprintf (stderr, "optimize_merge: start\n");
00598 #endif
00599   i = j = k = 0;
00600   do
00601     {
00602       if (i == optimize->nsaveds)
00603         {
00604           s[k] = simulation_best[j];
00605           e[k] = error_best[j];
00606           ++j;
00607           ++k;
00608           if (j == nsaveds)
00609             break;
00610         }
00611       else if (j == nsaveds)
00612         {
00613           s[k] = optimize->simulation_best[i];
00614           e[k] = optimize->error_best[i];
00615           ++i;
00616           ++k;
00617           if (i == optimize->nsaveds)
00618             break;
00619         }
00620       else if (optimize->error_best[i] > error_best[j])
00621         {
00622           s[k] = simulation_best[j];
00623           e[k] = error_best[j];
00624           ++j;
00625           ++k;
00626         }
00627       else
00628         {
00629           s[k] = optimize->simulation_best[i];
00630           e[k] = optimize->error_best[i];
00631           ++i;
00632           ++k;
00633         }
00634     }
00635   while (k < optimize->nbest);
00636   optimize->nsaveds = k;
00637   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638   memcpy (optimize->error_best, e, k * sizeof (double));
```

```
00639 #if DEBUG_OPTIMIZE
00640   fprintf (stderr, "optimize_merge: end\n");
00641 #endif
00642 }
```

4.17.2.10 optimize_norm_euclidian()

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

Function to calculate the Euclidian error norm.

Parameters

simulation	simulation number.

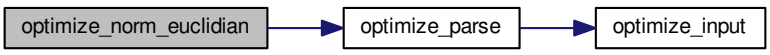
Returns

Euclidian error norm.

Definition at line 300 of file optimize.c.

```
00301 {
00302   double e, ei;
00303   unsigned int i;
00304 #if DEBUG_OPTIMIZE
00305   fprintf (stderr, "optimize_norm_euclidian: start\n");
00306 #endif
00307   e = 0.;
00308   for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310       ei = optimize_parse (simulation, i);
00311       e += ei * ei;
00312     }
00313   e = sqrt (e);
00314 #if DEBUG_OPTIMIZE
00315   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316   fprintf (stderr, "optimize_norm_euclidian: end\n");
00317 #endif
00318   return e;
00319 }
```

Here is the call graph for this function:



4.17.2.11 optimize_norm_maximum()

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

Function to calculate the maximum error norm.

Parameters

simulation	simulation number.

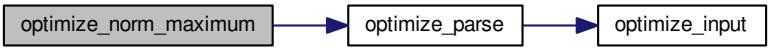
Returns

Maximum error norm.

Definition at line 329 of file optimize.c.

```
00330 {
00331   double e, ei;
00332   unsigned int i;
00333 #if DEBUG_OPTIMIZE
00334   fprintf (stderr, "optimize_norm_maximum: start\n");
00335 #endif
00336   e = 0.;
00337   for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339       ei = fabs (optimize_parse (simulation, i));
00340       e = fmax (e, ei);
00341     }
00342 #if DEBUG_OPTIMIZE
00343   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344   fprintf (stderr, "optimize_norm_maximum: end\n");
00345 #endif
00346   return e;
00347 }
```

Here is the call graph for this function:



4.17.2.12 optimize_norm_p()

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

Function to calculate the P error norm.

Parameters

simulation	simulation number.

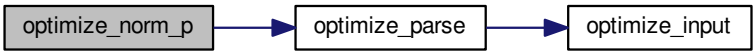
Returns

P error norm.

Definition at line 357 of file optimize.c.

```
00358 {
00359   double e, ei;
00360   unsigned int i;
00361 #if DEBUG_OPTIMIZE
00362   fprintf (stderr, "optimize_norm_p: start\n");
00363 #endif
00364   e = 0.;
00365   for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367       ei = fabs (optimize_parse (simulation, i));
00368       e += pow (ei, optimize->p);
00369     }
00370   e = pow (e, 1. / optimize->p);
00371 #if DEBUG_OPTIMIZE
00372   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373   fprintf (stderr, "optimize_norm_p: end\n");
00374 #endif
00375   return e;
00376 }
```

Here is the call graph for this function:



4.17.2.13 optimize_norm_taxicab()

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

Function to calculate the taxicab error norm.

Parameters

simulation	simulation number.

Returns

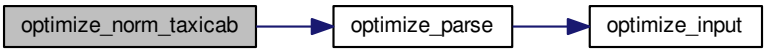
Taxicab error norm.

Definition at line 386 of file optimize.c.

```
00387 {
00388   double e;
00389   unsigned int i;
00390 #if DEBUG_OPTIMIZE
00391   fprintf (stderr, "optimize_norm_taxicab: start\n");
00392 #endif
00393   e = 0.;
00394   for (i = 0; i < optimize->nexperiments; ++i)
```

```
00395     e += fabs (optimize_parse (simulation, i));
00396 #if DEBUG_OPTIMIZE
00397   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398   fprintf (stderr, "optimize_norm_taxicab: end\n");
00399 #endif
00400   return e;
00401 }
```

Here is the call graph for this function:



4.17.2.14 optimize_parse()

```
double optimize_parse (
              unsigned int simulation,
              unsigned int experiment )
```

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 191 of file optimize.c.

```
00192 {
00193   unsigned int i;
00194   double e;
00195   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196     *buffer3, *buffer4;
00197   FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200   fprintf (stderr, "optimize_parse: start\n");
00201   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202            simulation, experiment);
00203 #endif
00204
00205   // Opening input files
00206   for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210 #if DEBUG_OPTIMIZE
00211       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
```

```
00213       optimize_input (simulation, &input[i][0],
00214                       optimize->file[i][experiment]);
00215     }
00216   for (; i < MAX_NINPUTS; ++i)
00217     strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219   fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222   // Performing the simulation
00223   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224   buffer2 = g_path_get_dirname (optimize->simulator);
00225   buffer3 = g_path_get_basename (optimize->simulator);
00226   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230   g_free (buffer4);
00231   g_free (buffer3);
00232   g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234   fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236   system (buffer);
00237
00238   // Checking the objective value function
00239   if (optimize->evaluator)
00240     {
00241       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242       buffer2 = g_path_get_dirname (optimize->evaluator);
00243       buffer3 = g_path_get_basename (optimize->evaluator);
00244       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245       snprintf (buffer, 512, "\"%s\" %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247       g_free (buffer4);
00248       g_free (buffer3);
00249       g_free (buffer2);
00250 #if DEBUG_OPTIMIZE
00251       fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253       system (buffer);
00254       file_result = g_fopen (result, "r");
00255       e = atof (fgets (buffer, 512, file_result));
00256       fclose (file_result);
00257     }
00258   else
00259     {
00260       strcpy (result, "");
00261       file_result = g_fopen (output, "r");
00262       e = atof (fgets (buffer, 512, file_result));
00263       fclose (file_result);
00264     }
00265
00266   // Removing files
00267 #if !DEBUG_OPTIMIZE
00268   for (i = 0; i < optimize->ninputs; ++i)
00269     {
00270       if (optimize->file[i][0])
00271         {
00272           snprintf (buffer, 512, RM " %s", &input[i][0]);
00273           system (buffer);
00274         }
00275     }
00276   snprintf (buffer, 512, RM " %s %s", output, result);
00277   system (buffer);
00278 #endif
00279
00280   // Processing pending events
00281   if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285   fprintf (stderr, "optimize_parse: end\n");
00286 #endif
00287
00288   // Returning the objective function
00289   return e * optimize->weight[experiment];
00290 }
```

Here is the call graph for this function:

4.17.2.15 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 439 of file optimize.c.

```
00440 {
00441   unsigned int i;
00442   char buffer[64];
00443 #if DEBUG_OPTIMIZE
00444   fprintf (stderr, "optimize_save_variables: start\n");
00445 #endif
00446   for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449       fprintf (optimize->file_variables, buffer,
00450                optimize->value[simulation * optimize->
      nvariables + i]);
00451     }
00452   fprintf (optimize->file_variables, "%.14le\n", error);
00453   fflush (optimize->file_variables);
00454 #if DEBUG_OPTIMIZE
00455   fprintf (stderr, "optimize_save_variables: end\n");
00456 #endif
00457 }
```

4.17.2.16 optimize_step_direction()

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

Function to do a step of the direction search method.

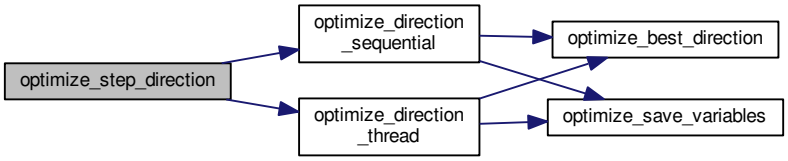
Parameters

simulation	Simulation number.

Definition at line 968 of file optimize.c.

```
00969 {
00970   GThread *thread[nthreads_direction];
00971   ParallelData data[nthreads_direction];
00972   unsigned int i, j, k, b;
00973 #if DEBUG_OPTIMIZE
00974   fprintf (stderr, "optimize_step_direction: start\n");
00975 #endif
00976   for (i = 0; i < optimize->nestimates; ++i)
00977     {
00978       k = (simulation + i) * optimize->nvariables;
00979       b = optimize->simulation_best[0] * optimize->
      nvariables;
00980 #if DEBUG_OPTIMIZE
00981       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00982                simulation + i, optimize->simulation_best[0]);
00983 #endif
00984       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00985         {
00986 #if DEBUG_OPTIMIZE
00987           fprintf (stderr,
00988                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989                    i, j, optimize->value[b]);
00990 #endif
00991           optimize->value[k]
00992             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00993           optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                            optimize->rangeminabs[j]),
00995                                      optimize->rangemaxabs[j]);
00996 #if DEBUG_OPTIMIZE
00997           fprintf (stderr,
00998                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999                    i, j, optimize->value[k]);
01000 #endif
01001         }
01002     }
01003   if (nthreads_direction == 1)
01004     optimize_direction_sequential (simulation);
01005   else
01006     {
01007       for (i = 0; i <= nthreads_direction; ++i)
01008         {
01009           optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01012               / nthreads_direction;
01013 #if DEBUG_OPTIMIZE
01014           fprintf (stderr,
01015                    "optimize_step_direction: i=%u thread_direction=%u\n",
01016                    i, optimize->thread_direction[i]);
01017 #endif
01018         }
01019       for (i = 0; i < nthreads_direction; ++i)
01020         {
01021           data[i].thread = i;
01022           thread[i] = g_thread_new
01023             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01024         }
01025       for (i = 0; i < nthreads_direction; ++i)
01026         g_thread_join (thread[i]);
01027     }
01028 #if DEBUG_OPTIMIZE
01029   fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031 }
```

Here is the call graph for this function:



4.17.2.17 optimize_thread()

```
void * optimize_thread (
            ParallelData * data )
```

Function to optimize on a thread.

Parameters

data	Function data.

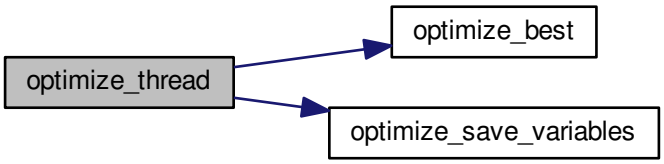
Returns

NULL

Definition at line 545 of file optimize.c.

```
00546 {
00547   unsigned int i, thread;
00548   double e;
00549 #if DEBUG_OPTIMIZE
00550   fprintf (stderr, "optimize_thread: start\n");
00551 #endif
00552   thread = data->thread;
00553 #if DEBUG_OPTIMIZE
00554   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555            optimize->thread[thread], optimize->thread[thread + 1]);
00556 #endif
00557   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559       e = optimize_norm (i);
00560       g_mutex_lock (mutex);
00561       optimize_best (i, e);
00562       optimize_save_variables (i, e);
00563       if (e < optimize->threshold)
00564         optimize->stop = 1;
00565       g_mutex_unlock (mutex);
00566       if (optimize->stop)
00567         break;
00568 #if DEBUG_OPTIMIZE
00569       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570 #endif
00571     }
00572 #if DEBUG_OPTIMIZE
00573   fprintf (stderr, "optimize_thread: end\n");
00574 #endif
00575   g_thread_exit (NULL);
00576   return NULL;
00577 }
```

Here is the call graph for this function:



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 2012-2016, 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 int ntasks;
00079 unsigned int nthreads;
00080 unsigned int nthreads_direction;
00082 GMutex mutex[1];
00083 void (*optimize_algorithm) ();
00085 double (*optimize_estimate_direction) (unsigned int variable,
00086                                        unsigned int estimate);
00088 double (*optimize_norm) (unsigned int simulation);
00090 Optimize optimize[1];
00091
00103 void
00104 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00105 {
00106   unsigned int i;
00107   char buffer[32], value[32], *buffer2, *buffer3, *content;
00108   FILE *file;
00109   gsize length;
00110   GRegex *regex;
00111
00112 #if DEBUG_OPTIMIZE
00113   fprintf (stderr, "optimize_input: start\n");
00114 #endif
00115
00116   // Checking the file
00117   if (!template)
00118     goto optimize_input_end;
00119
00120   // Opening template
00121   content = g_mapped_file_get_contents (template);
00122   length = g_mapped_file_get_length (template);
00123 #if DEBUG_OPTIMIZE
00124   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125            content);
00126 #endif
00127   file = g_fopen (input, "w");
00128
00129   // Parsing template
00130   for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132 #if DEBUG_OPTIMIZE
00133       fprintf (stderr, "optimize_input: variable=%u\n", i);
00134 #endif
00135       snprintf (buffer, 32, "@variable%u@", i + 1);
00136       regex = g_regex_new (buffer, 0, 0, NULL);
00137       if (i == 0)
00138         {
00139           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                              optimize->label[i], 0, NULL);
00141 #if DEBUG_OPTIMIZE
00142           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143 #endif
00144         }
00145       else
00146         {
00147           length = strlen (buffer3);
00148           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                              optimize->label[i], 0, NULL);
00150           g_free (buffer3);
00151         }
00152       g_regex_unref (regex);
00153       length = strlen (buffer2);
00154       snprintf (buffer, 32, "@value%u@", i + 1);
00155       regex = g_regex_new (buffer, 0, 0, NULL);
00156       snprintf (value, 32, format[optimize->precision[i]],
00157                 optimize->value[simulation * optimize->nvariables + i]);
00158
00159 #if DEBUG_OPTIMIZE
00160       fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                          0, NULL);
00164       g_free (buffer2);
00165       g_regex_unref (regex);
00166     }
00167
00168   // Saving input file
```

```
00169   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170   g_free (buffer3);
00171   fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175   fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177   return;
00178 }
00179
00190 double
00191 optimize_parse (unsigned int simulation, unsigned int experiment)
00192 {
00193   unsigned int i;
00194   double e;
00195   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196     *buffer3, *buffer4;
00197   FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200   fprintf (stderr, "optimize_parse: start\n");
00201   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202            simulation, experiment);
00203 #endif
00204
00205   // Opening input files
00206   for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210 #if DEBUG_OPTIMIZE
00211       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
00213       optimize_input (simulation, &input[i][0],
00214                       optimize->file[i][experiment]);
00215     }
00216   for (; i < MAX_NINPUTS; ++i)
00217     strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219   fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222   // Performing the simulation
00223   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224   buffer2 = g_path_get_dirname (optimize->simulator);
00225   buffer3 = g_path_get_basename (optimize->simulator);
00226   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230   g_free (buffer4);
00231   g_free (buffer3);
00232   g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234   fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236   system (buffer);
00237
00238   // Checking the objective value function
00239   if (optimize->evaluator)
00240     {
00241       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242       buffer2 = g_path_get_dirname (optimize->evaluator);
00243       buffer3 = g_path_get_basename (optimize->evaluator);
00244       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245       snprintf (buffer, 512, "\"%s\" %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247       g_free (buffer4);
00248       g_free (buffer3);
00249       g_free (buffer2);
00250 #if DEBUG_OPTIMIZE
00251       fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253       system (buffer);
00254       file_result = g_fopen (result, "r");
00255       e = atof (fgets (buffer, 512, file_result));
00256       fclose (file_result);
00257     }
00258   else
00259     {
00260       strcpy (result, "");
00261       file_result = g_fopen (output, "r");
00262       e = atof (fgets (buffer, 512, file_result));
00263       fclose (file_result);
00264     }
00265
```

```
00266   // Removing files
00267 #if !DEBUG_OPTIMIZE
00268   for (i = 0; i < optimize->ninputs; ++i)
00269     {
00270       if (optimize->file[i][0])
00271         {
00272           snprintf (buffer, 512, RM " %s", &input[i][0]);
00273           system (buffer);
00274         }
00275     }
00276   snprintf (buffer, 512, RM " %s %s", output, result);
00277   system (buffer);
00278 #endif
00279
00280   // Processing pending events
00281   if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285   fprintf (stderr, "optimize_parse: end\n");
00286 #endif
00287
00288   // Returning the objective function
00289   return e * optimize->weight[experiment];
00290 }
00291
00299 double
00300 optimize_norm_euclidian (unsigned int simulation)
00301 {
00302   double e, ei;
00303   unsigned int i;
00304 #if DEBUG_OPTIMIZE
00305   fprintf (stderr, "optimize_norm_euclidian: start\n");
00306 #endif
00307   e = 0.;
00308   for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310       ei = optimize_parse (simulation, i);
00311       e += ei * ei;
00312     }
00313   e = sqrt (e);
00314 #if DEBUG_OPTIMIZE
00315   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316   fprintf (stderr, "optimize_norm_euclidian: end\n");
00317 #endif
00318   return e;
00319 }
00320
00328 double
00329 optimize_norm_maximum (unsigned int simulation)
00330 {
00331   double e, ei;
00332   unsigned int i;
00333 #if DEBUG_OPTIMIZE
00334   fprintf (stderr, "optimize_norm_maximum: start\n");
00335 #endif
00336   e = 0.;
00337   for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339       ei = fabs (optimize_parse (simulation, i));
00340       e = fmax (e, ei);
00341     }
00342 #if DEBUG_OPTIMIZE
00343   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344   fprintf (stderr, "optimize_norm_maximum: end\n");
00345 #endif
00346   return e;
00347 }
00348
00356 double
00357 optimize_norm_p (unsigned int simulation)
00358 {
00359   double e, ei;
00360   unsigned int i;
00361 #if DEBUG_OPTIMIZE
00362   fprintf (stderr, "optimize_norm_p: start\n");
00363 #endif
00364   e = 0.;
00365   for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367       ei = fabs (optimize_parse (simulation, i));
00368       e += pow (ei, optimize->p);
00369     }
00370   e = pow (e, 1. / optimize->p);
00371 #if DEBUG_OPTIMIZE
00372   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373   fprintf (stderr, "optimize_norm_p: end\n");
```

```
00374 #endif
00375   return e;
00376 }
00377
00385 double
00386 optimize_norm_taxicab (unsigned int simulation)
00387 {
00388   double e;
00389   unsigned int i;
00390 #if DEBUG_OPTIMIZE
00391   fprintf (stderr, "optimize_norm_taxicab: start\n");
00392 #endif
00393   e = 0.;
00394   for (i = 0; i < optimize->nexperiments; ++i)
00395     e += fabs (optimize_parse (simulation, i));
00396 #if DEBUG_OPTIMIZE
00397   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398   fprintf (stderr, "optimize_norm_taxicab: end\n");
00399 #endif
00400   return e;
00401 }
00402
00407 void
00408 optimize_print ()
00409 {
00410   unsigned int i;
00411   char buffer[512];
00412 #if HAVE_MPI
00413   if (optimize->mpi_rank)
00414     return;
00415 #endif
00416   printf ("%s\n", _("Best result"));
00417   fprintf (optimize->file_result, "%s\n", _("Best result"));
00418   printf ("error = %.15le\n", optimize->error_old[0]);
00419   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
00420   for (i = 0; i < optimize->nvariables; ++i)
00421     {
00422       snprintf (buffer, 512, "%s = %s\n",
00423                 optimize->label[i], format[optimize->precision[i]]);
00424       printf (buffer, optimize->value_old[i]);
00425       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00426     }
00427   fflush (optimize->file_result);
00428 }
00429
00438 void
00439 optimize_save_variables (unsigned int simulation, double error)
00440 {
00441   unsigned int i;
00442   char buffer[64];
00443 #if DEBUG_OPTIMIZE
00444   fprintf (stderr, "optimize_save_variables: start\n");
00445 #endif
00446   for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449       fprintf (optimize->file_variables, buffer,
00450                optimize->value[simulation * optimize->nvariables + i]);
00451     }
00452   fprintf (optimize->file_variables, "%.14le\n", error);
00453   fflush (optimize->file_variables);
00454 #if DEBUG_OPTIMIZE
00455   fprintf (stderr, "optimize_save_variables: end\n");
00456 #endif
00457 }
00458
00467 void
00468 optimize_best (unsigned int simulation, double value)
00469 {
00470   unsigned int i, j;
00471   double e;
00472 #if DEBUG_OPTIMIZE
00473   fprintf (stderr, "optimize_best: start\n");
00474   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475            optimize->nsaveds, optimize->nbest);
00476 #endif
00477   if (optimize->nsaveds < optimize->nbest
00478       || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480       if (optimize->nsaveds < optimize->nbest)
00481         ++optimize->nsaveds;
00482       optimize->error_best[optimize->nsaveds - 1] = value;
00483       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484       for (i = optimize->nsaveds; --i;)
00485         {
00486           if (optimize->error_best[i] < optimize->error_best[i - 1])
```

```
00487             {
00488               j = optimize->simulation_best[i];
00489               e = optimize->error_best[i];
00490               optimize->simulation_best[i] = optimize->
    simulation_best[i - 1];
00491               optimize->error_best[i] = optimize->error_best[i - 1];
00492               optimize->simulation_best[i - 1] = j;
00493               optimize->error_best[i - 1] = e;
00494             }
00495           else
00496             break;
00497         }
00498     }
00499 #if DEBUG_OPTIMIZE
00500   fprintf (stderr, "optimize_best: end\n");
00501 #endif
00502 }
00503
00508 void
00509 optimize_sequential ()
00510 {
00511   unsigned int i;
00512   double e;
00513 #if DEBUG_OPTIMIZE
00514   fprintf (stderr, "optimize_sequential: start\n");
00515   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00516            optimize->nstart, optimize->nend);
00517 #endif
00518   for (i = optimize->nstart; i < optimize->nend; ++i)
00519     {
00520       e = optimize_norm (i);
00521       optimize_best (i, e);
00522       optimize_save_variables (i, e);
00523       if (e < optimize->threshold)
00524         {
00525           optimize->stop = 1;
00526           break;
00527         }
00528 #if DEBUG_OPTIMIZE
00529       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00530 #endif
00531     }
00532 #if DEBUG_OPTIMIZE
00533   fprintf (stderr, "optimize_sequential: end\n");
00534 #endif
00535 }
00536
00544 void *
00545 optimize_thread (ParallelData * data)
00546 {
00547   unsigned int i, thread;
00548   double e;
00549 #if DEBUG_OPTIMIZE
00550   fprintf (stderr, "optimize_thread: start\n");
00551 #endif
00552   thread = data->thread;
00553 #if DEBUG_OPTIMIZE
00554   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555            optimize->thread[thread], optimize->thread[thread + 1]);
00556 #endif
00557   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559       e = optimize_norm (i);
00560       g_mutex_lock (mutex);
00561       optimize_best (i, e);
00562       optimize_save_variables (i, e);
00563       if (e < optimize->threshold)
00564         optimize->stop = 1;
00565       g_mutex_unlock (mutex);
00566       if (optimize->stop)
00567         break;
00568 #if DEBUG_OPTIMIZE
00569       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570 #endif
00571     }
00572 #if DEBUG_OPTIMIZE
00573   fprintf (stderr, "optimize_thread: end\n");
00574 #endif
00575   g_thread_exit (NULL);
00576   return NULL;
00577 }
00578
00590 void
00591 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00592                 double *error_best)
00593 {
00594   unsigned int i, j, k, s[optimize->nbest];
```

```
00595   double e[optimize->nbest];
00596 #if DEBUG_OPTIMIZE
00597   fprintf (stderr, "optimize_merge: start\n");
00598 #endif
00599   i = j = k = 0;
00600   do
00601     {
00602       if (i == optimize->nsaveds)
00603         {
00604           s[k] = simulation_best[j];
00605           e[k] = error_best[j];
00606           ++j;
00607           ++k;
00608           if (j == nsaveds)
00609             break;
00610         }
00611       else if (j == nsaveds)
00612         {
00613           s[k] = optimize->simulation_best[i];
00614           e[k] = optimize->error_best[i];
00615           ++i;
00616           ++k;
00617           if (i == optimize->nsaveds)
00618             break;
00619         }
00620       else if (optimize->error_best[i] > error_best[j])
00621         {
00622           s[k] = simulation_best[j];
00623           e[k] = error_best[j];
00624           ++j;
00625           ++k;
00626         }
00627       else
00628         {
00629           s[k] = optimize->simulation_best[i];
00630           e[k] = optimize->error_best[i];
00631           ++i;
00632           ++k;
00633         }
00634     }
00635   while (k < optimize->nbest);
00636   optimize->nsaveds = k;
00637   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638   memcpy (optimize->error_best, e, k * sizeof (double));
00639 #if DEBUG_OPTIMIZE
00640   fprintf (stderr, "optimize_merge: end\n");
00641 #endif
00642 }
00643
00648 #if HAVE_MPI
00649 void
00650 optimize_synchronise ()
00651 {
00652   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00653   double error_best[optimize->nbest];
00654   MPI_Status mpi_stat;
00655 #if DEBUG_OPTIMIZE
00656   fprintf (stderr, "optimize_synchronise: start\n");
00657 #endif
00658   if (optimize->mpi_rank == 0)
00659     {
00660       for (i = 1; i < ntasks; ++i)
00661         {
00662           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00663           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00664                     MPI_COMM_WORLD, &mpi_stat);
00665           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00666                     MPI_COMM_WORLD, &mpi_stat);
00667           optimize_merge (nsaveds, simulation_best, error_best);
00668           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00669           if (stop)
00670             optimize->stop = 1;
00671         }
00672       for (i = 1; i < ntasks; ++i)
00673         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00674     }
00675   else
00676     {
00677       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00678       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00679                 MPI_COMM_WORLD);
00680       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00681                 MPI_COMM_WORLD);
00682       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00683       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00684       if (stop)
00685         optimize->stop = 1;
```

```
00686     }
00687 #if DEBUG_OPTIMIZE
00688   fprintf (stderr, "optimize_synchronise: end\n");
00689 #endif
00690 }
00691 #endif
00692
00697 void
00698 optimize_sweep ()
00699 {
00700   unsigned int i, j, k, l;
00701   double e;
00702   GThread *thread[nthreads];
00703   ParallelData data[nthreads];
00704 #if DEBUG_OPTIMIZE
00705   fprintf (stderr, "optimize_sweep: start\n");
00706 #endif
00707   for (i = 0; i < optimize->nsimulations; ++i)
00708     {
00709       k = i;
00710       for (j = 0; j < optimize->nvariables; ++j)
00711         {
00712           l = k % optimize->nsweeps[j];
00713           k /= optimize->nsweeps[j];
00714           e = optimize->rangemin[j];
00715           if (optimize->nsweeps[j] > 1)
00716             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00717               / (optimize->nsweeps[j] - 1);
00718           optimize->value[i * optimize->nvariables + j] = e;
00719         }
00720     }
00721   optimize->nsaveds = 0;
00722   if (nthreads <= 1)
00723     optimize_sequential ();
00724   else
00725     {
00726       for (i = 0; i < nthreads; ++i)
00727         {
00728           data[i].thread = i;
00729           thread[i] =
00730             g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00731         }
00732       for (i = 0; i < nthreads; ++i)
00733         g_thread_join (thread[i]);
00734     }
00735 #if HAVE_MPI
00736   // Communicating tasks results
00737   optimize_synchronise ();
00738 #endif
00739 #if DEBUG_OPTIMIZE
00740   fprintf (stderr, "optimize_sweep: end\n");
00741 #endif
00742 }
00743
00748 void
00749 optimize_MonteCarlo ()
00750 {
00751   unsigned int i, j;
00752   GThread *thread[nthreads];
00753   ParallelData data[nthreads];
00754 #if DEBUG_OPTIMIZE
00755   fprintf (stderr, "optimize_MonteCarlo: start\n");
00756 #endif
00757   for (i = 0; i < optimize->nsimulations; ++i)
00758     for (j = 0; j < optimize->nvariables; ++j)
00759       optimize->value[i * optimize->nvariables + j]
00760         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00761         * (optimize->rangemax[j] - optimize->rangemin[j]);
00762   optimize->nsaveds = 0;
00763   if (nthreads <= 1)
00764     optimize_sequential ();
00765   else
00766     {
00767       for (i = 0; i < nthreads; ++i)
00768         {
00769           data[i].thread = i;
00770           thread[i] =
00771             g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00772         }
00773       for (i = 0; i < nthreads; ++i)
00774         g_thread_join (thread[i]);
00775     }
00776 #if HAVE_MPI
00777   // Communicating tasks results
00778   optimize_synchronise ();
00779 #endif
00780 #if DEBUG_OPTIMIZE
```

```
00781   fprintf (stderr, "optimize_MonteCarlo: end\n");
00782 #endif
00783 }
00784
00794 void
00795 optimize_best_direction (unsigned int simulation, double value)
00796 {
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_best_direction: start\n");
00799   fprintf (stderr,
00800            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801            simulation, value, optimize->error_best[0]);
00802 #endif
00803   if (value < optimize->error_best[0])
00804     {
00805       optimize->error_best[0] = value;
00806       optimize->simulation_best[0] = simulation;
00807 #if DEBUG_OPTIMIZE
00808       fprintf (stderr,
00809                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810                simulation, value);
00811 #endif
00812     }
00813 #if DEBUG_OPTIMIZE
00814   fprintf (stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }
00817
00824 void
00825 optimize_direction_sequential (unsigned int simulation)
00826 {
00827   unsigned int i, j;
00828   double e;
00829 #if DEBUG_OPTIMIZE
00830   fprintf (stderr, "optimize_direction_sequential: start\n");
00831   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832            "nend_direction=%u\n",
00833            optimize->nstart_direction, optimize->nend_direction);
00834 #endif
00835   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837       j = simulation + i;
00838       e = optimize_norm (j);
00839       optimize_best_direction (j, e);
00840       optimize_save_variables (j, e);
00841       if (e < optimize->threshold)
00842         {
00843           optimize->stop = 1;
00844           break;
00845         }
00846 #if DEBUG_OPTIMIZE
00847       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00848 #endif
00849     }
00850 #if DEBUG_OPTIMIZE
00851   fprintf (stderr, "optimize_direction_sequential: end\n");
00852 #endif
00853 }
00854
00862 void *
00863 optimize_direction_thread (ParallelData * data)
00864 {
00865   unsigned int i, thread;
00866   double e;
00867 #if DEBUG_OPTIMIZE
00868   fprintf (stderr, "optimize_direction_thread: start\n");
00869 #endif
00870   thread = data->thread;
00871 #if DEBUG_OPTIMIZE
00872   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873            thread,
00874            optimize->thread_direction[thread],
00875            optimize->thread_direction[thread + 1]);
00876 #endif
00877   for (i = optimize->thread_direction[thread];
00878        i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880       e = optimize_norm (i);
00881       g_mutex_lock (mutex);
00882       optimize_best_direction (i, e);
00883       optimize_save_variables (i, e);
00884       if (e < optimize->threshold)
00885         optimize->stop = 1;
00886       g_mutex_unlock (mutex);
00887       if (optimize->stop)
00888         break;
00889 #if DEBUG_OPTIMIZE
```

```
00890       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00891 #endif
00892     }
00893 #if DEBUG_OPTIMIZE
00894   fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896   g_thread_exit (NULL);
00897   return NULL;
00898 }
00899
00909 double
00910 optimize_estimate_direction_random (unsigned int variable,
00911                                     unsigned int estimate)
00912 {
00913   double x;
00914 #if DEBUG_OPTIMIZE
00915   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916 #endif
00917   x = optimize->direction[variable]
00918     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00919 #if DEBUG_OPTIMIZE
00920   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00921            variable, x);
00922   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00923 #endif
00924   return x;
00925 }
00926
00936 double
00937 optimize_estimate_direction_coordinates (unsigned int variable,
00938                                          unsigned int estimate)
00939 {
00940   double x;
00941 #if DEBUG_OPTIMIZE
00942   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00943 #endif
00944   x = optimize->direction[variable];
00945   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947       if (estimate & 1)
00948         x += optimize->step[variable];
00949       else
00950         x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953   fprintf (stderr,
00954            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00955            variable, x);
00956   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00957 #endif
00958   return x;
00959 }
00960
00967 void
00968 optimize_step_direction (unsigned int simulation)
00969 {
00970   GThread *thread[nthreads_direction];
00971   ParallelData data[nthreads_direction];
00972   unsigned int i, j, k, b;
00973 #if DEBUG_OPTIMIZE
00974   fprintf (stderr, "optimize_step_direction: start\n");
00975 #endif
00976   for (i = 0; i < optimize->nestimates; ++i)
00977     {
00978       k = (simulation + i) * optimize->nvariables;
00979       b = optimize->simulation_best[0] * optimize->nvariables;
00980 #if DEBUG_OPTIMIZE
00981       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00982                simulation + i, optimize->simulation_best[0]);
00983 #endif
00984       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00985         {
00986 #if DEBUG_OPTIMIZE
00987           fprintf (stderr,
00988                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989                    i, j, optimize->value[b]);
00990 #endif
00991           optimize->value[k]
00992             = optimize->value[b] + optimize_estimate_direction (j, i);
00993           optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                            optimize->rangeminabs[j]),
00995                                      optimize->rangemaxabs[j]);
00996 #if DEBUG_OPTIMIZE
00997           fprintf (stderr,
00998                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999                    i, j, optimize->value[k]);
01000 #endif
```

```
01001         }
01002     }
01003   if (nthreads_direction == 1)
01004     optimize_direction_sequential (simulation);
01005   else
01006     {
01007       for (i = 0; i <= nthreads_direction; ++i)
01008         {
01009           optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01012             / nthreads_direction;
01013 #if DEBUG_OPTIMIZE
01014           fprintf (stderr,
01015                    "optimize_step_direction: i=%u thread_direction=%u\n",
01016                    i, optimize->thread_direction[i]);
01017 #endif
01018         }
01019       for (i = 0; i < nthreads_direction; ++i)
01020         {
01021           data[i].thread = i;
01022           thread[i] = g_thread_new
01023             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01024         }
01025       for (i = 0; i < nthreads_direction; ++i)
01026         g_thread_join (thread[i]);
01027     }
01028 #if DEBUG_OPTIMIZE
01029   fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031 }
01032
01037 void
01038 optimize_direction ()
01039 {
01040   unsigned int i, j, k, b, s, adjust;
01041 #if DEBUG_OPTIMIZE
01042   fprintf (stderr, "optimize_direction: start\n");
01043 #endif
01044   for (i = 0; i < optimize->nvariables; ++i)
01045     optimize->direction[i] = 0.;
01046   b = optimize->simulation_best[0] * optimize->nvariables;
01047   s = optimize->nsimulations;
01048   adjust = 1;
01049   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01050     {
01051 #if DEBUG_OPTIMIZE
01052       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01053                i, optimize->simulation_best[0]);
01054 #endif
01055       optimize_step_direction (s);
01056       k = optimize->simulation_best[0] * optimize->nvariables;
01057 #if DEBUG_OPTIMIZE
01058       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01059                i, optimize->simulation_best[0]);
01060 #endif
01061       if (k == b)
01062         {
01063           if (adjust)
01064             for (j = 0; j < optimize->nvariables; ++j)
01065               optimize->step[j] *= 0.5;
01066           for (j = 0; j < optimize->nvariables; ++j)
01067             optimize->direction[j] = 0.;
01068           adjust = 1;
01069         }
01070       else
01071         {
01072           for (j = 0; j < optimize->nvariables; ++j)
01073             {
01074 #if DEBUG_OPTIMIZE
01075               fprintf (stderr,
01076                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01077                        j, optimize->value[k + j], j, optimize->value[b + j]);
01078 #endif
01079               optimize->direction[j]
01080                 = (1. - optimize->relaxation) * optimize->direction[j]
01081                 + optimize->relaxation
01082                 * (optimize->value[k + j] - optimize->value[b + j]);
01083 #if DEBUG_OPTIMIZE
01084               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01085                        j, optimize->direction[j]);
01086 #endif
01087             }
01088           adjust = 0;
01089         }
01090     }
```

```
01091 #if DEBUG_OPTIMIZE
01092   fprintf (stderr, "optimize_direction: end\n");
01093 #endif
01094 }
01095
01103 double
01104 optimize_genetic_objective (Entity * entity)
01105 {
01106   unsigned int j;
01107   double objective;
01108   char buffer[64];
01109 #if DEBUG_OPTIMIZE
01110   fprintf (stderr, "optimize_genetic_objective: start\n");
01111 #endif
01112   for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114       optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117   objective = optimize_norm (entity->id);
01118   g_mutex_lock (mutex);
01119   for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122       fprintf (optimize->file_variables, buffer,
01123                genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125   fprintf (optimize->file_variables, "%.14le\n", objective);
01126   g_mutex_unlock (mutex);
01127 #if DEBUG_OPTIMIZE
01128   fprintf (stderr, "optimize_genetic_objective: end\n");
01129 #endif
01130   return objective;
01131 }
01132
01137 void
01138 optimize_genetic ()
01139 {
01140   char *best_genome;
01141   double best_objective, *best_variable;
01142 #if DEBUG_OPTIMIZE
01143   fprintf (stderr, "optimize_genetic: start\n");
01144   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01145            nthreads);
01146   fprintf (stderr,
01147            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01148            optimize->nvariables, optimize->nsimulations,
01149            optimize->niterations);
01150   fprintf (stderr,
01151            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01152            optimize->mutation_ratio, optimize->reproduction_ratio,
01153            optimize->adaptation_ratio);
01154 #endif
01155   genetic_algorithm_default (optimize->nvariables,
01156                              optimize->genetic_variable,
01157                              optimize->nsimulations,
01158                              optimize->niterations,
01159                              optimize->mutation_ratio,
01160                              optimize->reproduction_ratio,
01161                              optimize->adaptation_ratio,
01162                              optimize->seed,
01163                              optimize->threshold,
01164                              &optimize_genetic_objective,
01165                              &best_genome, &best_variable, &best_objective);
01166 #if DEBUG_OPTIMIZE
01167   fprintf (stderr, "optimize_genetic: the best\n");
01168 #endif
01169   optimize->error_old = (double *) g_malloc (sizeof (double));
01170   optimize->value_old
01171     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01172   optimize->error_old[0] = best_objective;
01173   memcpy (optimize->value_old, best_variable,
01174           optimize->nvariables * sizeof (double));
01175   g_free (best_genome);
01176   g_free (best_variable);
01177   optimize_print ();
01178 #if DEBUG_OPTIMIZE
01179   fprintf (stderr, "optimize_genetic: end\n");
01180 #endif
01181 }
01182
01187 void
01188 optimize_save_old ()
01189 {
01190   unsigned int i, j;
01191 #if DEBUG_OPTIMIZE
01192   fprintf (stderr, "optimize_save_old: start\n");
```

```
01193   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01194 #endif
01195   memcpy (optimize->error_old, optimize->error_best,
01196           optimize->nbest * sizeof (double));
01197   for (i = 0; i < optimize->nbest; ++i)
01198     {
01199       j = optimize->simulation_best[i];
01200 #if DEBUG_OPTIMIZE
01201       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01202 #endif
01203       memcpy (optimize->value_old + i * optimize->nvariables,
01204               optimize->value + j * optimize->nvariables,
01205               optimize->nvariables * sizeof (double));
01206     }
01207 #if DEBUG_OPTIMIZE
01208   for (i = 0; i < optimize->nvariables; ++i)
01209     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01210              i, optimize->value_old[i]);
01211   fprintf (stderr, "optimize_save_old: end\n");
01212 #endif
01213 }
01214
01220 void
01221 optimize_merge_old ()
01222 {
01223   unsigned int i, j, k;
01224   double v[optimize->nbest * optimize->nvariables], e[optimize->
    nbest],
01225     *enew, *eold;
01226 #if DEBUG_OPTIMIZE
01227   fprintf (stderr, "optimize_merge_old: start\n");
01228 #endif
01229   enew = optimize->error_best;
01230   eold = optimize->error_old;
01231   i = j = k = 0;
01232   do
01233     {
01234       if (*enew < *eold)
01235         {
01236           memcpy (v + k * optimize->nvariables,
01237                   optimize->value
01238                   + optimize->simulation_best[i] * optimize->
    nvariables,
01239                   optimize->nvariables * sizeof (double));
01240           e[k] = *enew;
01241           ++k;
01242           ++enew;
01243           ++i;
01244         }
01245       else
01246         {
01247           memcpy (v + k * optimize->nvariables,
01248                   optimize->value_old + j * optimize->nvariables,
01249                   optimize->nvariables * sizeof (double));
01250           e[k] = *eold;
01251           ++k;
01252           ++eold;
01253           ++j;
01254         }
01255     }
01256   while (k < optimize->nbest);
01257   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01258   memcpy (optimize->error_old, e, k * sizeof (double));
01259 #if DEBUG_OPTIMIZE
01260   fprintf (stderr, "optimize_merge_old: end\n");
01261 #endif
01262 }
01263
01269 void
01270 optimize_refine ()
01271 {
01272   unsigned int i, j;
01273   double d;
01274 #if HAVE_MPI
01275   MPI_Status mpi_stat;
01276 #endif
01277 #if DEBUG_OPTIMIZE
01278   fprintf (stderr, "optimize_refine: start\n");
01279 #endif
01280 #if HAVE_MPI
01281   if (!optimize->mpi_rank)
01282     {
01283 #endif
01284       for (j = 0; j < optimize->nvariables; ++j)
01285         {
01286           optimize->rangemin[j] = optimize->rangemax[j]
01287             = optimize->value_old[j];
```

```
01288         }
01289       for (i = 0; ++i < optimize->nbest;)
01290         {
01291           for (j = 0; j < optimize->nvariables; ++j)
01292             {
01293               optimize->rangemin[j]
01294                 = fmin (optimize->rangemin[j],
01295                         optimize->value_old[i * optimize->nvariables + j]);
01296               optimize->rangemax[j]
01297                 = fmax (optimize->rangemax[j],
01298                         optimize->value_old[i * optimize->nvariables + j]);
01299             }
01300         }
01301       for (j = 0; j < optimize->nvariables; ++j)
01302         {
01303           d = optimize->tolerance
01304             * (optimize->rangemax[j] - optimize->rangemin[j]);
01305           switch (optimize->algorithm)
01306             {
01307             case ALGORITHM_MONTE_CARLO:
01308               d *= 0.5;
01309               break;
01310             default:
01311               if (optimize->nsweeps[j] > 1)
01312                 d /= optimize->nsweeps[j] - 1;
01313               else
01314                 d = 0.;
01315             }
01316           optimize->rangemin[j] -= d;
01317           optimize->rangemin[j]
01318             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01319           optimize->rangemax[j] += d;
01320           optimize->rangemax[j]
01321             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01322           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01323                   optimize->rangemin[j], optimize->rangemax[j]);
01324           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01325                    optimize->label[j], optimize->rangemin[j],
01326                    optimize->rangemax[j]);
01327         }
01328 #if HAVE_MPI
01329       for (i = 1; i < ntasks; ++i)
01330         {
01331           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01332                     1, MPI_COMM_WORLD);
01333           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01334                     1, MPI_COMM_WORLD);
01335         }
01336     }
01337   else
01338     {
01339       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01340                 MPI_COMM_WORLD, &mpi_stat);
01341       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01342                 MPI_COMM_WORLD, &mpi_stat);
01343     }
01344 #endif
01345 #if DEBUG_OPTIMIZE
01346   fprintf (stderr, "optimize_refine: end\n");
01347 #endif
01348 }
01349
01354 void
01355 optimize_step ()
01356 {
01357 #if DEBUG_OPTIMIZE
01358   fprintf (stderr, "optimize_step: start\n");
01359 #endif
01360   optimize_algorithm ();
01361   if (optimize->nsteps)
01362     optimize_direction ();
01363 #if DEBUG_OPTIMIZE
01364   fprintf (stderr, "optimize_step: end\n");
01365 #endif
01366 }
01367
01372 void
01373 optimize_iterate ()
01374 {
01375   unsigned int i;
01376 #if DEBUG_OPTIMIZE
01377   fprintf (stderr, "optimize_iterate: start\n");
01378 #endif
01379   optimize->error_old =
01380     (double *) g_malloc (optimize->nbest * sizeof (double));
01381   optimize->value_old =
01382     (double *) g_malloc (optimize->nbest * optimize->nvariables *
```

```
01383                           sizeof (double));
01384   optimize_step ();
01385   optimize_save_old ();
01386   optimize_refine ();
01387   optimize_print ();
01388   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01389     {
01390       optimize_step ();
01391       optimize_merge_old ();
01392       optimize_refine ();
01393       optimize_print ();
01394     }
01395 #if DEBUG_OPTIMIZE
01396   fprintf (stderr, "optimize_iterate: end\n");
01397 #endif
01398 }
01399
01404 void
01405 optimize_free ()
01406 {
01407   unsigned int i, j;
01408 #if DEBUG_OPTIMIZE
01409   fprintf (stderr, "optimize_free: start\n");
01410 #endif
01411   for (j = 0; j < optimize->ninputs; ++j)
01412     {
01413       for (i = 0; i < optimize->nexperiments; ++i)
01414         g_mapped_file_unref (optimize->file[j][i]);
01415       g_free (optimize->file[j]);
01416     }
01417   g_free (optimize->error_old);
01418   g_free (optimize->value_old);
01419   g_free (optimize->value);
01420   g_free (optimize->genetic_variable);
01421 #if DEBUG_OPTIMIZE
01422   fprintf (stderr, "optimize_free: end\n");
01423 #endif
01424 }
01425
01430 void
01431 optimize_open ()
01432 {
01433   GTimeZone *tz;
01434   GDateTime *t0, *t;
01435   unsigned int i, j;
01436
01437 #if DEBUG_OPTIMIZE
01438   char *buffer;
01439   fprintf (stderr, "optimize_open: start\n");
01440 #endif
01441
01442   // Getting initial time
01443 #if DEBUG_OPTIMIZE
01444   fprintf (stderr, "optimize_open: getting initial time\n");
01445 #endif
01446   tz = g_time_zone_new_utc ();
01447   t0 = g_date_time_new_now (tz);
01448
01449   // Obtaining and initing the pseudo-random numbers generator seed
01450 #if DEBUG_OPTIMIZE
01451   fprintf (stderr, "optimize_open: getting initial seed\n");
01452 #endif
01453   if (optimize->seed == DEFAULT_RANDOM_SEED)
01454     optimize->seed = input->seed;
01455   gsl_rng_set (optimize->rng, optimize->seed);
01456
01457   // Replacing the working directory
01458 #if DEBUG_OPTIMIZE
01459   fprintf (stderr, "optimize_open: replacing the working directory\n");
01460 #endif
01461   g_chdir (input->directory);
01462
01463   // Getting results file names
01464   optimize->result = input->result;
01465   optimize->variables = input->variables;
01466
01467   // Obtaining the simulator file
01468   optimize->simulator = input->simulator;
01469
01470   // Obtaining the evaluator file
01471   optimize->evaluator = input->evaluator;
01472
01473   // Reading the algorithm
01474   optimize->algorithm = input->algorithm;
01475   switch (optimize->algorithm)
01476     {
01477     case ALGORITHM_MONTE_CARLO:
```

```
01478       optimize_algorithm = optimize_MonteCarlo;
01479       break;
01480     case ALGORITHM_SWEEP:
01481       optimize_algorithm = optimize_sweep;
01482       break;
01483     default:
01484       optimize_algorithm = optimize_genetic;
01485       optimize->mutation_ratio = input->mutation_ratio;
01486       optimize->reproduction_ratio = input->
      reproduction_ratio;
01487       optimize->adaptation_ratio = input->adaptation_ratio;
01488     }
01489   optimize->nvariables = input->nvariables;
01490   optimize->nsimulations = input->nsimulations;
01491   optimize->niterations = input->niterations;
01492   optimize->nbest = input->nbest;
01493   optimize->tolerance = input->tolerance;
01494   optimize->nsteps = input->nsteps;
01495   optimize->nestimates = 0;
01496   optimize->threshold = input->threshold;
01497   optimize->stop = 0;
01498   if (input->nsteps)
01499     {
01500       optimize->relaxation = input->relaxation;
01501       switch (input->direction)
01502         {
01503         case DIRECTION_METHOD_COORDINATES:
01504           optimize->nestimates = 2 * optimize->nvariables;
01505           optimize_estimate_direction =
01506             optimize_estimate_direction_coordinates;
01507           break;
01508         default:
01509           optimize->nestimates = input->nestimates;
01510           optimize_estimate_direction =
      optimize_estimate_direction_random;
01511         }
01512     }
01513
01514 #if DEBUG_OPTIMIZE
01515   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01516 #endif
01517   optimize->simulation_best
01518     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01519   optimize->error_best =
01520     (double *) alloca (optimize->nbest * sizeof (double));
01521
01522   // Reading the experimental data
01523 #if DEBUG_OPTIMIZE
01524   buffer = g_get_current_dir ();
01525   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01526   g_free (buffer);
01527 #endif
01528   optimize->nexperiments = input->nexperiments;
01529   optimize->ninputs = input->experiment->ninputs;
01530   optimize->experiment
01531     = (char **) alloca (input->nexperiments * sizeof (char *));
01532   optimize->weight =
01533     (double *) alloca (input->nexperiments * sizeof (double));
01534   for (i = 0; i < input->experiment->ninputs; ++i)
01535     optimize->file[i] = (GMappedFile **)
01536       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01537   for (i = 0; i < input->nexperiments; ++i)
01538     {
01539 #if DEBUG_OPTIMIZE
01540       fprintf (stderr, "optimize_open: i=%u\n", i);
01541 #endif
01542       optimize->experiment[i] = input->experiment[i].
      name;
01543       optimize->weight[i] = input->experiment[i].weight;
01544 #if DEBUG_OPTIMIZE
01545       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01546                optimize->experiment[i], optimize->weight[i]);
01547 #endif
01548       for (j = 0; j < input->experiment->ninputs; ++j)
01549         {
01550 #if DEBUG_OPTIMIZE
01551           fprintf (stderr, "optimize_open: template%u\n", j + 1);
01552 #endif
01553           optimize->file[j][i]
01554             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01555         }
01556     }
01557
01558   // Reading the variables data
01559 #if DEBUG_OPTIMIZE
01560   fprintf (stderr, "optimize_open: reading variables\n");
01561 #endif
```

```
01562   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01563   j = input->nvariables * sizeof (double);
01564   optimize->rangemin = (double *) alloca (j);
01565   optimize->rangeminabs = (double *) alloca (j);
01566   optimize->rangemax = (double *) alloca (j);
01567   optimize->rangemaxabs = (double *) alloca (j);
01568   optimize->step = (double *) alloca (j);
01569   j = input->nvariables * sizeof (unsigned int);
01570   optimize->precision = (unsigned int *) alloca (j);
01571   optimize->nsweeps = (unsigned int *) alloca (j);
01572   optimize->nbits = (unsigned int *) alloca (j);
01573   for (i = 0; i < input->nvariables; ++i)
01574     {
01575       optimize->label[i] = input->variable[i].name;
01576       optimize->rangemin[i] = input->variable[i].rangemin;
01577       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01578       optimize->rangemax[i] = input->variable[i].rangemax;
01579       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01580       optimize->precision[i] = input->variable[i].
      precision;
01581       optimize->step[i] = input->variable[i].step;
01582       optimize->nsweeps[i] = input->variable[i].nsweeps;
01583       optimize->nbits[i] = input->variable[i].nbits;
01584     }
01585   if (input->algorithm == ALGORITHM_SWEEP)
01586     {
01587       optimize->nsimulations = 1;
01588       for (i = 0; i < input->nvariables; ++i)
01589         {
01590           if (input->algorithm == ALGORITHM_SWEEP)
01591             {
01592               optimize->nsimulations *= optimize->nsweeps[i];
01593 #if DEBUG_OPTIMIZE
01594               fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01595                        optimize->nsweeps[i], optimize->nsimulations);
01596 #endif
01597             }
01598         }
01599     }
01600   if (optimize->nsteps)
01601     optimize->direction
01602       = (double *) alloca (optimize->nvariables * sizeof (double));
01603
01604   // Setting error norm
01605   switch (input->norm)
01606     {
01607     case ERROR_NORM_EUCLIDIAN:
01608       optimize_norm = optimize_norm_euclidian;
01609       break;
01610     case ERROR_NORM_MAXIMUM:
01611       optimize_norm = optimize_norm_maximum;
01612       break;
01613     case ERROR_NORM_P:
01614       optimize_norm = optimize_norm_p;
01615       optimize->p = input->p;
01616       break;
01617     default:
01618       optimize_norm = optimize_norm_taxicab;
01619     }
01620
01621   // Allocating values
01622 #if DEBUG_OPTIMIZE
01623   fprintf (stderr, "optimize_open: allocating variables\n");
01624   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01625            optimize->nvariables, optimize->algorithm);
01626 #endif
01627   optimize->genetic_variable = NULL;
01628   if (optimize->algorithm == ALGORITHM_GENETIC)
01629     {
01630       optimize->genetic_variable = (GeneticVariable *)
01631         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01632       for (i = 0; i < optimize->nvariables; ++i)
01633         {
01634 #if DEBUG_OPTIMIZE
01635           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01636                    i, optimize->rangemin[i], optimize->rangemax[i],
01637                    optimize->nbits[i]);
01638 #endif
01639           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
01640           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
01641           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01642         }
01643     }
```

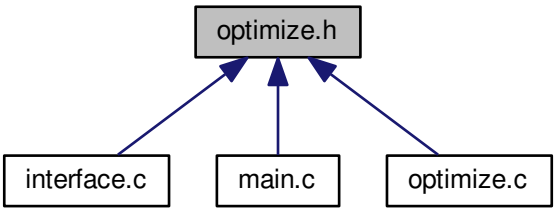
```
01644 #if DEBUG_OPTIMIZE
01645   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01646            optimize->nvariables, optimize->nsimulations);
01647 #endif
01648   optimize->value = (double *)
01649     g_malloc ((optimize->nsimulations
01650                + optimize->nestimates * optimize->nsteps)
01651               * optimize->nvariables * sizeof (double));
01652
01653   // Calculating simulations to perform for each task
01654 #if HAVE_MPI
01655 #if DEBUG_OPTIMIZE
01656   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01657            optimize->mpi_rank, ntasks);
01658 #endif
01659   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
01660   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
01661   if (optimize->nsteps)
01662     {
01663       optimize->nstart_direction
01664         = optimize->mpi_rank * optimize->nestimates / ntasks;
01665       optimize->nend_direction
01666         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
01667     }
01668 #else
01669   optimize->nstart = 0;
01670   optimize->nend = optimize->nsimulations;
01671   if (optimize->nsteps)
01672     {
01673       optimize->nstart_direction = 0;
01674       optimize->nend_direction = optimize->nestimates;
01675     }
01676 #endif
01677 #if DEBUG_OPTIMIZE
01678   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01679            optimize->nend);
01680 #endif
01681
01682   // Calculating simulations to perform for each thread
01683   optimize->thread
01684     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01685   for (i = 0; i <= nthreads; ++i)
01686     {
01687       optimize->thread[i] = optimize->nstart
01688         + i * (optimize->nend - optimize->nstart) / nthreads;
01689 #if DEBUG_OPTIMIZE
01690       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01691                optimize->thread[i]);
01692 #endif
01693     }
01694   if (optimize->nsteps)
01695     optimize->thread_direction = (unsigned int *)
01696       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01697
01698   // Opening result files
01699   optimize->file_result = g_fopen (optimize->result, "w");
01700   optimize->file_variables = g_fopen (optimize->variables, "w");
01701
01702   // Performing the algorithm
01703   switch (optimize->algorithm)
01704     {
01705       // Genetic algorithm
01706     case ALGORITHM_GENETIC:
01707       optimize_genetic ();
01708       break;
01709
01710       // Iterative algorithm
01711     default:
01712       optimize_iterate ();
01713     }
01714
01715   // Getting calculation time
01716   t = g_date_time_new_now (tz);
01717   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01718   g_date_time_unref (t);
01719   g_date_time_unref (t0);
01720   g_time_zone_unref (tz);
01721   printf ("%s = %.6lg s\n",
01722           _("Calculation time"), optimize->calculation_time);
01723   fprintf (optimize->file_result, "%s = %.6lg s\n",
01724            _("Calculation time"), optimize->calculation_time);
01725
01726   // Closing result files
01727   fclose (optimize->file_variables);
```

```
01728   fclose (optimize->file_result);
01729
01730 #if DEBUG_OPTIMIZE
01731   fprintf (stderr, "optimize_open: end\n");
01732 #endif
01733 }
```

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 *template)

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

  Function to parse input files, simulating and calculating the \ objective function.
- double optimize_norm_euclidian (unsigned int simulation)

  Function to calculate the Euclidian error norm.
- double optimize_norm_maximum (unsigned int simulation)

  Function to calculate the maximum error norm.
- double optimize_norm_p (unsigned int simulation)

  Function to calculate the P error norm.
- double optimize_norm_taxicab (unsigned int simulation)

  Function to calculate the taxicab error norm.
- void optimize_print ()

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

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

- void optimize_best (unsigned int simulation, double value)

  Function to save the best simulations.
- void optimize_sequential ()

  Function to optimize sequentially.
- void * optimize_thread (ParallelData *data)

  Function to optimize on a thread.
- void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)

  Function to merge the 2 optimization results.
- void optimize_synchronise ()

  Function to synchronise the optimization results of MPI tasks.
- void optimize_sweep ()

  Function to optimize with the sweep algorithm.
- void optimize_MonteCarlo ()

  Function to optimize with the Monte-Carlo algorithm.
- void optimize_best_direction (unsigned int simulation, double value)

  Function to save the best simulation in a direction search method.
- void optimize_direction_sequential (unsigned int simulation)

  Function to estimate the direction search sequentially.
- void * optimize_direction_thread (ParallelData *data)

  Function to estimate the direction search on a thread.
- double optimize_estimate_direction_random (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- double optimize_estimate_direction_coordinates (unsigned int variable, unsigned int estimate)

  Function to estimate a component of the direction search vector.
- void optimize_step_direction (unsigned int simulation)

  Function to do a step of the direction search method.
- void optimize_direction ()

  Function to optimize with a direction search method.
- double optimize_genetic_objective (**Entity** *entity)

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

  Function to optimize with the genetic algorithm.
- void optimize_save_old ()

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

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

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

  Function to do a step of the iterative algorithm.
- void optimize_iterate ()

  Function to iterate the algorithm.
- void optimize_free ()

  Function to free the memory used by the Optimize struct.
- void optimize_open ()

  Function to open and perform a optimization.

Variables

- int ntasks

  Number of tasks.
- unsigned int nthreads

  Number of threads.
- unsigned int nthreads_direction

  Number of threads for the direction search method.
- GMutex mutex [1]

  Mutex struct.
- 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

Copyright 2012-2016, all rights reserved.

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 468 of file optimize.c.

```
00469 {
00470   unsigned int i, j;
00471   double e;
00472 #if DEBUG_OPTIMIZE
00473   fprintf (stderr, "optimize_best: start\n");
00474   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475            optimize->nsaveds, optimize->nbest);
00476 #endif
00477   if (optimize->nsaveds < optimize->nbest
00478       || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480       if (optimize->nsaveds < optimize->nbest)
00481         ++optimize->nsaveds;
00482       optimize->error_best[optimize->nsaveds - 1] = value;
00483       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484       for (i = optimize->nsaveds; --i;)
00485         {
00486           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00487             {
00488               j = optimize->simulation_best[i];
00489               e = optimize->error_best[i];
00490               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00491               optimize->error_best[i] = optimize->
      error_best[i - 1];
00492               optimize->simulation_best[i - 1] = j;
00493               optimize->error_best[i - 1] = e;
00494             }
00495           else
00496             break;
00497         }
00498     }
00499 #if DEBUG_OPTIMIZE
00500   fprintf (stderr, "optimize_best: end\n");
00501 #endif
00502 }
```

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 795 of file optimize.c.

```
00796 {
00797 #if DEBUG_OPTIMIZE
00798   fprintf (stderr, "optimize_best_direction: start\n");
00799   fprintf (stderr,
00800            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801            simulation, value, optimize->error_best[0]);
00802 #endif
00803   if (value < optimize->error_best[0])
00804     {
00805       optimize->error_best[0] = value;
00806       optimize->simulation_best[0] = simulation;
00807 #if DEBUG_OPTIMIZE
```

```
00808       fprintf (stderr,
00809                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810                simulation, value);
00811 #endif
00812     }
00813 #if DEBUG_OPTIMIZE
00814   fprintf (stderr, "optimize_best_direction: end\n");
00815 #endif
00816 }
```

4.19.2.3 optimize_direction_sequential()

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

Function to estimate the direction search sequentially.

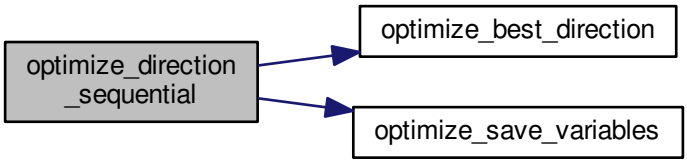
Parameters

simulation	Simulation number.

Definition at line 825 of file optimize.c.

```
00826 {
00827   unsigned int i, j;
00828   double e;
00829 #if DEBUG_OPTIMIZE
00830   fprintf (stderr, "optimize_direction_sequential: start\n");
00831   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832            "nend_direction=%u\n",
00833            optimize->nstart_direction, optimize->
      nend_direction);
00834 #endif
00835   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837       j = simulation + i;
00838       e = optimize_norm (j);
00839       optimize_best_direction (j, e);
00840       optimize_save_variables (j, e);
00841       if (e < optimize->threshold)
00842         {
00843           optimize->stop = 1;
00844           break;
00845         }
00846 #if DEBUG_OPTIMIZE
00847       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00848 #endif
00849     }
00850 #if DEBUG_OPTIMIZE
00851   fprintf (stderr, "optimize_direction_sequential: end\n");
00852 #endif
00853 }
```

Here is the call graph for this function:



4.19.2.4 optimize_direction_thread()

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

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

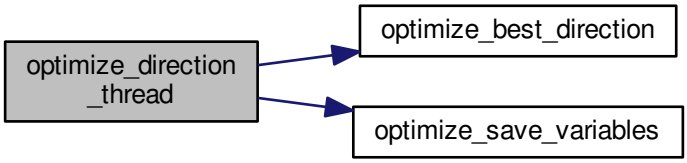
NULL

Definition at line 863 of file optimize.c.

```
00864 {
00865   unsigned int i, thread;
00866   double e;
00867 #if DEBUG_OPTIMIZE
00868   fprintf (stderr, "optimize_direction_thread: start\n");
00869 #endif
00870   thread = data->thread;
00871 #if DEBUG_OPTIMIZE
00872   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873            thread,
00874            optimize->thread_direction[thread],
00875            optimize->thread_direction[thread + 1]);
00876 #endif
00877   for (i = optimize->thread_direction[thread];
00878        i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880       e = optimize_norm (i);
00881       g_mutex_lock (mutex);
00882       optimize_best_direction (i, e);
00883       optimize_save_variables (i, e);
00884       if (e < optimize->threshold)
00885         optimize->stop = 1;
00886       g_mutex_unlock (mutex);
00887       if (optimize->stop)
00888         break;
00889 #if DEBUG_OPTIMIZE
00890       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
```

```
00891 #endif
00892     }
00893 #if DEBUG_OPTIMIZE
00894   fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896   g_thread_exit (NULL);
00897   return NULL;
00898 }
```

Here is the call graph for this function:



4.19.2.5 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.

```
00939 {
00940   double x;
00941 #if DEBUG_OPTIMIZE
00942   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00943 #endif
00944   x = optimize->direction[variable];
00945   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00946     {
00947       if (estimate & 1)
00948         x += optimize->step[variable];
00949       else
00950         x -= optimize->step[variable];
00951     }
00952 #if DEBUG_OPTIMIZE
00953   fprintf (stderr,
00954            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00955            variable, x);
00956   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00957 #endif
00958   return x;
00959 }
```

4.19.2.6 optimize_estimate_direction_random()

```
double optimize_estimate_direction_random (
             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 910 of file optimize.c.

```
00912 {
00913   double x;
00914 #if DEBUG_OPTIMIZE
00915   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00916 #endif
00917   x = optimize->direction[variable]
00918     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00919 #if DEBUG_OPTIMIZE
00920   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00921            variable, x);
00922   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00923 #endif
00924   return x;
00925 }
```

4.19.2.7 optimize_genetic_objective()

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

Function to calculate the objective function of an entity.

Parameters

entity	entity data.

Returns

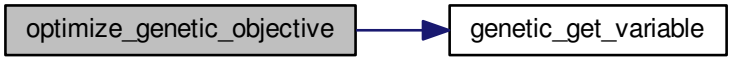
objective function value.

Definition at line 1104 of file optimize.c.

```
01105 {
01106   unsigned int j;
```

```
01107   double objective;
01108   char buffer[64];
01109 #if DEBUG_OPTIMIZE
01110   fprintf (stderr, "optimize_genetic_objective: start\n");
01111 #endif
01112   for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114       optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117   objective = optimize_norm (entity->id);
01118   g_mutex_lock (mutex);
01119   for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122       fprintf (optimize->file_variables, buffer,
01123                genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125   fprintf (optimize->file_variables, "%.14le\n", objective);
01126   g_mutex_unlock (mutex);
01127 #if DEBUG_OPTIMIZE
01128   fprintf (stderr, "optimize_genetic_objective: end\n");
01129 #endif
01130   return objective;
01131 }
```

Here is the call graph for this function:



4.19.2.8 optimize_input()

```
void optimize_input (
             unsigned int simulation,
             char * input,
             GMappedFile * template )
```

Function to write the simulation input file.

Parameters

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

Definition at line 104 of file optimize.c.

```
00105 {
00106   unsigned int i;
00107   char buffer[32], value[32], *buffer2, *buffer3, *content;
00108   FILE *file;
```

```
00109   gsize length;
00110   GRegex *regex;
00111
00112 #if DEBUG_OPTIMIZE
00113   fprintf (stderr, "optimize_input: start\n");
00114 #endif
00115
00116   // Checking the file
00117   if (!template)
00118     goto optimize_input_end;
00119
00120   // Opening template
00121   content = g_mapped_file_get_contents (template);
00122   length = g_mapped_file_get_length (template);
00123 #if DEBUG_OPTIMIZE
00124   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125            content);
00126 #endif
00127   file = g_fopen (input, "w");
00128
00129   // Parsing template
00130   for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132 #if DEBUG_OPTIMIZE
00133       fprintf (stderr, "optimize_input: variable=%u\n", i);
00134 #endif
00135       snprintf (buffer, 32, "@variable%u@", i + 1);
00136       regex = g_regex_new (buffer, 0, 0, NULL);
00137       if (i == 0)
00138         {
00139           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                              optimize->label[i], 0, NULL);
00141 #if DEBUG_OPTIMIZE
00142           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143 #endif
00144         }
00145       else
00146         {
00147           length = strlen (buffer3);
00148           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                              optimize->label[i], 0, NULL);
00150           g_free (buffer3);
00151         }
00152       g_regex_unref (regex);
00153       length = strlen (buffer2);
00154       snprintf (buffer, 32, "@value%u@", i + 1);
00155       regex = g_regex_new (buffer, 0, 0, NULL);
00156       snprintf (value, 32, format[optimize->precision[i]],
00157                 optimize->value[simulation * optimize->
      nvariables + i]);
00158
00159 #if DEBUG_OPTIMIZE
00160       fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                          0, NULL);
00164       g_free (buffer2);
00165       g_regex_unref (regex);
00166     }
00167
00168   // Saving input file
00169   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170   g_free (buffer3);
00171   fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175   fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177   return;
00178 }
```

4.19.2.9 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 591 of file optimize.c.

```
00593 {
00594   unsigned int i, j, k, s[optimize->nbest];
00595   double e[optimize->nbest];
00596 #if DEBUG_OPTIMIZE
00597   fprintf (stderr, "optimize_merge: start\n");
00598 #endif
00599   i = j = k = 0;
00600   do
00601     {
00602       if (i == optimize->nsaveds)
00603         {
00604           s[k] = simulation_best[j];
00605           e[k] = error_best[j];
00606           ++j;
00607           ++k;
00608           if (j == nsaveds)
00609             break;
00610         }
00611       else if (j == nsaveds)
00612         {
00613           s[k] = optimize->simulation_best[i];
00614           e[k] = optimize->error_best[i];
00615           ++i;
00616           ++k;
00617           if (i == optimize->nsaveds)
00618             break;
00619         }
00620       else if (optimize->error_best[i] > error_best[j])
00621         {
00622           s[k] = simulation_best[j];
00623           e[k] = error_best[j];
00624           ++j;
00625           ++k;
00626         }
00627       else
00628         {
00629           s[k] = optimize->simulation_best[i];
00630           e[k] = optimize->error_best[i];
00631           ++i;
00632           ++k;
00633         }
00634     }
00635   while (k < optimize->nbest);
00636   optimize->nsaveds = k;
00637   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638   memcpy (optimize->error_best, e, k * sizeof (double));
00639 #if DEBUG_OPTIMIZE
00640   fprintf (stderr, "optimize_merge: end\n");
00641 #endif
00642 }
```

4.19.2.10 optimize_norm_euclidian()

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

Function to calculate the Euclidian error norm.

Parameters

simulation	simulation number.

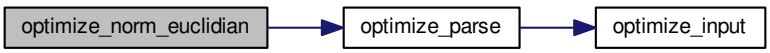
Returns

Euclidian error norm.

Definition at line 300 of file optimize.c.

```
00301 {
00302   double e, ei;
00303   unsigned int i;
00304 #if DEBUG_OPTIMIZE
00305   fprintf (stderr, "optimize_norm_euclidian: start\n");
00306 #endif
00307   e = 0.;
00308   for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310       ei = optimize_parse (simulation, i);
00311       e += ei * ei;
00312     }
00313   e = sqrt (e);
00314 #if DEBUG_OPTIMIZE
00315   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316   fprintf (stderr, "optimize_norm_euclidian: end\n");
00317 #endif
00318   return e;
00319 }
```

Here is the call graph for this function:



4.19.2.11 optimize_norm_maximum()

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

Function to calculate the maximum error norm.

Parameters

simulation	simulation number.

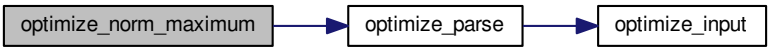
Returns

Maximum error norm.

Definition at line 329 of file optimize.c.

```
00330 {
00331   double e, ei;
00332   unsigned int i;
00333 #if DEBUG_OPTIMIZE
00334   fprintf (stderr, "optimize_norm_maximum: start\n");
00335 #endif
00336   e = 0.;
00337   for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339       ei = fabs (optimize_parse (simulation, i));
00340       e = fmax (e, ei);
00341     }
00342 #if DEBUG_OPTIMIZE
00343   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344   fprintf (stderr, "optimize_norm_maximum: end\n");
00345 #endif
00346   return e;
00347 }
```

Here is the call graph for this function:



4.19.2.12 optimize_norm_p()

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

Function to calculate the P error norm.

Parameters

simulation	simulation number.

Returns

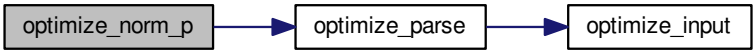
P error norm.

Definition at line 357 of file optimize.c.

```
00358 {
00359   double e, ei;
00360   unsigned int i;
00361 #if DEBUG_OPTIMIZE
00362   fprintf (stderr, "optimize_norm_p: start\n");
00363 #endif
00364   e = 0.;
00365   for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367       ei = fabs (optimize_parse (simulation, i));
```

```
00368       e += pow (ei, optimize->p);
00369     }
00370   e = pow (e, 1. / optimize->p);
00371 #if DEBUG_OPTIMIZE
00372   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373   fprintf (stderr, "optimize_norm_p: end\n");
00374 #endif
00375   return e;
00376 }
```

Here is the call graph for this function:



4.19.2.13 optimize_norm_taxicab()

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

Function to calculate the taxicab error norm.

Parameters

simulation	simulation number.

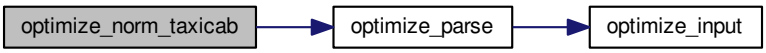
Returns

Taxicab error norm.

Definition at line 386 of file optimize.c.

```
00387 {
00388   double e;
00389   unsigned int i;
00390 #if DEBUG_OPTIMIZE
00391   fprintf (stderr, "optimize_norm_taxicab: start\n");
00392 #endif
00393   e = 0.;
00394   for (i = 0; i < optimize->nexperiments; ++i)
00395     e += fabs (optimize_parse (simulation, i));
00396 #if DEBUG_OPTIMIZE
00397   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398   fprintf (stderr, "optimize_norm_taxicab: end\n");
00399 #endif
00400   return e;
00401 }
```

Here is the call graph for this function:



4.19.2.14 optimize_parse()

```
double optimize_parse (
            unsigned int simulation,
            unsigned int experiment )
```

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

Parameters

simulation	Simulation number.
experiment	Experiment number.

Returns

Objective function value.

Definition at line 191 of file optimize.c.

```
00192 {
00193   unsigned int i;
00194   double e;
00195   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196     *buffer3, *buffer4;
00197   FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200   fprintf (stderr, "optimize_parse: start\n");
00201   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202            simulation, experiment);
00203 #endif
00204
00205   // Opening input files
00206   for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210 #if DEBUG_OPTIMIZE
00211       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
00213       optimize_input (simulation, &input[i][0],
00214                       optimize->file[i][experiment]);
00215     }
00216   for (; i < MAX_NINPUTS; ++i)
00217     strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219   fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222   // Performing the simulation
```

```
00223   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224   buffer2 = g_path_get_dirname (optimize->simulator);
00225   buffer3 = g_path_get_basename (optimize->simulator);
00226   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230   g_free (buffer4);
00231   g_free (buffer3);
00232   g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234   fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236   system (buffer);
00237
00238   // Checking the objective value function
00239   if (optimize->evaluator)
00240     {
00241       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242       buffer2 = g_path_get_dirname (optimize->evaluator);
00243       buffer3 = g_path_get_basename (optimize->evaluator);
00244       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245       snprintf (buffer, 512, "\"%s\" %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247       g_free (buffer4);
00248       g_free (buffer3);
00249       g_free (buffer2);
00250 #if DEBUG_OPTIMIZE
00251       fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253       system (buffer);
00254       file_result = g_fopen (result, "r");
00255       e = atof (fgets (buffer, 512, file_result));
00256       fclose (file_result);
00257     }
00258   else
00259     {
00260       strcpy (result, "");
00261       file_result = g_fopen (output, "r");
00262       e = atof (fgets (buffer, 512, file_result));
00263       fclose (file_result);
00264     }
00265
00266   // Removing files
00267 #if !DEBUG_OPTIMIZE
00268   for (i = 0; i < optimize->ninputs; ++i)
00269     {
00270       if (optimize->file[i][0])
00271         {
00272           snprintf (buffer, 512, RM " %s", &input[i][0]);
00273           system (buffer);
00274         }
00275     }
00276   snprintf (buffer, 512, RM " %s %s", output, result);
00277   system (buffer);
00278 #endif
00279
00280   // Processing pending events
00281   if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285   fprintf (stderr, "optimize_parse: end\n");
00286 #endif
00287
00288   // Returning the objective function
00289   return e * optimize->weight[experiment];
00290 }
```

Here is the call graph for this function:

4.19.2.15 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 439 of file optimize.c.

```
00440 {
00441   unsigned int i;
00442   char buffer[64];
00443 #if DEBUG_OPTIMIZE
00444   fprintf (stderr, "optimize_save_variables: start\n");
00445 #endif
00446   for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449       fprintf (optimize->file_variables, buffer,
00450                optimize->value[simulation * optimize->
      nvariables + i]);
00451     }
00452   fprintf (optimize->file_variables, "%.14le\n", error);
00453   fflush (optimize->file_variables);
00454 #if DEBUG_OPTIMIZE
00455   fprintf (stderr, "optimize_save_variables: end\n");
00456 #endif
00457 }
```

4.19.2.16 optimize_step_direction()

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

Function to do a step of the direction search method.

Parameters

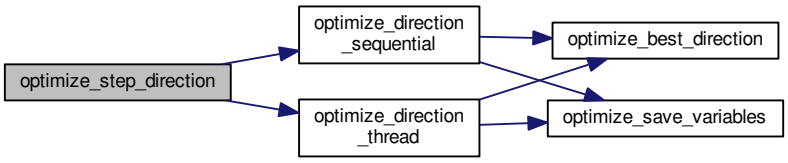
simulation	Simulation number.

Definition at line 968 of file optimize.c.

```
00969 {
00970   GThread *thread[nthreads_direction];
00971   ParallelData data[nthreads_direction];
00972   unsigned int i, j, k, b;
00973 #if DEBUG_OPTIMIZE
```

```
00974   fprintf (stderr, "optimize_step_direction: start\n");
00975 #endif
00976   for (i = 0; i < optimize->nestimates; ++i)
00977     {
00978       k = (simulation + i) * optimize->nvariables;
00979       b = optimize->simulation_best[0] * optimize->
      nvariables;
00980 #if DEBUG_OPTIMIZE
00981       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00982                simulation + i, optimize->simulation_best[0]);
00983 #endif
00984       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00985         {
00986 #if DEBUG_OPTIMIZE
00987           fprintf (stderr,
00988                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989                    i, j, optimize->value[b]);
00990 #endif
00991           optimize->value[k]
00992             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00993           optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                            optimize->rangeminabs[j]),
00995                                      optimize->rangemaxabs[j]);
00996 #if DEBUG_OPTIMIZE
00997           fprintf (stderr,
00998                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999                    i, j, optimize->value[k]);
01000 #endif
01001         }
01002     }
01003   if (nthreads_direction == 1)
01004     optimize_direction_sequential (simulation);
01005   else
01006     {
01007       for (i = 0; i <= nthreads_direction; ++i)
01008         {
01009           optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01012             / nthreads_direction;
01013 #if DEBUG_OPTIMIZE
01014           fprintf (stderr,
01015                    "optimize_step_direction: i=%u thread_direction=%u\n",
01016                    i, optimize->thread_direction[i]);
01017 #endif
01018         }
01019       for (i = 0; i < nthreads_direction; ++i)
01020         {
01021           data[i].thread = i;
01022           thread[i] = g_thread_new
01023             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01024         }
01025       for (i = 0; i < nthreads_direction; ++i)
01026         g_thread_join (thread[i]);
01027     }
01028 #if DEBUG_OPTIMIZE
01029   fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031 }
```

Here is the call graph for this function:



4.19.2.17 optimize_thread()

```
void* optimize_thread (
            ParallelData * data )
```

Function to optimize on a thread.

Parameters

data	Function data.

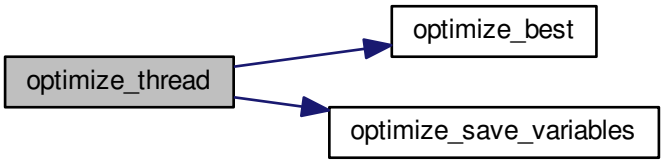
Returns

NULL

Definition at line 545 of file optimize.c.

```
00546 {
00547   unsigned int i, thread;
00548   double e;
00549 #if DEBUG_OPTIMIZE
00550   fprintf (stderr, "optimize_thread: start\n");
00551 #endif
00552   thread = data->thread;
00553 #if DEBUG_OPTIMIZE
00554   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555            optimize->thread[thread], optimize->thread[thread + 1]);
00556 #endif
00557   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559       e = optimize_norm (i);
00560       g_mutex_lock (mutex);
00561       optimize_best (i, e);
00562       optimize_save_variables (i, e);
00563       if (e < optimize->threshold)
00564         optimize->stop = 1;
00565       g_mutex_unlock (mutex);
00566       if (optimize->stop)
00567         break;
00568 #if DEBUG_OPTIMIZE
00569       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570 #endif
00571     }
00572 #if DEBUG_OPTIMIZE
00573   fprintf (stderr, "optimize_thread: end\n");
00574 #endif
00575   g_thread_exit (NULL);
00576   return NULL;
00577 }
```

Here is the call graph for this function:



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 2012-2016, 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 * template);
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_best_direction (unsigned int simulation, double value);
00159 void optimize_direction_sequential (unsigned int simulation);
00160 void *optimize_direction_thread (ParallelData * data);
00161 double optimize_estimate_direction_random (unsigned int variable,
00162                                            unsigned int estimate);
00163 double optimize_estimate_direction_coordinates (unsigned int
      variable,
00164                                                 unsigned int estimate);
00165 void optimize_step_direction (unsigned int simulation);
00166 void optimize_direction ();
00167 double optimize_genetic_objective (Entity * entity);
00168 void optimize_genetic ();
00169 void optimize_save_old ();
00170 void optimize_merge_old ();
00171 void optimize_refine ();
00172 void optimize_step ();
00173 void optimize_iterate ();
00174 void optimize_free ();
00175 void optimize_open ();
00176
00177 #endif
```

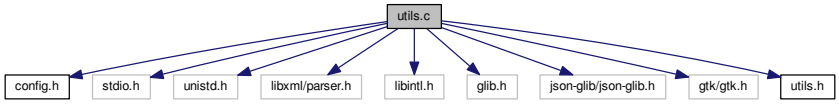
4.21 utils.c File Reference

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)

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

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

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

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

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

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

  Function to set a floating point number in a XML node property.
- int json_object_get_int (JsonObject *object, const char *prop, int *error_code)

  Function to get an integer number of a JSON object property.
- 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.
- 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.
- 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.
- 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.
- void json_object_set_int (JsonObject *object, const char *prop, int value)

  Function to set an integer number in a JSON object property.
- 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.

- void json_object_set_float (JsonObject *object, const char *prop, double value)

  Function to set a floating point number in a JSON object property.
- int cores_number ()

  Function to obtain the cores number.
- void process_pending ()

  Function to process events on long computation.
- unsigned int gtk_array_get_active (GtkRadioButton *array[ ], unsigned int n)

  Function to get the active GtkRadioButton.

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 2012-2016, 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 532 of file utils.c.

```
00533 {
00534 #ifdef G_OS_WIN32
00535   SYSTEM_INFO sysinfo;
00536   GetSystemInfo (&sysinfo);
00537   return sysinfo.dwNumberOfProcessors;
00538 #else
00539   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540 #endif
00541 }
```

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.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 567 of file utils.c.

```
00568 {
00569   unsigned int i;
00570   for (i = 0; i < n; ++i)
00571     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572       break;
00573   return i;
00574 }
```

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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 421 of file utils.c.

```
00422 {
```

```
00423   const char *buffer;
00424   double x = 0.;
00425   buffer = json_object_get_string_member (object, prop);
00426   if (!buffer)
00427     *error_code = 1;
00428   else
00429     {
00430       if (sscanf (buffer, "%lf", &x) != 1)
00431         *error_code = 2;
00432       else
00433         *error_code = 0;
00434     }
00435   return x;
00436 }
```

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.

Parameters

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

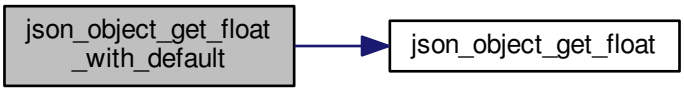
Returns

Floating point number value.

Definition at line 454 of file utils.c.

```
00456 {
00457   double x;
00458   if (json_object_get_member (object, prop))
00459     x = json_object_get_float (object, prop, error_code);
00460   else
00461     {
00462       x = default_value;
00463       *error_code = 0;
00464     }
00465   return x;
00466 }
```

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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Integer number value.

Definition at line 330 of file utils.c.

```
00331 {
00332   const char *buffer;
00333   int i = 0;
00334   buffer = json_object_get_string_member (object, prop);
00335   if (!buffer)
00336     *error_code = 1;
00337   else
00338     {
00339       if (sscanf (buffer, "%d", &i) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343     }
00344   return i;
00345 }
```

4.21.2.6 json_object_get_uint()

```
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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 360 of file utils.c.

```
00361 {
00362   const char *buffer;
00363   unsigned int i = 0;
00364   buffer = json_object_get_string_member (object, prop);
00365   if (!buffer)
00366     *error_code = 1;
00367   else
00368     {
00369       if (sscanf (buffer, "%u", &i) != 1)
00370         *error_code = 2;
00371       else
00372         *error_code = 0;
00373     }
00374   return i;
00375 }
```

4.21.2.7 json_object_get_uint_with_default()

```
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.

Parameters

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

Returns

Unsigned integer number value.

Definition at line 393 of file utils.c.

```
00396 {
00397   unsigned int i;
00398   if (json_object_get_member (object, prop))
00399     i = json_object_get_uint (object, prop, error_code);
00400   else
00401     {
00402       i = default_value;
00403       *error_code = 0;
00404     }
00405   return i;
00406 }
```

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 519 of file utils.c.

```
00520 {
00521   char buffer[64];
00522   snprintf (buffer, 64, "%.14lg", value);
00523   json_object_set_string_member (object, prop, buffer);
00524 }
```

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 480 of file utils.c.

```
00481 {
00482   char buffer[64];
00483   snprintf (buffer, 64, "%d", value);
00484   json_object_set_string_member (object, prop, buffer);
00485 }
```

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 499 of file utils.c.

```
00501 {
00502   char buffer[64];
00503   snprintf (buffer, 64, "%u", value);
00504   json_object_set_string_member (object, prop, buffer);
00505 }
```

4.21.2.11 show_error()

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

Function to show a dialog with an error message.

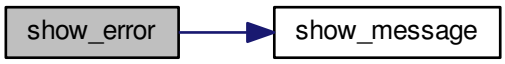
Parameters

msg	Error message.

Definition at line 103 of file utils.c.

```
00104 {
00105   show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
```

Here is the call graph for this function:



4.21.2.12 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 73 of file utils.c.

```
00074 {
00075 #if HAVE_GTK
```

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

4.21.2.13 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.

Parameters

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

Returns

Floating point number value.

Definition at line 213 of file utils.c.

```
00214 {
00215   double x = 0.;
00216   xmlChar *buffer;
00217   buffer = xmlGetProp (node, prop);
00218   if (!buffer)
00219     *error_code = 1;
00220   else
00221     {
00222       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223         *error_code = 2;
00224       else
00225         *error_code = 0;
00226       xmlFree (buffer);
00227     }
00228   return x;
00229 }
```

4.21.2.14 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.

Parameters

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

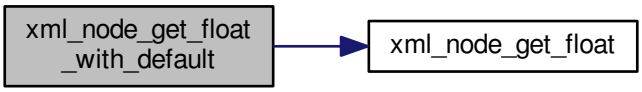
Returns

Floating point number value.

Definition at line 247 of file utils.c.

```
00249 {
00250   double x;
00251   if (xmlHasProp (node, prop))
00252     x = xml_node_get_float (node, prop, error_code);
00253   else
00254     {
00255       x = default_value;
00256       *error_code = 0;
00257     }
00258   return x;
00259 }
```

Here is the call graph for this function:



4.21.2.15 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.

Parameters

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

Returns

Integer number value.

Definition at line 121 of file utils.c.

```
00122 {
00123   int i = 0;
00124   xmlChar *buffer;
00125   buffer = xmlGetProp (node, prop);
00126   if (!buffer)
00127     *error_code = 1;
00128   else
00129     {
00130       if (sscanf ((char *) buffer, "%d", &i) != 1)
00131         *error_code = 2;
00132       else
00133         *error_code = 0;
00134       xmlFree (buffer);
00135     }
00136   return i;
00137 }
```

4.21.2.16 xml_node_get_uint()

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

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

Parameters

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

Returns

Unsigned integer number value.

Definition at line 152 of file utils.c.

```
00153 {
00154   unsigned int i = 0;
00155   xmlChar *buffer;
00156   buffer = xmlGetProp (node, prop);
```

```
00157   if (!buffer)
00158     *error_code = 1;
00159   else
00160     {
00161       if (sscanf ((char *) buffer, "%u", &i) != 1)
00162         *error_code = 2;
00163       else
00164         *error_code = 0;
00165       xmlFree (buffer);
00166     }
00167   return i;
00168 }
```

4.21.2.17 xml_node_get_uint_with_default()

```
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.

Parameters

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

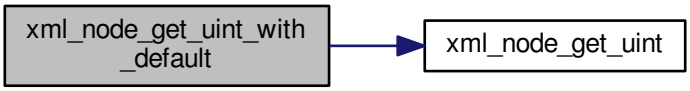
Returns

Unsigned integer number value.

Definition at line 186 of file utils.c.

```
00188 {
00189   unsigned int i;
00190   if (xmlHasProp (node, prop))
00191     i = xml_node_get_uint (node, prop, error_code);
00192   else
00193     {
00194       i = default_value;
00195       *error_code = 0;
00196     }
00197   return i;
00198 }
```

Here is the call graph for this function:



4.21.2.18 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 310 of file utils.c.

```
00311 {
00312   xmlChar buffer[64];
00313   snprintf ((char *) buffer, 64, "%.14lg", value);
00314   xmlSetProp (node, prop, buffer);
00315 }
```

4.21.2.19 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 272 of file utils.c.

```
00273 {
00274   xmlChar buffer[64];
00275   snprintf ((char *) buffer, 64, "%d", value);
00276   xmlSetProp (node, prop, buffer);
00277 }
```

4.21.2.20 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 291 of file utils.c.

```
00292 {
00293   xmlChar buffer[64];
00294   snprintf ((char *) buffer, 64, "%u", value);
00295   xmlSetProp (node, prop, buffer);
00296 }
```

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 2012-2016, 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
00072 void
00073 show_message (char *title, char *msg, int type)
00074 {
00075 #if HAVE_GTK
00076   GtkMessageDialog *dlg;
00077
00078   // Creating the dialog
00079   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00080     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00081
00082   // Setting the dialog title
00083   gtk_window_set_title (GTK_WINDOW (dlg), title);
00084
00085   // Showing the dialog and waiting response
00086   gtk_dialog_run (GTK_DIALOG (dlg));
00087
00088   // Closing and freeing memory
00089   gtk_widget_destroy (GTK_WIDGET (dlg));
00090
00091 #else
00092   printf ("%s: %s\n", title, msg);
00093 #endif
00094 }
00095
00102 void
00103 show_error (char *msg)
00104 {
00105   show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
00107
00120 int
00121 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00122 {
00123   int i = 0;
00124   xmlChar *buffer;
00125   buffer = xmlGetProp (node, prop);
00126   if (!buffer)
00127     *error_code = 1;
00128   else
00129     {
00130       if (sscanf ((char *) buffer, "%d", &i) != 1)
00131         *error_code = 2;
00132       else
00133         *error_code = 0;
00134       xmlFree (buffer);
00135     }
00136   return i;
00137 }
00138
00151 unsigned int
00152 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00153 {
00154   unsigned int i = 0;
00155   xmlChar *buffer;
00156   buffer = xmlGetProp (node, prop);
00157   if (!buffer)
00158     *error_code = 1;
00159   else
00160     {
00161       if (sscanf ((char *) buffer, "%u", &i) != 1)
00162         *error_code = 2;
00163       else
00164         *error_code = 0;
00165       xmlFree (buffer);
00166     }
00167   return i;
00168 }
00169
00185 unsigned int
00186 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00187                                 unsigned int default_value, int *error_code)
00188 {
00189   unsigned int i;
00190   if (xmlHasProp (node, prop))
00191     i = xml_node_get_uint (node, prop, error_code);
00192   else
00193     {
```

```
00194       i = default_value;
00195       *error_code = 0;
00196     }
00197   return i;
00198 }
00199
00212 double
00213 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00214 {
00215   double x = 0.;
00216   xmlChar *buffer;
00217   buffer = xmlGetProp (node, prop);
00218   if (!buffer)
00219     *error_code = 1;
00220   else
00221     {
00222       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223         *error_code = 2;
00224       else
00225         *error_code = 0;
00226       xmlFree (buffer);
00227     }
00228   return x;
00229 }
00230
00246 double
00247 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00248                                  double default_value, int *error_code)
00249 {
00250   double x;
00251   if (xmlHasProp (node, prop))
00252     x = xml_node_get_float (node, prop, error_code);
00253   else
00254     {
00255       x = default_value;
00256       *error_code = 0;
00257     }
00258   return x;
00259 }
00260
00271 void
00272 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00273 {
00274   xmlChar buffer[64];
00275   snprintf ((char *) buffer, 64, "%d", value);
00276   xmlSetProp (node, prop, buffer);
00277 }
00278
00290 void
00291 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00292 {
00293   xmlChar buffer[64];
00294   snprintf ((char *) buffer, 64, "%u", value);
00295   xmlSetProp (node, prop, buffer);
00296 }
00297
00309 void
00310 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00311 {
00312   xmlChar buffer[64];
00313   snprintf ((char *) buffer, 64, "%.14lg", value);
00314   xmlSetProp (node, prop, buffer);
00315 }
00316
00329 int
00330 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00331 {
00332   const char *buffer;
00333   int i = 0;
00334   buffer = json_object_get_string_member (object, prop);
00335   if (!buffer)
00336     *error_code = 1;
00337   else
00338     {
00339       if (sscanf (buffer, "%d", &i) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343     }
00344   return i;
00345 }
00346
00359 unsigned int
00360 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00361 {
00362   const char *buffer;
00363   unsigned int i = 0;
```

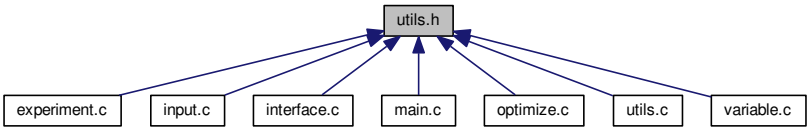
```
00364   buffer = json_object_get_string_member (object, prop);
00365   if (!buffer)
00366     *error_code = 1;
00367   else
00368     {
00369       if (sscanf (buffer, "%u", &i) != 1)
00370         *error_code = 2;
00371       else
00372         *error_code = 0;
00373     }
00374   return i;
00375 }
00376
00392 unsigned int
00393 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00394                                    unsigned int default_value,
00395                                    int *error_code)
00396 {
00397   unsigned int i;
00398   if (json_object_get_member (object, prop))
00399     i = json_object_get_uint (object, prop, error_code);
00400   else
00401     {
00402       i = default_value;
00403       *error_code = 0;
00404     }
00405   return i;
00406 }
00407
00420 double
00421 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00422 {
00423   const char *buffer;
00424   double x = 0.;
00425   buffer = json_object_get_string_member (object, prop);
00426   if (!buffer)
00427     *error_code = 1;
00428   else
00429     {
00430       if (sscanf (buffer, "%lf", &x) != 1)
00431         *error_code = 2;
00432       else
00433         *error_code = 0;
00434     }
00435   return x;
00436 }
00437
00453 double
00454 json_object_get_float_with_default (JsonObject * object, const char *prop
      ,
00455                                     double default_value, int *error_code)
00456 {
00457   double x;
00458   if (json_object_get_member (object, prop))
00459     x = json_object_get_float (object, prop, error_code);
00460   else
00461     {
00462       x = default_value;
00463       *error_code = 0;
00464     }
00465   return x;
00466 }
00467
00479 void
00480 json_object_set_int (JsonObject * object, const char *prop, int value)
00481 {
00482   char buffer[64];
00483   snprintf (buffer, 64, "%d", value);
00484   json_object_set_string_member (object, prop, buffer);
00485 }
00486
00498 void
00499 json_object_set_uint (JsonObject * object, const char *prop,
00500                       unsigned int value)
00501 {
00502   char buffer[64];
00503   snprintf (buffer, 64, "%u", value);
00504   json_object_set_string_member (object, prop, buffer);
00505 }
00506
00518 void
00519 json_object_set_float (JsonObject * object, const char *prop, double value)
00520 {
00521   char buffer[64];
00522   snprintf (buffer, 64, "%.14lg", value);
00523   json_object_set_string_member (object, prop, buffer);
00524 }
```

```
00525
00531 int
00532 cores_number ()
00533 {
00534 #ifdef G_OS_WIN32
00535   SYSTEM_INFO sysinfo;
00536   GetSystemInfo (&sysinfo);
00537   return sysinfo.dwNumberOfProcessors;
00538 #else
00539   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540 #endif
00541 }
00542
00543 #if HAVE_GTK
00544
00549 void
00550 process_pending ()
00551 {
00552   while (gtk_events_pending ())
00553     gtk_main_iteration ();
00554 }
00555
00566 unsigned int
00567 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00568 {
00569   unsigned int i;
00570   for (i = 0; i < n; ++i)
00571     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572       break;
00573   return i;
00574 }
00575
00576 #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)

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

  Function to show a dialog with an error message.

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

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

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

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

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

  Function to set a floating point number in a XML node property.
- int json_object_get_int (JsonObject *object, const char *prop, int *error_code)

  Function to get an integer number of a JSON object property.
- 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.
- 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.
- 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.
- 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.
- void json_object_set_int (JsonObject *object, const char *prop, int value)

  Function to set an integer number in a JSON object property.
- 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.
- void json_object_set_float (JsonObject *object, const char *prop, double value)

  Function to set a floating point number in a JSON object property.
- int cores_number ()

  Function to obtain the cores number.
- void process_pending ()

  Function to process events on long computation.
- unsigned int gtk_array_get_active (GtkRadioButton *array[], unsigned int n)

  Function to get the active GtkRadioButton.

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

Copyright 2012-2016, all rights reserved.

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 532 of file utils.c.

```
00533 {
00534 #ifdef G_OS_WIN32
00535   SYSTEM_INFO sysinfo;
00536   GetSystemInfo (&sysinfo);
00537   return sysinfo.dwNumberOfProcessors;
00538 #else
00539   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540 #endif
00541 }
```

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.

Parameters

array	Array of GtkRadioButtons.
n	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 567 of file utils.c.

```
00568 {
00569   unsigned int i;
00570   for (i = 0; i < n; ++i)
00571     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572       break;
00573   return i;
00574 }
```

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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Floating point number value.

Definition at line 421 of file utils.c.

```
00422 {
00423   const char *buffer;
00424   double x = 0.;
00425   buffer = json_object_get_string_member (object, prop);
00426   if (!buffer)
00427     *error_code = 1;
00428   else
00429     {
00430       if (sscanf (buffer, "%lf", &x) != 1)
00431         *error_code = 2;
00432       else
00433         *error_code = 0;
00434     }
00435   return x;
00436 }
```

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.

Parameters

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

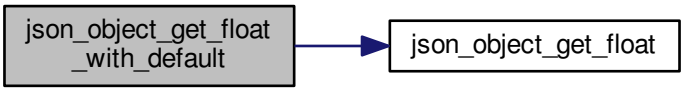
Returns

Floating point number value.

Definition at line 454 of file utils.c.

```
00456 {
00457   double x;
00458   if (json_object_get_member (object, prop))
00459     x = json_object_get_float (object, prop, error_code);
00460   else
00461     {
00462       x = default_value;
00463       *error_code = 0;
00464     }
00465   return x;
00466 }
```

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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Integer number value.

Definition at line 330 of file utils.c.

```
00331 {
00332   const char *buffer;
00333   int i = 0;
00334   buffer = json_object_get_string_member (object, prop);
00335   if (!buffer)
00336     *error_code = 1;
00337   else
00338     {
00339       if (sscanf (buffer, "%d", &i) != 1)
00340         *error_code = 2;
00341       else
00342         *error_code = 0;
00343     }
00344   return i;
00345 }
```

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.

Parameters

object	JSON object.
prop	JSON property.
error_code	Error code.

Returns

Unsigned integer number value.

Definition at line 360 of file utils.c.

```
00361 {
00362   const char *buffer;
00363   unsigned int i = 0;
00364   buffer = json_object_get_string_member (object, prop);
```

```
00365   if (!buffer)
00366     *error_code = 1;
00367   else
00368     {
00369       if (sscanf (buffer, "%u", &i) != 1)
00370         *error_code = 2;
00371       else
00372         *error_code = 0;
00373     }
00374   return i;
00375 }
```

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.

Parameters

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

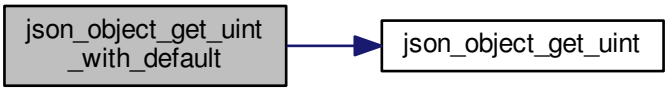
Returns

Unsigned integer number value.

Definition at line 393 of file utils.c.

```
00396 {
00397   unsigned int i;
00398   if (json_object_get_member (object, prop))
00399     i = json_object_get_uint (object, prop, error_code);
00400   else
00401     {
00402       i = default_value;
00403       *error_code = 0;
00404     }
00405   return i;
00406 }
```

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 519 of file utils.c.

```
00520 {
00521   char buffer[64];
00522   snprintf (buffer, 64, "%.14lg", value);
00523   json_object_set_string_member (object, prop, buffer);
00524 }
```

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 480 of file utils.c.

```
00481 {
00482   char buffer[64];
00483   snprintf (buffer, 64, "%d", value);
00484   json_object_set_string_member (object, prop, buffer);
00485 }
```

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 499 of file utils.c.

```
00501 {
00502   char buffer[64];
00503   snprintf (buffer, 64, "%u", value);
00504   json_object_set_string_member (object, prop, buffer);
00505 }
```

4.23.2.11 show_error()

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

Function to show a dialog with an error message.

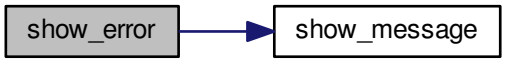
Parameters

msg	Error message.

Definition at line 103 of file utils.c.

```
00104 {
00105   show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
```

Here is the call graph for this function:



4.23.2.12 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 73 of file utils.c.

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

4.23.2.13 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.

Parameters

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

Returns

Floating point number value.

Definition at line 213 of file utils.c.

```
00214 {
00215   double x = 0.;
00216   xmlChar *buffer;
00217   buffer = xmlGetProp (node, prop);
00218   if (!buffer)
00219     *error_code = 1;
00220   else
00221     {
00222       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223         *error_code = 2;
00224       else
00225         *error_code = 0;
00226       xmlFree (buffer);
00227     }
00228   return x;
00229 }
```

4.23.2.14 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.

Parameters

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

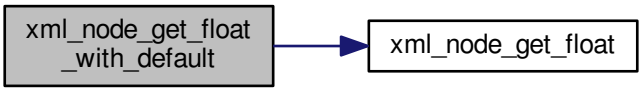
Returns

Floating point number value.

Definition at line 247 of file utils.c.

```
00249 {
00250   double x;
00251   if (xmlHasProp (node, prop))
00252     x = xml_node_get_float (node, prop, error_code);
00253   else
00254     {
00255       x = default_value;
00256       *error_code = 0;
00257     }
00258   return x;
00259 }
```

Here is the call graph for this function:



4.23.2.15 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.

Parameters

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

Returns

Integer number value.

Definition at line 121 of file utils.c.

```
00122 {
00123   int i = 0;
00124   xmlChar *buffer;
00125   buffer = xmlGetProp (node, prop);
00126   if (!buffer)
00127     *error_code = 1;
00128   else
00129     {
00130       if (sscanf ((char *) buffer, "%d", &i) != 1)
00131         *error_code = 2;
00132       else
00133         *error_code = 0;
00134       xmlFree (buffer);
00135     }
00136   return i;
00137 }
```

4.23.2.16 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.

Parameters

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

Returns

Unsigned integer number value.

Definition at line 152 of file utils.c.

```
00153 {
00154   unsigned int i = 0;
00155   xmlChar *buffer;
00156   buffer = xmlGetProp (node, prop);
00157   if (!buffer)
00158     *error_code = 1;
00159   else
00160     {
00161       if (sscanf ((char *) buffer, "%u", &i) != 1)
00162         *error_code = 2;
00163       else
00164         *error_code = 0;
00165       xmlFree (buffer);
00166     }
00167   return i;
00168 }
```

4.23.2.17 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.

Parameters

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

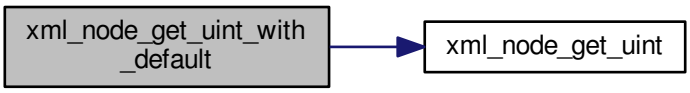
Returns

Unsigned integer number value.

Definition at line 186 of file utils.c.

```
00188 {
00189   unsigned int i;
00190   if (xmlHasProp (node, prop))
00191     i = xml_node_get_uint (node, prop, error_code);
00192   else
00193     {
00194       i = default_value;
00195       *error_code = 0;
00196     }
00197   return i;
00198 }
```

Here is the call graph for this function:



4.23.2.18 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 310 of file utils.c.

```
00311 {
00312   xmlChar buffer[64];
00313   snprintf ((char *) buffer, 64, "%.14lg", value);
00314   xmlSetProp (node, prop, buffer);
00315 }
```

4.23.2.19 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 272 of file utils.c.

```
00273 {
00274   xmlChar buffer[64];
00275   snprintf ((char *) buffer, 64, "%d", value);
00276   xmlSetProp (node, prop, buffer);
00277 }
```

4.23.2.20 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 291 of file utils.c.

```
00292 {
00293   xmlChar buffer[64];
00294   snprintf ((char *) buffer, 64, "%u", value);
00295   xmlSetProp (node, prop, buffer);
00296 }
```

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 2012-2016, 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,
00073                                         int *error_code);
00074 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00075 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00076                         unsigned int value);
00077 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00078 int json_object_get_int (JsonObject * object, const char *prop,
00079                          int *error_code);
00080 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00081                                    int *error_code);
00082 unsigned int json_object_get_uint_with_default (JsonObject * object,
00083                                                 const char *prop,
00084                                                 unsigned int default_value,
00085                                                 int *error_code);
00086 double json_object_get_float (JsonObject * object, const char *prop,
00087                               int *error_code);
00088 double json_object_get_float_with_default (JsonObject * object,
00089                                            const char *prop,
00090                                            double default_value,
00091                                            int *error_code);
00092 void json_object_set_int (JsonObject * object, const char *prop, int value);
00093 void json_object_set_uint (JsonObject * object, const char *prop,
00094                            unsigned int value);
00095 void json_object_set_float (JsonObject * object, const char *prop,
00096                             double value);
00097 int cores_number ();
00098 #if HAVE_GTK
00099 void process_pending ();
```

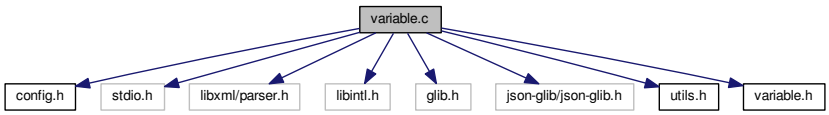
```
00100 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00101 #endif
00102
00103 #endif
```

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)

  Function to create a new Variable struct.
- void variable_free (Variable *variable, unsigned int type)

  Function to free the memory of a Variable struct.
- void variable_error (Variable *variable, char *message)

  Function to print a message error opening an Variable struct.
- int variable_open_xml (Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.
- int variable_open_json (Variable *variable, JsonNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.

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

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 110 of file variable.c.

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

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 87 of file variable.c.

```
00088 {
00089 #if DEBUG_VARIABLE
00090   fprintf (stderr, "variable_free: start\n");
00091 #endif
00092   if (type == INPUT_TYPE_XML)
00093     xmlFree (variable->name);
00094   else
00095     g_free (variable->name);
00096 #if DEBUG_VARIABLE
00097   fprintf (stderr, "variable_free: end\n");
00098 #endif
00099 }
```

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 67 of file variable.c.

```
00068 {
00069 #if DEBUG_VARIABLE
00070   fprintf (stderr, "variable_new: start\n");
00071 #endif
00072   variable->name = NULL;
00073 #if DEBUG_VARIABLE
00074   fprintf (stderr, "variable_new: end\n");
00075 #endif
00076 }
```

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.

Parameters

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

Returns

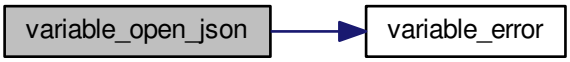
1 on success, 0 on error.

Definition at line 303 of file variable.c.

```
00305 {
00306   JsonObject *object;
00307   const char *label;
00308   int error_code;
00309 #if DEBUG_VARIABLE
00310   fprintf (stderr, "variable_open_json: start\n");
00311 #endif
00312   object = json_node_get_object (node);
00313   label = json_object_get_string_member (object, LABEL_NAME);
00314   if (!label)
00315     {
00316       variable_error (variable, _("no name"));
00317       goto exit_on_error;
00318     }
00319   variable->name = g_strdup (label);
00320   if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322       variable->rangemin
00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324       if (error_code)
00325         {
00326           variable_error (variable, _("bad minimum"));
00327           goto exit_on_error;
00328         }
00329       variable->rangeminabs
00330         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00331                                               -G_MAXDOUBLE, &error_code);
00332       if (error_code)
00333         {
00334           variable_error (variable, _("bad absolute minimum"));
00335           goto exit_on_error;
00336         }
00337       if (variable->rangemin < variable->rangeminabs)
00338         {
00339           variable_error (variable, _("minimum range not allowed"));
00340           goto exit_on_error;
00341         }
00342     }
00343   else
00344     {
00345       variable_error (variable, _("no minimum range"));
00346       goto exit_on_error;
00347     }
00348   if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350       variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352       if (error_code)
00353         {
00354           variable_error (variable, _("bad maximum"));
00355           goto exit_on_error;
00356         }
00357       variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00359                                               G_MAXDOUBLE, &error_code);
00360       if (error_code)
00361         {
00362           variable_error (variable, _("bad absolute maximum"));
00363           goto exit_on_error;
00364         }
00365       if (variable->rangemax > variable->rangemaxabs)
00366         {
```

```
00367           variable_error (variable, _("maximum range not allowed"));
00368           goto exit_on_error;
00369         }
00370       if (variable->rangemax < variable->rangemin)
00371         {
00372           variable_error (variable, _("bad range"));
00373           goto exit_on_error;
00374         }
00375     }
00376   else
00377     {
00378       variable_error (variable, _("no maximum range"));
00379       goto exit_on_error;
00380     }
00381   variable->precision
00382     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00383                                          DEFAULT_PRECISION, &error_code);
00384   if (error_code || variable->precision >= NPRECISIONS)
00385     {
00386       variable_error (variable, _("bad precision"));
00387       goto exit_on_error;
00388     }
00389   if (algorithm == ALGORITHM_SWEEP)
00390     {
00391       if (json_object_get_member (object, LABEL_NSWEEPS))
00392         {
00393           variable->nsweeps
00394             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00395           if (error_code || !variable->nsweeps)
00396             {
00397               variable_error (variable, _("bad sweeps"));
00398               goto exit_on_error;
00399             }
00400         }
00401       else
00402         {
00403           variable_error (variable, _("no sweeps number"));
00404           goto exit_on_error;
00405         }
00406 #if DEBUG_VARIABLE
00407       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00408 #endif
00409     }
00410   if (algorithm == ALGORITHM_GENETIC)
00411     {
00412       // Obtaining bits representing each variable
00413       if (json_object_get_member (object, LABEL_NBITS))
00414         {
00415           variable->nbits
00416             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00417           if (error_code || !variable->nbits)
00418             {
00419               variable_error (variable, _("invalid bits number"));
00420               goto exit_on_error;
00421             }
00422         }
00423       else
00424         {
00425           variable_error (variable, _("no bits number"));
00426           goto exit_on_error;
00427         }
00428     }
00429   else if (nsteps)
00430     {
00431       variable->step =
00432         json_object_get_float (object, LABEL_STEP, &error_code);
00433       if (error_code || variable->step < 0.)
00434         {
00435           variable_error (variable, _("bad step size"));
00436           goto exit_on_error;
00437         }
00438     }
00439
00440 #if DEBUG_VARIABLE
00441   fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443   return 1;
00444 exit_on_error:
00445   variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447   fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449   return 0;
00450 }
```

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.

Parameters

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

Returns

1 on success, 0 on error.

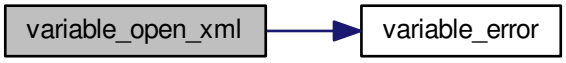
Definition at line 136 of file variable.c.

```
00138 {
00139   int error_code;
00140 
00141 #if DEBUG_VARIABLE
00142   fprintf (stderr, "variable_open_xml: start\n");
00143 #endif
00144 
00145   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146   if (!variable->name)
00147     {
00148       variable_error (variable, _("no name"));
00149       goto exit_on_error;
00150     }
00151   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153       variable->rangemin
00154         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00155                               &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, _("bad minimum"));
00159           goto exit_on_error;
```

```
00160         }
00161       variable->rangeminabs = xml_node_get_float_with_default
00162         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163          &error_code);
00164       if (error_code)
00165         {
00166           variable_error (variable, _("bad absolute minimum"));
00167           goto exit_on_error;
00168         }
00169       if (variable->rangemin < variable->rangeminabs)
00170         {
00171           variable_error (variable, _("minimum range not allowed"));
00172           goto exit_on_error;
00173         }
00174     }
00175   else
00176     {
00177       variable_error (variable, _("no minimum range"));
00178       goto exit_on_error;
00179     }
00180   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182       variable->rangemax
00183         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00184                               &error_code);
00185       if (error_code)
00186         {
00187           variable_error (variable, _("bad maximum"));
00188           goto exit_on_error;
00189         }
00190       variable->rangemaxabs = xml_node_get_float_with_default
00191         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192          &error_code);
00193       if (error_code)
00194         {
00195           variable_error (variable, _("bad absolute maximum"));
00196           goto exit_on_error;
00197         }
00198       if (variable->rangemax > variable->rangemaxabs)
00199         {
00200           variable_error (variable, _("maximum range not allowed"));
00201           goto exit_on_error;
00202         }
00203       if (variable->rangemax < variable->rangemin)
00204         {
00205           variable_error (variable, _("bad range"));
00206           goto exit_on_error;
00207         }
00208     }
00209   else
00210     {
00211       variable_error (variable, _("no maximum range"));
00212       goto exit_on_error;
00213     }
00214   variable->precision
00215     = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_PRECISION,
00216                                       DEFAULT_PRECISION, &error_code);
00217   if (error_code || variable->precision >= NPRECISIONS)
00218     {
00219       variable_error (variable, _("bad precision"));
00220       goto exit_on_error;
00221     }
00222   if (algorithm == ALGORITHM_SWEEP)
00223     {
00224       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225         {
00226           variable->nsweeps
00227             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00228                                  &error_code);
00229           if (error_code || !variable->nsweeps)
00230             {
00231               variable_error (variable, _("bad sweeps"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           variable_error (variable, _("no sweeps number"));
00238           goto exit_on_error;
00239         }
00240 #if DEBUG_VARIABLE
00241       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243     }
00244   if (algorithm == ALGORITHM_GENETIC)
00245     {
```

```
00246       // Obtaining bits representing each variable
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248         {
00249           variable->nbits
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00251                                  &error_code);
00252           if (error_code || !variable->nbits)
00253             {
00254               variable_error (variable, _("invalid bits number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           variable_error (variable, _("no bits number"));
00261           goto exit_on_error;
00262         }
00263     }
00264   else if (nsteps)
00265     {
00266       variable->step
00267         =
00268         xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269       if (error_code || variable->step < 0.)
00270         {
00271           variable_error (variable, _("bad step size"));
00272           goto exit_on_error;
00273         }
00274     }
00275
00276 #if DEBUG_VARIABLE
00277   fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279   return 1;
00280 exit_on_error:
00281   variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283   fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285   return 0;
00286 }
```

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 2012-2016, 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
00066 void
00067 variable_new (Variable * variable)
00068 {
00069 #if DEBUG_VARIABLE
```

```
00070   fprintf (stderr, "variable_new: start\n");
00071 #endif
00072   variable->name = NULL;
00073 #if DEBUG_VARIABLE
00074   fprintf (stderr, "variable_new: end\n");
00075 #endif
00076 }
00077
00086 void
00087 variable_free (Variable * variable, unsigned int type)
00088 {
00089 #if DEBUG_VARIABLE
00090   fprintf (stderr, "variable_free: start\n");
00091 #endif
00092   if (type == INPUT_TYPE_XML)
00093     xmlFree (variable->name);
00094   else
00095     g_free (variable->name);
00096 #if DEBUG_VARIABLE
00097   fprintf (stderr, "variable_free: end\n");
00098 #endif
00099 }
00100
00109 void
00110 variable_error (Variable * variable, char *message)
00111 {
00112   char buffer[64];
00113   if (!variable->name)
00114     snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name,
00117               message);
00118   error_message = g_strdup (buffer);
00119 }
00120
00135 int
00136 variable_open_xml (Variable * variable, xmlNode * node,
00137                    unsigned int algorithm, unsigned int nsteps)
00138 {
00139   int error_code;
00140
00141 #if DEBUG_VARIABLE
00142   fprintf (stderr, "variable_open_xml: start\n");
00143 #endif
00144
00145   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146   if (!variable->name)
00147     {
00148       variable_error (variable, _("no name"));
00149       goto exit_on_error;
00150     }
00151   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153       variable->rangemin
00154         = xml_node_get_float (node, (const xmlChar *) LABEL_MINIMUM,
00155                               &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, _("bad minimum"));
00159           goto exit_on_error;
00160         }
00161       variable->rangeminabs = xml_node_get_float_with_default
00162         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163          &error_code);
00164       if (error_code)
00165         {
00166           variable_error (variable, _("bad absolute minimum"));
00167           goto exit_on_error;
00168         }
00169       if (variable->rangemin < variable->rangeminabs)
00170         {
00171           variable_error (variable, _("minimum range not allowed"));
00172           goto exit_on_error;
00173         }
00174     }
00175   else
00176     {
00177       variable_error (variable, _("no minimum range"));
00178       goto exit_on_error;
00179     }
00180   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182       variable->rangemax
00183         = xml_node_get_float (node, (const xmlChar *) LABEL_MAXIMUM,
00184                               &error_code);
00185       if (error_code)
00186         {
```

```
00187           variable_error (variable, _("bad maximum"));
00188           goto exit_on_error;
00189         }
00190       variable->rangemaxabs = xml_node_get_float_with_default
00191         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192          &error_code);
00193       if (error_code)
00194         {
00195           variable_error (variable, _("bad absolute maximum"));
00196           goto exit_on_error;
00197         }
00198       if (variable->rangemax > variable->rangemaxabs)
00199         {
00200           variable_error (variable, _("maximum range not allowed"));
00201           goto exit_on_error;
00202         }
00203       if (variable->rangemax < variable->rangemin)
00204         {
00205           variable_error (variable, _("bad range"));
00206           goto exit_on_error;
00207         }
00208     }
00209   else
00210     {
00211       variable_error (variable, _("no maximum range"));
00212       goto exit_on_error;
00213     }
00214   variable->precision
00215     = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_PRECISION,
00216                                       DEFAULT_PRECISION, &error_code);
00217   if (error_code || variable->precision >= NPRECISIONS)
00218     {
00219       variable_error (variable, _("bad precision"));
00220       goto exit_on_error;
00221     }
00222   if (algorithm == ALGORITHM_SWEEP)
00223     {
00224       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225         {
00226           variable->nsweeps
00227             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00228                                  &error_code);
00229           if (error_code || !variable->nsweeps)
00230             {
00231               variable_error (variable, _("bad sweeps"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           variable_error (variable, _("no sweeps number"));
00238           goto exit_on_error;
00239         }
00240 #if DEBUG_VARIABLE
00241       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243     }
00244   if (algorithm == ALGORITHM_GENETIC)
00245     {
00246       // Obtaining bits representing each variable
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248         {
00249           variable->nbits
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00251                                  &error_code);
00252           if (error_code || !variable->nbits)
00253             {
00254               variable_error (variable, _("invalid bits number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           variable_error (variable, _("no bits number"));
00261           goto exit_on_error;
00262         }
00263     }
00264   else if (nsteps)
00265     {
00266       variable->step
00267         =
00268         xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269       if (error_code || variable->step < 0.)
00270         {
00271           variable_error (variable, _("bad step size"));
00272           goto exit_on_error;
```

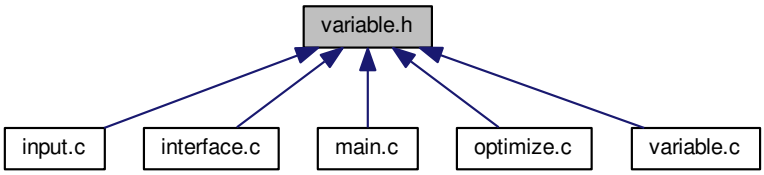
```
00273         }
00274     }
00275
00276 #if DEBUG_VARIABLE
00277   fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279   return 1;
00280 exit_on_error:
00281   variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283   fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285   return 0;
00286 }
00287
00302 int
00303 variable_open_json (Variable * variable, JsonNode * node,
00304                     unsigned int algorithm, unsigned int nsteps)
00305 {
00306   JsonObject *object;
00307   const char *label;
00308   int error_code;
00309 #if DEBUG_VARIABLE
00310   fprintf (stderr, "variable_open_json: start\n");
00311 #endif
00312   object = json_node_get_object (node);
00313   label = json_object_get_string_member (object, LABEL_NAME);
00314   if (!label)
00315     {
00316       variable_error (variable, _("no name"));
00317       goto exit_on_error;
00318     }
00319   variable->name = g_strdup (label);
00320   if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322       variable->rangemin
00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324       if (error_code)
00325         {
00326           variable_error (variable, _("bad minimum"));
00327           goto exit_on_error;
00328         }
00329       variable->rangeminabs
00330         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00331                                               -G_MAXDOUBLE, &error_code);
00332       if (error_code)
00333         {
00334           variable_error (variable, _("bad absolute minimum"));
00335           goto exit_on_error;
00336         }
00337       if (variable->rangemin < variable->rangeminabs)
00338         {
00339           variable_error (variable, _("minimum range not allowed"));
00340           goto exit_on_error;
00341         }
00342     }
00343   else
00344     {
00345       variable_error (variable, _("no minimum range"));
00346       goto exit_on_error;
00347     }
00348   if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350       variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352       if (error_code)
00353         {
00354           variable_error (variable, _("bad maximum"));
00355           goto exit_on_error;
00356         }
00357       variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00359                                               G_MAXDOUBLE, &error_code);
00360       if (error_code)
00361         {
00362           variable_error (variable, _("bad absolute maximum"));
00363           goto exit_on_error;
00364         }
00365       if (variable->rangemax > variable->rangemaxabs)
00366         {
00367           variable_error (variable, _("maximum range not allowed"));
00368           goto exit_on_error;
00369         }
00370       if (variable->rangemax < variable->rangemin)
00371         {
```

```
00372           variable_error (variable, _("bad range"));
00373           goto exit_on_error;
00374         }
00375     }
00376   else
00377     {
00378       variable_error (variable, _("no maximum range"));
00379       goto exit_on_error;
00380     }
00381   variable->precision
00382     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00383                                          DEFAULT_PRECISION, &error_code);
00384   if (error_code || variable->precision >= NPRECISIONS)
00385     {
00386       variable_error (variable, _("bad precision"));
00387       goto exit_on_error;
00388     }
00389   if (algorithm == ALGORITHM_SWEEP)
00390     {
00391       if (json_object_get_member (object, LABEL_NSWEEPS))
00392         {
00393           variable->nsweeps
00394             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00395           if (error_code || !variable->nsweeps)
00396             {
00397               variable_error (variable, _("bad sweeps"));
00398               goto exit_on_error;
00399             }
00400         }
00401       else
00402         {
00403           variable_error (variable, _("no sweeps number"));
00404           goto exit_on_error;
00405         }
00406 #if DEBUG_VARIABLE
00407       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00408 #endif
00409     }
00410   if (algorithm == ALGORITHM_GENETIC)
00411     {
00412       // Obtaining bits representing each variable
00413       if (json_object_get_member (object, LABEL_NBITS))
00414         {
00415           variable->nbits
00416             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00417           if (error_code || !variable->nbits)
00418             {
00419               variable_error (variable, _("invalid bits number"));
00420               goto exit_on_error;
00421             }
00422         }
00423       else
00424         {
00425           variable_error (variable, _("no bits number"));
00426           goto exit_on_error;
00427         }
00428     }
00429   else if (nsteps)
00430     {
00431       variable->step =
00432         json_object_get_float (object, LABEL_STEP, &error_code);
00433       if (error_code || variable->step < 0.)
00434         {
00435           variable_error (variable, _("bad step size"));
00436           goto exit_on_error;
00437         }
00438     }
00439
00440 #if DEBUG_VARIABLE
00441   fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443   return 1;
00444 exit_on_error:
00445   variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447   fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449   return 0;
00450 }
```

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 }

  Enum to define the algorithms.

Functions

- void variable_new (Variable *variable)

  Function to create a new Variable struct.
- void variable_free (Variable *variable, unsigned int type)

  Function to free the memory of a Variable struct.
- void variable_error (Variable *variable, char *message)

  Function to print a message error opening an Variable struct.
- int variable_open_xml (Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.
- int variable_open_json (Variable *variable, JsonNode *node, unsigned int algorithm, unsigned int nsteps)

  Function to open the variable file.

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

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.

Definition at line 45 of file variable.h.

```
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
```

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 110 of file variable.c.

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

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 87 of file variable.c.

```
00088 {
00089 #if DEBUG_VARIABLE
00090   fprintf (stderr, "variable_free: start\n");
00091 #endif
00092   if (type == INPUT_TYPE_XML)
00093     xmlFree (variable->name);
00094   else
00095     g_free (variable->name);
00096 #if DEBUG_VARIABLE
00097   fprintf (stderr, "variable_free: end\n");
00098 #endif
00099 }
```

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 67 of file variable.c.

```
00068 {
00069 #if DEBUG_VARIABLE
00070   fprintf (stderr, "variable_new: start\n");
00071 #endif
00072   variable->name = NULL;
00073 #if DEBUG_VARIABLE
00074   fprintf (stderr, "variable_new: end\n");
00075 #endif
00076 }
```

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.

Parameters

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

Returns

1 on success, 0 on error.

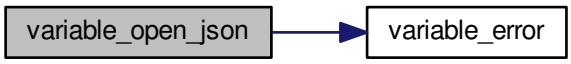
Definition at line 303 of file variable.c.

```
00305 {
00306   JsonObject *object;
00307   const char *label;
00308   int error_code;
00309 #if DEBUG_VARIABLE
00310   fprintf (stderr, "variable_open_json: start\n");
00311 #endif
00312   object = json_node_get_object (node);
00313   label = json_object_get_string_member (object, LABEL_NAME);
00314   if (!label)
00315     {
00316       variable_error (variable, _("no name"));
00317       goto exit_on_error;
00318     }
00319   variable->name = g_strdup (label);
00320   if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322       variable->rangemin
```

```
00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324       if (error_code)
00325         {
00326           variable_error (variable, _("bad minimum"));
00327           goto exit_on_error;
00328         }
00329       variable->rangeminabs
00330         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MINIMUM,
00331                                               -G_MAXDOUBLE, &error_code);
00332       if (error_code)
00333         {
00334           variable_error (variable, _("bad absolute minimum"));
00335           goto exit_on_error;
00336         }
00337       if (variable->rangemin < variable->rangeminabs)
00338         {
00339           variable_error (variable, _("minimum range not allowed"));
00340           goto exit_on_error;
00341         }
00342     }
00343   else
00344     {
00345       variable_error (variable, _("no minimum range"));
00346       goto exit_on_error;
00347     }
00348   if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350       variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352       if (error_code)
00353         {
00354           variable_error (variable, _("bad maximum"));
00355           goto exit_on_error;
00356         }
00357       variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MAXIMUM,
00359                                               G_MAXDOUBLE, &error_code);
00360       if (error_code)
00361         {
00362           variable_error (variable, _("bad absolute maximum"));
00363           goto exit_on_error;
00364         }
00365       if (variable->rangemax > variable->rangemaxabs)
00366         {
00367           variable_error (variable, _("maximum range not allowed"));
00368           goto exit_on_error;
00369         }
00370       if (variable->rangemax < variable->rangemin)
00371         {
00372           variable_error (variable, _("bad range"));
00373           goto exit_on_error;
00374         }
00375     }
00376   else
00377     {
00378       variable_error (variable, _("no maximum range"));
00379       goto exit_on_error;
00380     }
00381   variable->precision
00382     = json_object_get_uint_with_default (object,
    LABEL_PRECISION,
00383                                          DEFAULT_PRECISION, &error_code);
00384   if (error_code || variable->precision >= NPRECISIONS)
00385     {
00386       variable_error (variable, _("bad precision"));
00387       goto exit_on_error;
00388     }
00389   if (algorithm == ALGORITHM_SWEEP)
00390     {
00391       if (json_object_get_member (object, LABEL_NSWEEPS))
00392         {
00393           variable->nsweeps
00394             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00395           if (error_code || !variable->nsweeps)
00396             {
00397               variable_error (variable, _("bad sweeps"));
00398               goto exit_on_error;
00399             }
00400         }
00401       else
00402         {
00403           variable_error (variable, _("no sweeps number"));
00404           goto exit_on_error;
00405         }
00406 #if DEBUG_VARIABLE
```

```
00407       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00408 #endif
00409     }
00410   if (algorithm == ALGORITHM_GENETIC)
00411     {
00412       // Obtaining bits representing each variable
00413       if (json_object_get_member (object, LABEL_NBITS))
00414         {
00415           variable->nbits
00416             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00417           if (error_code || !variable->nbits)
00418             {
00419               variable_error (variable, _("invalid bits number"));
00420               goto exit_on_error;
00421             }
00422         }
00423       else
00424         {
00425           variable_error (variable, _("no bits number"));
00426           goto exit_on_error;
00427         }
00428     }
00429   else if (nsteps)
00430     {
00431       variable->step =
00432         json_object_get_float (object, LABEL_STEP, &error_code);
00433       if (error_code || variable->step < 0.)
00434         {
00435           variable_error (variable, _("bad step size"));
00436           goto exit_on_error;
00437         }
00438     }
00439
00440 #if DEBUG_VARIABLE
00441   fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443   return 1;
00444 exit_on_error:
00445   variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447   fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449   return 0;
00450 }
```

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.

Parameters

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

Returns

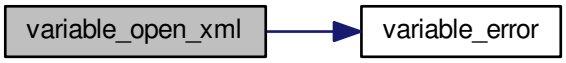
1 on success, 0 on error.

Definition at line 136 of file variable.c.

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

```
00202         }
00203       if (variable->rangemax < variable->rangemin)
00204         {
00205           variable_error (variable, _("bad range"));
00206           goto exit_on_error;
00207         }
00208     }
00209   else
00210     {
00211       variable_error (variable, _("no maximum range"));
00212       goto exit_on_error;
00213     }
00214   variable->precision
00215     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_PRECISION,
00216                                       DEFAULT_PRECISION, &error_code);
00217   if (error_code || variable->precision >= NPRECISIONS)
00218     {
00219       variable_error (variable, _("bad precision"));
00220       goto exit_on_error;
00221     }
00222   if (algorithm == ALGORITHM_SWEEP)
00223     {
00224       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225         {
00226           variable->nsweeps
00227             = xml_node_get_uint (node, (const xmlChar *) LABEL_NSWEEPS,
00228                                  &error_code);
00229           if (error_code || !variable->nsweeps)
00230             {
00231               variable_error (variable, _("bad sweeps"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           variable_error (variable, _("no sweeps number"));
00238           goto exit_on_error;
00239         }
00240 #if DEBUG_VARIABLE
00241       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243     }
00244   if (algorithm == ALGORITHM_GENETIC)
00245     {
00246       // Obtaining bits representing each variable
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248         {
00249           variable->nbits
00250             = xml_node_get_uint (node, (const xmlChar *) LABEL_NBITS,
00251                                  &error_code);
00252           if (error_code || !variable->nbits)
00253             {
00254               variable_error (variable, _("invalid bits number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           variable_error (variable, _("no bits number"));
00261           goto exit_on_error;
00262         }
00263     }
00264   else if (nsteps)
00265     {
00266       variable->step
00267         =
00268         xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269       if (error_code || variable->step < 0.)
00270         {
00271           variable_error (variable, _("bad step size"));
00272           goto exit_on_error;
00273         }
00274     }
00275
00276 #if DEBUG_VARIABLE
00277   fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279   return 1;
00280 exit_on_error:
00281   variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283   fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285   return 0;
00286 }
```

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 2012-2016, 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
00045 enum Algorithm
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
00051
00056 typedef struct
00057 {
00058   char *name;
00059   double rangemin;
00060   double rangemax;
00061   double rangeminabs;
00062   double rangemaxabs;
00063   double step;
00064   unsigned int precision;
00065   unsigned int nsweeps;
00066   unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable, unsigned int type);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open_xml (Variable * variable, xmlNode * node,
```

```
00077                         unsigned int algorithm, unsigned int nsteps);
00078 int variable_open_json (Variable * variable, JsonNode * node,
00079                          unsigned int algorithm, unsigned int nsteps);
00080
00081 #endif
```

Index