# MPCOTool

3.4.2

Generated by Doxygen 1.8.11

# 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 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:

## 4.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2017, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00039 #ifndef CONFIG__H
00040 #define CONFIG__H 1
00041
00042 // Gettext simplification
00043 #define _(string) (gettext(string))
00044
00045 // Array sizes
00046
00047 #define MAX_NINPUTS 8
00048 #define NALGORITHMS 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 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 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 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 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 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 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 2012-2017, 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 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 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 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 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 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 2012-2017, 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 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 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 949 of file input.c.

```
00950 {
```

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

Here is the call graph for this function:

#### 4.7.2.3 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 560 of file input.c.

```
00561 {
00562   JsonNode *node, *child;
```

```
00563   JsonObject *object;
00564   JsonArray *array;
00565   const char *buffer;
00566   int error_code;
00567   unsigned int i, n;
00568
00569 #if DEBUG_INPUT
00570   fprintf (stderr, "input_open_json: start\n");
00571 #endif
00572
00573   // Resetting input data
00574   input->type = INPUT_TYPE_JSON;
00575
00576   // Getting the root node
00577 #if DEBUG_INPUT
00578   fprintf (stderr, "input_open_json: getting the root node\n");
00579 #endif
00580   node = json_parser_get_root (parser);
00581   object = json_node_get_object (node);
00582
00583   // Getting result and variables file names
00584   if (!input->result)
00585     {
00586       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00587       if (!buffer)
00588         buffer = result_name;
00589       input->result = g_strdup (buffer);
00590     }
00591   else
00592     input->result = g_strdup (result_name);
00593   if (!input->variables)
00594     {
00595       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00596       if (!buffer)
00597         buffer = variables_name;
00598       input->variables = g_strdup (buffer);
00599     }
00600   else
00601     input->variables = g_strdup (variables_name);
00602
00603   // Opening simulator program name
00604   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00605   if (!buffer)
00606     {
00607       input_error (_("Bad simulator program"));
00608       goto exit_on_error;
00609     }
00610   input->simulator = g_strdup (buffer);
00611
00612   // Opening evaluator program name
00613   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00614   if (buffer)
00615     input->evaluator = g_strdup (buffer);
00616
00617   // Obtaining pseudo-random numbers generator seed
00618   input->seed
00619     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00620                                          DEFAULT_RANDOM_SEED, &error_code);
00621   if (error_code)
00622     {
00623       input_error (_("Bad pseudo-random numbers generator seed"));
00624       goto exit_on_error;
00625     }
00626
00627   // Opening algorithm
00628   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00629   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00630     {
00631       input->algorithm = ALGORITHM_MONTE_CARLO;
00632
00633       // Obtaining simulations number
00634       input->nsimulations
00635         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00636       if (error_code)
00637         {
00638           input_error (_("Bad simulations number"));
00639           goto exit_on_error;
00640         }
00641     }
00642   else if (!strcmp (buffer, LABEL_SWEEP))
00643     input->algorithm = ALGORITHM_SWEEP;
00644   else if (!strcmp (buffer, LABEL_GENETIC))
00645     {
00646       input->algorithm = ALGORITHM_GENETIC;
00647
```

```
00648       // Obtaining population
00649       if (json_object_get_member (object, LABEL_NPOPULATION))
00650         {
00651           input->nsimulations
00652             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00653           if (error_code || input->nsimulations < 3)
00654             {
00655               input_error (_("Invalid population number"));
00656               goto exit_on_error;
00657             }
00658         }
00659       else
00660         {
00661           input_error (_("No population number"));
00662           goto exit_on_error;
00663         }
00664
00665       // Obtaining generations
00666       if (json_object_get_member (object, LABEL_NGENERATIONS))
00667         {
00668           input->niterations
00669             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00670           if (error_code || !input->niterations)
00671             {
00672               input_error (_("Invalid generations number"));
00673               goto exit_on_error;
00674             }
00675         }
00676       else
00677         {
00678           input_error (_("No generations number"));
00679           goto exit_on_error;
00680         }
00681
00682       // Obtaining mutation probability
00683       if (json_object_get_member (object, LABEL_MUTATION))
00684         {
00685           input->mutation_ratio
00686             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00687           if (error_code || input->mutation_ratio < 0.
00688               || input->mutation_ratio >= 1.)
00689             {
00690               input_error (_("Invalid mutation probability"));
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         {
00696           input_error (_("No mutation probability"));
00697           goto exit_on_error;
00698         }
00699
00700       // Obtaining reproduction probability
00701       if (json_object_get_member (object, LABEL_REPRODUCTION))
00702         {
00703           input->reproduction_ratio
00704             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00705           if (error_code || input->reproduction_ratio < 0.
00706               || input->reproduction_ratio >= 1.0)
00707             {
00708               input_error (_("Invalid reproduction probability"));
00709               goto exit_on_error;
00710             }
00711         }
00712       else
00713         {
00714           input_error (_("No reproduction probability"));
00715           goto exit_on_error;
00716         }
00717
00718       // Obtaining adaptation probability
00719       if (json_object_get_member (object, LABEL_ADAPTATION))
00720         {
00721           input->adaptation_ratio
00722             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00723           if (error_code || input->adaptation_ratio < 0.
00724               || input->adaptation_ratio >= 1.)
00725             {
00726               input_error (_("Invalid adaptation probability"));
00727               goto exit_on_error;
00728             }
00729         }
```

```
00730       else
00731         {
00732           input_error (_("No adaptation probability"));
00733           goto exit_on_error;
00734         }
00735
00736       // Checking survivals
00737       i = input->mutation_ratio * input->nsimulations;
00738       i += input->reproduction_ratio * input->
      nsimulations;
00739       i += input->adaptation_ratio * input->
      nsimulations;
00740       if (i > input->nsimulations - 2)
00741         {
00742           input_error
00743             (_("No enough survival entities to reproduce the population"));
00744           goto exit_on_error;
00745         }
00746     }
00747   else
00748     {
00749       input_error (_("Unknown algorithm"));
00750       goto exit_on_error;
00751     }
00752
00753   if (input->algorithm == ALGORITHM_MONTE_CARLO
00754       || input->algorithm == ALGORITHM_SWEEP)
00755     {
00756
00757       // Obtaining iterations number
00758       input->niterations
00759         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00760       if (error_code == 1)
00761         input->niterations = 1;
00762       else if (error_code)
00763         {
00764           input_error (_("Bad iterations number"));
00765           goto exit_on_error;
00766         }
00767
00768       // Obtaining best number
00769       input->nbest
00770         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00771                                              &error_code);
00772       if (error_code || !input->nbest)
00773         {
00774           input_error (_("Invalid best number"));
00775           goto exit_on_error;
00776         }
00777
00778       // Obtaining tolerance
00779       input->tolerance
00780         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00781                                             &error_code);
00782       if (error_code || input->tolerance < 0.)
00783         {
00784           input_error (_("Invalid tolerance"));
00785           goto exit_on_error;
00786         }
00787
00788       // Getting direction search method parameters
00789       if (json_object_get_member (object, LABEL_NSTEPS))
00790         {
00791           input->nsteps
00792             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00793           if (error_code)
00794             {
00795               input_error (_("Invalid steps number"));
00796               goto exit_on_error;
00797             }
00798           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00799           if (!strcmp (buffer, LABEL_COORDINATES))
00800             input->direction = DIRECTION_METHOD_COORDINATES;
00801           else if (!strcmp (buffer, LABEL_RANDOM))
00802             {
00803               input->direction = DIRECTION_METHOD_RANDOM;
00804               input->nestimates
00805                 = json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00806               if (error_code || !input->nestimates)
00807                 {
00808                   input_error (_("Invalid estimates number"));
00809                   goto exit_on_error;
00810                 }
```

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

```
00888       ++input->nvariables;
00889     }
00890   if (!input->nvariables)
00891     {
00892       input_error (_("No optimization variables"));
00893       goto exit_on_error;
00894     }
00895
00896   // Obtaining the error norm
00897   if (json_object_get_member (object, LABEL_NORM))
00898     {
00899       buffer = json_object_get_string_member (object, LABEL_NORM);
00900       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00901         input->norm = ERROR_NORM_EUCLIDIAN;
00902       else if (!strcmp (buffer, LABEL_MAXIMUM))
00903         input->norm = ERROR_NORM_MAXIMUM;
00904       else if (!strcmp (buffer, LABEL_P))
00905         {
00906           input->norm = ERROR_NORM_P;
00907           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00908           if (!error_code)
00909             {
00910               input_error (_("Bad P parameter"));
00911               goto exit_on_error;
00912             }
00913         }
00914       else if (!strcmp (buffer, LABEL_TAXICAB))
00915         input->norm = ERROR_NORM_TAXICAB;
00916       else
00917         {
00918           input_error (_("Unknown error norm"));
00919           goto exit_on_error;
00920         }
00921     }
00922   else
00923     input->norm = ERROR_NORM_EUCLIDIAN;
00924
00925   // Closing the JSON document
00926   g_object_unref (parser);
00927
00928 #if DEBUG_INPUT
00929   fprintf (stderr, "input_open_json: end\n");
00930 #endif
00931   return 1;
00932
00933 exit_on_error:
00934   g_object_unref (parser);
00935 #if DEBUG_INPUT
00936   fprintf (stderr, "input_open_json: end\n");
00937 #endif
00938   return 0;
00939 }
```

Here is the call graph for this function:

#### 4.7.2.4 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, &error_code);
00421           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00422             {
00423               input_error (_("Invalid relaxation parameter"));
00424               goto exit_on_error;
00425             }
00426         }
00427       else
00428         input->nsteps = 0;
00429     }
00430   // Obtaining the threshold
00431   input->threshold =
00432     xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_THRESHOLD,
00433                                      0., &error_code);
00434   if (error_code)
00435     {
00436       input_error (_("Invalid threshold"));
00437       goto exit_on_error;
00438     }
00439
00440   // Reading the experimental data
00441   for (child = node->children; child; child = child->next)
00442     {
00443       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00444         break;
00445 #if DEBUG_INPUT
00446       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00447                input->nexperiments);
00448 #endif
00449       input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                    (1 + input->nexperiments) * sizeof (
    Experiment));
00452       if (!input->nexperiments)
00453         {
00454           if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456         }
00457       else
00458         {
00459           if (!experiment_open_xml (input->experiment +
    input->nexperiments,
00460                                     child, input->experiment->
    ninputs))
00461             goto exit_on_error;
00462         }
00463       ++input->nexperiments;
00464 #if DEBUG_INPUT
00465       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00466                input->nexperiments);
00467 #endif
00468     }
00469   if (!input->nexperiments)
```

```
00470     {
00471       input_error (_("No optimization experiments"));
00472       goto exit_on_error;
00473     }
00474   buffer = NULL;
00475
00476   // Reading the variables data
00477   for (; child; child = child->next)
00478     {
00479 #if DEBUG_INPUT
00480       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00481 #endif
00482       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00483         {
00484           snprintf (buffer2, 64, "%s %u: %s",
00485                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00486           input_error (buffer2);
00487           goto exit_on_error;
00488         }
00489       input->variable = (Variable *)
00490         g_realloc (input->variable,
00491                    (1 + input->nvariables) * sizeof (Variable));
00492       if (!variable_open_xml (input->variable +
    input->nvariables, child,
00493                               input->algorithm, input->nsteps))
00494         goto exit_on_error;
00495       ++input->nvariables;
00496     }
00497   if (!input->nvariables)
00498     {
00499       input_error (_("No optimization variables"));
00500       goto exit_on_error;
00501     }
00502   buffer = NULL;
00503
00504   // Obtaining the error norm
00505   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00506     {
00507       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00508       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00509         input->norm = ERROR_NORM_EUCLIDIAN;
00510       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00511         input->norm = ERROR_NORM_MAXIMUM;
00512       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00513         {
00514           input->norm = ERROR_NORM_P;
00515           input->p
00516             = xml_node_get_float (node, (const xmlChar *)
    LABEL_P, &error_code);
00517           if (!error_code)
00518             {
00519               input_error (_("Bad P parameter"));
00520               goto exit_on_error;
00521             }
00522         }
00523       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00524         input->norm = ERROR_NORM_TAXICAB;
00525       else
00526         {
00527           input_error (_("Unknown error norm"));
00528           goto exit_on_error;
00529         }
00530       xmlFree (buffer);
00531     }
00532   else
00533     input->norm = ERROR_NORM_EUCLIDIAN;
00534
00535   // Closing the XML document
00536   xmlFreeDoc (doc);
00537
00538 #if DEBUG_INPUT
00539   fprintf (stderr, "input_open_xml: end\n");
00540 #endif
00541   return 1;
00542
00543 exit_on_error:
00544   xmlFree (buffer);
00545   xmlFreeDoc (doc);
00546 #if DEBUG_INPUT
00547   fprintf (stderr, "input_open_xml: end\n");
00548 #endif
00549   return 0;
00550 }
```

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 2012-2017, 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, &error_code);
00421           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00422             {
00423               input_error (_("Invalid relaxation parameter"));
00424               goto exit_on_error;
00425             }
00426         }
00427       else
00428         input->nsteps = 0;
00429     }
00430   // Obtaining the threshold
00431   input->threshold =
00432     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00433                                      0., &error_code);
00434   if (error_code)
00435     {
00436       input_error (_("Invalid threshold"));
00437       goto exit_on_error;
00438     }
00439
00440   // Reading the experimental data
00441   for (child = node->children; child; child = child->next)
00442     {
00443       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00444         break;
00445 #if DEBUG_INPUT
```

```
00446       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00447                input->nexperiments);
00448 #endif
00449       input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                    (1 + input->nexperiments) * sizeof (Experiment));
00452       if (!input->nexperiments)
00453         {
00454           if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456         }
00457       else
00458         {
00459           if (!experiment_open_xml (input->experiment + input->
      nexperiments,
00460                                     child, input->experiment->ninputs))
00461             goto exit_on_error;
00462         }
00463       ++input->nexperiments;
00464 #if DEBUG_INPUT
00465       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00466                input->nexperiments);
00467 #endif
00468     }
00469   if (!input->nexperiments)
00470     {
00471       input_error (_("No optimization experiments"));
00472       goto exit_on_error;
00473     }
00474   buffer = NULL;
00475
00476   // Reading the variables data
00477   for (; child; child = child->next)
00478     {
00479 #if DEBUG_INPUT
00480       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00481 #endif
00482       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00483         {
00484           snprintf (buffer2, 64, "%s %u: %s",
00485                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00486           input_error (buffer2);
00487           goto exit_on_error;
00488         }
00489       input->variable = (Variable *)
00490         g_realloc (input->variable,
00491                    (1 + input->nvariables) * sizeof (Variable));
00492       if (!variable_open_xml (input->variable + input->
      nvariables, child,
00493                               input->algorithm, input->nsteps))
00494         goto exit_on_error;
00495       ++input->nvariables;
00496     }
00497   if (!input->nvariables)
00498     {
00499       input_error (_("No optimization variables"));
00500       goto exit_on_error;
00501     }
00502   buffer = NULL;
00503
00504   // Obtaining the error norm
00505   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00506     {
00507       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00508       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00509         input->norm = ERROR_NORM_EUCLIDIAN;
00510       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00511         input->norm = ERROR_NORM_MAXIMUM;
00512       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00513         {
00514           input->norm = ERROR_NORM_P;
00515           input->p
00516             = xml_node_get_float (node, (const xmlChar *)
      LABEL_P, &error_code);
00517           if (!error_code)
00518             {
00519               input_error (_("Bad P parameter"));
00520               goto exit_on_error;
00521             }
00522         }
00523       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00524         input->norm = ERROR_NORM_TAXICAB;
00525       else
00526         {
00527           input_error (_("Unknown error norm"));
00528           goto exit_on_error;
00529         }
```

```
00530       xmlFree (buffer);
00531     }
00532   else
00533     input->norm = ERROR_NORM_EUCLIDIAN;
00534
00535   // Closing the XML document
00536   xmlFreeDoc (doc);
00537
00538 #if DEBUG_INPUT
00539   fprintf (stderr, "input_open_xml: end\n");
00540 #endif
00541   return 1;
00542
00543 exit_on_error:
00544   xmlFree (buffer);
00545   xmlFreeDoc (doc);
00546 #if DEBUG_INPUT
00547   fprintf (stderr, "input_open_xml: end\n");
00548 #endif
00549   return 0;
00550 }
00551
00559 int
00560 input_open_json (JsonParser * parser)
00561 {
00562   JsonNode *node, *child;
00563   JsonObject *object;
00564   JsonArray *array;
00565   const char *buffer;
00566   int error_code;
00567   unsigned int i, n;
00568
00569 #if DEBUG_INPUT
00570   fprintf (stderr, "input_open_json: start\n");
00571 #endif
00572
00573   // Resetting input data
00574   input->type = INPUT_TYPE_JSON;
00575
00576   // Getting the root node
00577 #if DEBUG_INPUT
00578   fprintf (stderr, "input_open_json: getting the root node\n");
00579 #endif
00580   node = json_parser_get_root (parser);
00581   object = json_node_get_object (node);
00582
00583   // Getting result and variables file names
00584   if (!input->result)
00585     {
00586       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00587       if (!buffer)
00588         buffer = result_name;
00589       input->result = g_strdup (buffer);
00590     }
00591   else
00592     input->result = g_strdup (result_name);
00593   if (!input->variables)
00594     {
00595       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00596       if (!buffer)
00597         buffer = variables_name;
00598       input->variables = g_strdup (buffer);
00599     }
00600   else
00601     input->variables = g_strdup (variables_name);
00602
00603   // Opening simulator program name
00604   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00605   if (!buffer)
00606     {
00607       input_error (_("Bad simulator program"));
00608       goto exit_on_error;
00609     }
00610   input->simulator = g_strdup (buffer);
00611
00612   // Opening evaluator program name
00613   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00614   if (buffer)
00615     input->evaluator = g_strdup (buffer);
00616
00617   // Obtaining pseudo-random numbers generator seed
00618   input->seed
00619     = json_object_get_uint_with_default (object,
     LABEL_SEED,
00620                                          DEFAULT_RANDOM_SEED, &error_code);
00621   if (error_code)
00622     {
```

```
00623       input_error (_("Bad pseudo-random numbers generator seed"));
00624       goto exit_on_error;
00625     }
00626
00627   // Opening algorithm
00628   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00629   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00630     {
00631       input->algorithm = ALGORITHM_MONTE_CARLO;
00632
00633       // Obtaining simulations number
00634       input->nsimulations
00635         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00636       if (error_code)
00637         {
00638           input_error (_("Bad simulations number"));
00639           goto exit_on_error;
00640         }
00641     }
00642   else if (!strcmp (buffer, LABEL_SWEEP))
00643     input->algorithm = ALGORITHM_SWEEP;
00644   else if (!strcmp (buffer, LABEL_GENETIC))
00645     {
00646       input->algorithm = ALGORITHM_GENETIC;
00647
00648       // Obtaining population
00649       if (json_object_get_member (object, LABEL_NPOPULATION))
00650         {
00651           input->nsimulations
00652             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00653           if (error_code || input->nsimulations < 3)
00654             {
00655               input_error (_("Invalid population number"));
00656               goto exit_on_error;
00657             }
00658         }
00659       else
00660         {
00661           input_error (_("No population number"));
00662           goto exit_on_error;
00663         }
00664
00665       // Obtaining generations
00666       if (json_object_get_member (object, LABEL_NGENERATIONS))
00667         {
00668           input->niterations
00669             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00670           if (error_code || !input->niterations)
00671             {
00672               input_error (_("Invalid generations number"));
00673               goto exit_on_error;
00674             }
00675         }
00676       else
00677         {
00678           input_error (_("No generations number"));
00679           goto exit_on_error;
00680         }
00681
00682       // Obtaining mutation probability
00683       if (json_object_get_member (object, LABEL_MUTATION))
00684         {
00685           input->mutation_ratio
00686             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00687           if (error_code || input->mutation_ratio < 0.
00688               || input->mutation_ratio >= 1.)
00689             {
00690               input_error (_("Invalid mutation probability"));
00691               goto exit_on_error;
00692             }
00693         }
00694       else
00695         {
00696           input_error (_("No mutation probability"));
00697           goto exit_on_error;
00698         }
00699
00700       // Obtaining reproduction probability
00701       if (json_object_get_member (object, LABEL_REPRODUCTION))
00702         {
00703           input->reproduction_ratio
00704             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
```

```
00705           if (error_code || input->reproduction_ratio < 0.
00706               || input->reproduction_ratio >= 1.0)
00707             {
00708               input_error (_("Invalid reproduction probability"));
00709               goto exit_on_error;
00710             }
00711         }
00712       else
00713         {
00714           input_error (_("No reproduction probability"));
00715           goto exit_on_error;
00716         }
00717
00718       // Obtaining adaptation probability
00719       if (json_object_get_member (object, LABEL_ADAPTATION))
00720         {
00721           input->adaptation_ratio
00722             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00723           if (error_code || input->adaptation_ratio < 0.
00724               || input->adaptation_ratio >= 1.)
00725             {
00726               input_error (_("Invalid adaptation probability"));
00727               goto exit_on_error;
00728             }
00729         }
00730       else
00731         {
00732           input_error (_("No adaptation probability"));
00733           goto exit_on_error;
00734         }
00735
00736       // Checking survivals
00737       i = input->mutation_ratio * input->nsimulations;
00738       i += input->reproduction_ratio * input->nsimulations;
00739       i += input->adaptation_ratio * input->nsimulations;
00740       if (i > input->nsimulations - 2)
00741         {
00742           input_error
00743             (_("No enough survival entities to reproduce the population"));
00744           goto exit_on_error;
00745         }
00746     }
00747   else
00748     {
00749       input_error (_("Unknown algorithm"));
00750       goto exit_on_error;
00751     }
00752
00753   if (input->algorithm == ALGORITHM_MONTE_CARLO
00754       || input->algorithm == ALGORITHM_SWEEP)
00755     {
00756
00757       // Obtaining iterations number
00758       input->niterations
00759         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00760       if (error_code == 1)
00761         input->niterations = 1;
00762       else if (error_code)
00763         {
00764           input_error (_("Bad iterations number"));
00765           goto exit_on_error;
00766         }
00767
00768       // Obtaining best number
00769       input->nbest
00770         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00771                                              &error_code);
00772       if (error_code || !input->nbest)
00773         {
00774           input_error (_("Invalid best number"));
00775           goto exit_on_error;
00776         }
00777
00778       // Obtaining tolerance
00779       input->tolerance
00780         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00781                                               &error_code);
00782       if (error_code || input->tolerance < 0.)
00783         {
00784           input_error (_("Invalid tolerance"));
00785           goto exit_on_error;
00786         }
00787
```

```
00788       // Getting direction search method parameters
00789       if (json_object_get_member (object, LABEL_NSTEPS))
00790         {
00791           input->nsteps
00792             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00793           if (error_code)
00794             {
00795               input_error (_("Invalid steps number"));
00796               goto exit_on_error;
00797             }
00798           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00799           if (!strcmp (buffer, LABEL_COORDINATES))
00800             input->direction = DIRECTION_METHOD_COORDINATES;
00801           else if (!strcmp (buffer, LABEL_RANDOM))
00802             {
00803               input->direction = DIRECTION_METHOD_RANDOM;
00804               input->nestimates
00805                 = json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00806               if (error_code || !input->nestimates)
00807                 {
00808                   input_error (_("Invalid estimates number"));
00809                   goto exit_on_error;
00810                 }
00811             }
00812           else
00813             {
00814               input_error
00815                 (_("Unknown method to estimate the direction search"));
00816               goto exit_on_error;
00817             }
00818           input->relaxation
00819             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00820                                                   DEFAULT_RELAXATION,
00821                                                   &error_code);
00822           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00823             {
00824               input_error (_("Invalid relaxation parameter"));
00825               goto exit_on_error;
00826             }
00827         }
00828       else
00829         input->nsteps = 0;
00830     }
00831   // Obtaining the threshold
00832   input->threshold
00833     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00834                                           &error_code);
00835   if (error_code)
00836     {
00837       input_error (_("Invalid threshold"));
00838       goto exit_on_error;
00839     }
00840
00841   // Reading the experimental data
00842   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00843   n = json_array_get_length (array);
00844   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00845   for (i = 0; i < n; ++i)
00846     {
00847 #if DEBUG_INPUT
00848       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00849                input->nexperiments);
00850 #endif
00851       child = json_array_get_element (array, i);
00852       if (!input->nexperiments)
00853         {
00854           if (!experiment_open_json (input->experiment, child, 0))
00855             goto exit_on_error;
00856         }
00857       else
00858         {
00859           if (!experiment_open_json (input->experiment + input->
      nexperiments,
00860                                      child, input->experiment->ninputs))
00861             goto exit_on_error;
00862         }
00863       ++input->nexperiments;
00864 #if DEBUG_INPUT
00865       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00866                input->nexperiments);
00867 #endif
00868     }
```

```
00869   if (!input->nexperiments)
00870     {
00871       input_error (_("No optimization experiments"));
00872       goto exit_on_error;
00873     }
00874
00875   // Reading the variables data
00876   array = json_object_get_array_member (object, LABEL_VARIABLES);
00877   n = json_array_get_length (array);
00878   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00879   for (i = 0; i < n; ++i)
00880     {
00881 #if DEBUG_INPUT
00882       fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00883 #endif
00884       child = json_array_get_element (array, i);
00885       if (!variable_open_json (input->variable + input->
      nvariables, child,
00886                                input->algorithm, input->nsteps))
00887         goto exit_on_error;
00888       ++input->nvariables;
00889     }
00890   if (!input->nvariables)
00891     {
00892       input_error (_("No optimization variables"));
00893       goto exit_on_error;
00894     }
00895
00896   // Obtaining the error norm
00897   if (json_object_get_member (object, LABEL_NORM))
00898     {
00899       buffer = json_object_get_string_member (object, LABEL_NORM);
00900       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00901         input->norm = ERROR_NORM_EUCLIDIAN;
00902       else if (!strcmp (buffer, LABEL_MAXIMUM))
00903         input->norm = ERROR_NORM_MAXIMUM;
00904       else if (!strcmp (buffer, LABEL_P))
00905         {
00906           input->norm = ERROR_NORM_P;
00907           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00908           if (!error_code)
00909             {
00910               input_error (_("Bad P parameter"));
00911               goto exit_on_error;
00912             }
00913         }
00914       else if (!strcmp (buffer, LABEL_TAXICAB))
00915         input->norm = ERROR_NORM_TAXICAB;
00916       else
00917         {
00918           input_error (_("Unknown error norm"));
00919           goto exit_on_error;
00920         }
00921     }
00922   else
00923     input->norm = ERROR_NORM_EUCLIDIAN;
00924
00925   // Closing the JSON document
00926   g_object_unref (parser);
00927
00928 #if DEBUG_INPUT
00929   fprintf (stderr, "input_open_json: end\n");
00930 #endif
00931   return 1;
00932
00933 exit_on_error:
00934   g_object_unref (parser);
00935 #if DEBUG_INPUT
00936   fprintf (stderr, "input_open_json: end\n");
00937 #endif
00938   return 0;
00939 }
00940
00948 int
00949 input_open (char *filename)
00950 {
00951   xmlDoc *doc;
00952   JsonParser *parser;
00953
00954 #if DEBUG_INPUT
00955   fprintf (stderr, "input_open: start\n");
00956 #endif
00957
00958   // Resetting input data
00959   input_new ();
00960
```

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

## 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 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 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: $\max_i |w_i\,x_i|$.

***ERROR_NORM_P*** P-norm $\sqrt[p]{\sum_i |w_i\,x_i|^p}$.

***ERROR_NORM_TAXICAB*** Taxicab norm $\sum_i |w_i\,x_i|$.

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 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 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 949 of file input.c.

```
00950 {
00951   xmlDoc *doc;
00952   JsonParser *parser;
00953
00954 #if DEBUG_INPUT
00955   fprintf (stderr, "input_open: start\n");
00956 #endif
00957
00958   // Resetting input data
00959   input_new ();
00960
00961   // Opening input file
00962 #if DEBUG_INPUT
00963   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00964   fprintf (stderr, "input_open: trying XML format\n");
00965 #endif
00966   doc = xmlParseFile (filename);
00967   if (!doc)
00968     {
00969 #if DEBUG_INPUT
00970       fprintf (stderr, "input_open: trying JSON format\n");
00971 #endif
00972       parser = json_parser_new ();
00973       if (!json_parser_load_from_file (parser, filename, NULL))
00974         {
00975           input_error (_("Unable to parse the input file"));
00976           goto exit_on_error;
00977         }
00978       if (!input_open_json (parser))
00979         goto exit_on_error;
00980     }
00981   else if (!input_open_xml (doc))
00982     goto exit_on_error;
00983
00984   // Getting the working directory
00985   input->directory = g_path_get_dirname (filename);
00986   input->name = g_path_get_basename (filename);
00987
00988 #if DEBUG_INPUT
00989   fprintf (stderr, "input_open: end\n");
00990 #endif
00991   return 1;
00992
00993 exit_on_error:
00994   show_error (error_message);
00995   g_free (error_message);
00996   input_free ();
00997 #if DEBUG_INPUT
00998   fprintf (stderr, "input_open: end\n");
00999 #endif
01000   return 0;
01001 }
```

Here is the call graph for this function:

#### 4.9.3.3 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 560 of file input.c.

```
00561 {
00562   JsonNode *node, *child;
00563   JsonObject *object;
00564   JsonArray *array;
00565   const char *buffer;
00566   int error_code;
00567   unsigned int i, n;
00568
00569 #if DEBUG_INPUT
00570   fprintf (stderr, "input_open_json: start\n");
00571 #endif
00572
00573   // Resetting input data
00574   input->type = INPUT_TYPE_JSON;
00575
00576   // Getting the root node
00577 #if DEBUG_INPUT
00578   fprintf (stderr, "input_open_json: getting the root node\n");
00579 #endif
00580   node = json_parser_get_root (parser);
00581   object = json_node_get_object (node);
00582
00583   // Getting result and variables file names
00584   if (!input->result)
00585     {
00586       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00587       if (!buffer)
00588         buffer = result_name;
00589       input->result = g_strdup (buffer);
00590     }
00591   else
00592     input->result = g_strdup (result_name);
00593   if (!input->variables)
00594     {
00595       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00596       if (!buffer)
00597         buffer = variables_name;
00598       input->variables = g_strdup (buffer);
00599     }
00600   else
00601     input->variables = g_strdup (variables_name);
00602
00603   // Opening simulator program name
00604   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00605   if (!buffer)
00606     {
00607       input_error (_("Bad simulator program"));
00608       goto exit_on_error;
00609     }
00610   input->simulator = g_strdup (buffer);
00611
00612   // Opening evaluator program name
00613   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00614   if (buffer)
00615     input->evaluator = g_strdup (buffer);
00616
00617   // Obtaining pseudo-random numbers generator seed
00618   input->seed
00619     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00620                                          DEFAULT_RANDOM_SEED, &error_code);
00621   if (error_code)
00622     {
00623       input_error (_("Bad pseudo-random numbers generator seed"));
00624       goto exit_on_error;
00625     }
00626
00627   // Opening algorithm
00628   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00629   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00630     {
00631       input->algorithm = ALGORITHM_MONTE_CARLO;
00632
00633       // Obtaining simulations number
00634       input->nsimulations
00635         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00636       if (error_code)
00637         {
00638           input_error (_("Bad simulations number"));
00639           goto exit_on_error;
00640         }
00641     }
00642   else if (!strcmp (buffer, LABEL_SWEEP))
00643     input->algorithm = ALGORITHM_SWEEP;
00644   else if (!strcmp (buffer, LABEL_GENETIC))
00645     {
```

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

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

```
00809                   goto exit_on_error;
00810                 }
00811             }
00812           else
00813             {
00814               input_error
00815                 (_("Unknown method to estimate the direction search"));
00816               goto exit_on_error;
00817             }
00818           input->relaxation
00819             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00820                                                   DEFAULT_RELAXATION,
00821                                                   &error_code);
00822           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00823             {
00824               input_error (_("Invalid relaxation parameter"));
00825               goto exit_on_error;
00826             }
00827         }
00828       else
00829         input->nsteps = 0;
00830     }
00831   // Obtaining the threshold
00832   input->threshold
00833     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00834                                           &error_code);
00835   if (error_code)
00836     {
00837       input_error (_("Invalid threshold"));
00838       goto exit_on_error;
00839     }
00840
00841   // Reading the experimental data
00842   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00843   n = json_array_get_length (array);
00844   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00845   for (i = 0; i < n; ++i)
00846     {
00847 #if DEBUG_INPUT
00848       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00849                input->nexperiments);
00850 #endif
00851       child = json_array_get_element (array, i);
00852       if (!input->nexperiments)
00853         {
00854           if (!experiment_open_json (input->experiment, child, 0))
00855             goto exit_on_error;
00856         }
00857       else
00858         {
00859           if (!experiment_open_json (input->experiment +
      input->nexperiments,
00860                                      child, input->experiment->
      ninputs))
00861             goto exit_on_error;
00862         }
00863       ++input->nexperiments;
00864 #if DEBUG_INPUT
00865       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00866                input->nexperiments);
00867 #endif
00868     }
00869   if (!input->nexperiments)
00870     {
00871       input_error (_("No optimization experiments"));
00872       goto exit_on_error;
00873     }
00874
00875   // Reading the variables data
00876   array = json_object_get_array_member (object, LABEL_VARIABLES);
00877   n = json_array_get_length (array);
00878   input->variable = (Variable *) g_malloc (n * sizeof (
      Variable));
00879   for (i = 0; i < n; ++i)
00880     {
00881 #if DEBUG_INPUT
00882       fprintf (stderr, "input_open_json: nvariables=%u\n", input->
      nvariables);
00883 #endif
00884       child = json_array_get_element (array, i);
00885       if (!variable_open_json (input->variable +
      input->nvariables, child,
00886                                input->algorithm, input->
```

```
      nsteps))
00887         goto exit_on_error;
00888       ++input->nvariables;
00889     }
00890   if (!input->nvariables)
00891     {
00892       input_error (_("No optimization variables"));
00893       goto exit_on_error;
00894     }
00895
00896   // Obtaining the error norm
00897   if (json_object_get_member (object, LABEL_NORM))
00898     {
00899       buffer = json_object_get_string_member (object, LABEL_NORM);
00900       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00901         input->norm = ERROR_NORM_EUCLIDIAN;
00902       else if (!strcmp (buffer, LABEL_MAXIMUM))
00903         input->norm = ERROR_NORM_MAXIMUM;
00904       else if (!strcmp (buffer, LABEL_P))
00905         {
00906           input->norm = ERROR_NORM_P;
00907           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00908           if (!error_code)
00909             {
00910               input_error (_("Bad P parameter"));
00911               goto exit_on_error;
00912             }
00913         }
00914       else if (!strcmp (buffer, LABEL_TAXICAB))
00915         input->norm = ERROR_NORM_TAXICAB;
00916       else
00917         {
00918           input_error (_("Unknown error norm"));
00919           goto exit_on_error;
00920         }
00921     }
00922   else
00923     input->norm = ERROR_NORM_EUCLIDIAN;
00924
00925   // Closing the JSON document
00926   g_object_unref (parser);
00927
00928 #if DEBUG_INPUT
00929   fprintf (stderr, "input_open_json: end\n");
00930 #endif
00931   return 1;
00932
00933 exit_on_error:
00934   g_object_unref (parser);
00935 #if DEBUG_INPUT
00936   fprintf (stderr, "input_open_json: end\n");
00937 #endif
00938   return 0;
00939 }
```

Here is the call graph for this function:

#### 4.9.3.4 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, &error_code);
00421           if (error_code || input->relaxation < 0. || input->
    relaxation > 2.)
00422             {
00423               input_error (_("Invalid relaxation parameter"));
00424               goto exit_on_error;
00425             }
00426         }
00427       else
00428         input->nsteps = 0;
00429     }
00430   // Obtaining the threshold
00431   input->threshold =
00432     xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_THRESHOLD,
00433                                      0., &error_code);
00434   if (error_code)
00435     {
00436       input_error (_("Invalid threshold"));
00437       goto exit_on_error;
00438     }
00439
00440   // Reading the experimental data
00441   for (child = node->children; child; child = child->next)
00442     {
00443       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00444         break;
00445 #if DEBUG_INPUT
00446       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00447                input->nexperiments);
00448 #endif
00449       input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                    (1 + input->nexperiments) * sizeof (
    Experiment));
00452       if (!input->nexperiments)
00453         {
00454           if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456         }
00457       else
00458         {
00459           if (!experiment_open_xml (input->experiment +
    input->nexperiments,
00460                                     child, input->experiment->
    ninputs))
00461             goto exit_on_error;
00462         }
00463       ++input->nexperiments;
00464 #if DEBUG_INPUT
00465       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00466                input->nexperiments);
00467 #endif
00468     }
```

```
00469   if (!input->nexperiments)
00470     {
00471       input_error (_("No optimization experiments"));
00472       goto exit_on_error;
00473     }
00474   buffer = NULL;
00475
00476   // Reading the variables data
00477   for (; child; child = child->next)
00478     {
00479 #if DEBUG_INPUT
00480       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00481 #endif
00482       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00483         {
00484           snprintf (buffer2, 64, "%s %u: %s",
00485                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00486           input_error (buffer2);
00487           goto exit_on_error;
00488         }
00489       input->variable = (Variable *)
00490         g_realloc (input->variable,
00491                    (1 + input->nvariables) * sizeof (Variable));
00492       if (!variable_open_xml (input->variable +
    input->nvariables, child,
00493                               input->algorithm, input->nsteps))
00494         goto exit_on_error;
00495       ++input->nvariables;
00496     }
00497   if (!input->nvariables)
00498     {
00499       input_error (_("No optimization variables"));
00500       goto exit_on_error;
00501     }
00502   buffer = NULL;
00503
00504   // Obtaining the error norm
00505   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00506     {
00507       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00508       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00509         input->norm = ERROR_NORM_EUCLIDIAN;
00510       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00511         input->norm = ERROR_NORM_MAXIMUM;
00512       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00513         {
00514           input->norm = ERROR_NORM_P;
00515           input->p
00516             = xml_node_get_float (node, (const xmlChar *)
    LABEL_P, &error_code);
00517           if (!error_code)
00518             {
00519               input_error (_("Bad P parameter"));
00520               goto exit_on_error;
00521             }
00522         }
00523       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00524         input->norm = ERROR_NORM_TAXICAB;
00525       else
00526         {
00527           input_error (_("Unknown error norm"));
00528           goto exit_on_error;
00529         }
00530       xmlFree (buffer);
00531     }
00532   else
00533     input->norm = ERROR_NORM_EUCLIDIAN;
00534
00535   // Closing the XML document
00536   xmlFreeDoc (doc);
00537
00538 #if DEBUG_INPUT
00539   fprintf (stderr, "input_open_xml: end\n");
00540 #endif
00541   return 1;
00542
00543 exit_on_error:
00544   xmlFree (buffer);
00545   xmlFreeDoc (doc);
00546 #if DEBUG_INPUT
00547   fprintf (stderr, "input_open_xml: end\n");
00548 #endif
00549   return 0;
00550 }
```

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 2012-2017, 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 2012-2017, all rights reserved.

Definition in file interface.c.

## 4.11.2 Function Documentation

### 4.11.2.1 void input_save ( char * *filename* )

Function to save the input file.

**Parameters**

| | |
|---|---|
| *filename* | Input file name. |

Definition at line 575 of file interface.c.

```
00576 {
00577   xmlDoc *doc;
00578   JsonGenerator *generator;
00579
00580 #if DEBUG_INTERFACE
00581   fprintf (stderr, "input_save: start\n");
00582 #endif
00583
00584   // Getting the input file directory
00585   input->name = g_path_get_basename (filename);
00586   input->directory = g_path_get_dirname (filename);
00587
00588   if (input->type == INPUT_TYPE_XML)
00589     {
00590       // Opening the input file
00591       doc = xmlNewDoc ((const xmlChar *) "1.0");
00592       input_save_xml (doc);
00593
00594       // Saving the XML file
00595       xmlSaveFormatFile (filename, doc, 1);
00596
00597       // Freeing memory
00598       xmlFreeDoc (doc);
00599     }
00600   else
00601     {
00602       // Opening the input file
00603       generator = json_generator_new ();
00604       json_generator_set_pretty (generator, TRUE);
00605       input_save_json (generator);
00606
00607       // Saving the JSON file
00608       json_generator_to_file (generator, filename, NULL);
00609
00610       // Freeing memory
00611       g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615   fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
```

Here is the call graph for this function:

#### 4.11.2.2 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,
      LABEL_RANDOM);
00227           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00228         }
00229     }
00230 #if DEBUG_INTERFACE
00231   fprintf (stderr, "input_save_direction_json: end\n");
00232 #endif
00233 }
```

Here is the call graph for this function:

#### 4.11.2.3 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 void input_save_json ( JsonGenerator * *generator* )

Function to save the input file in JSON format.

### Parameters

| | |
|---|---|
| *generator* | JsonGenerator struct. |

Definition at line 412 of file interface.c.

```
00413 {
00414   unsigned int i, j;
00415   char *buffer;
00416   JsonNode *node, *child;
00417   JsonObject *object;
00418   JsonArray *array;
00419   GFile *file, *file2;
00420
00421 #if DEBUG_INTERFACE
00422   fprintf (stderr, "input_save_json: start\n");
00423 #endif
00424
00425   // Setting root JSON node
00426   node = json_node_new (JSON_NODE_OBJECT);
00427   object = json_node_get_object (node);
00428   json_generator_set_root (generator, node);
00429
00430   // Adding properties to the root JSON node
00431   if (strcmp (input->result, result_name))
00432     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00433   if (strcmp (input->variables, variables_name))
00434     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00435                                    input->variables);
00436   file = g_file_new_for_path (input->directory);
00437   file2 = g_file_new_for_path (input->simulator);
00438   buffer = g_file_get_relative_path (file, file2);
00439   g_object_unref (file2);
00440   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00441   g_free (buffer);
00442   if (input->evaluator)
00443     {
00444       file2 = g_file_new_for_path (input->evaluator);
00445       buffer = g_file_get_relative_path (file, file2);
00446       g_object_unref (file2);
00447       if (strlen (buffer))
00448         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00449       g_free (buffer);
00450     }
00451   if (input->seed != DEFAULT_RANDOM_SEED)
00452     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00453
00454   // Setting the algorithm
00455   buffer = (char *) g_slice_alloc (64);
00456   switch (input->algorithm)
00457     {
00458     case ALGORITHM_MONTE_CARLO:
00459       json_object_set_string_member (object, LABEL_ALGORITHM,
00460                                      LABEL_MONTE_CARLO);
00461       snprintf (buffer, 64, "%u", input->nsimulations);
00462       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00463       snprintf (buffer, 64, "%u", input->niterations);
00464       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00465       snprintf (buffer, 64, "%.3lg", input->tolerance);
00466       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00467       snprintf (buffer, 64, "%u", input->nbest);
00468       json_object_set_string_member (object, LABEL_NBEST, buffer);
00469       input_save_direction_json (node);
00470       break;
00471     case ALGORITHM_SWEEP:
00472       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00473       snprintf (buffer, 64, "%u", input->niterations);
00474       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00475       snprintf (buffer, 64, "%.3lg", input->tolerance);
00476       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00477       snprintf (buffer, 64, "%u", input->nbest);
00478       json_object_set_string_member (object, LABEL_NBEST, buffer);
00479       input_save_direction_json (node);
00480       break;
00481     default:
00482       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_GENETIC);
00483       snprintf (buffer, 64, "%u", input->nsimulations);
00484       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
```

```
00485       snprintf (buffer, 64, "%u", input->niterations);
00486       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00487       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00488       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00489       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00490       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00491       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00492       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00493       break;
00494     }
00495   g_slice_free1 (64, buffer);
00496   if (input->threshold != 0.)
00497     json_object_set_float (object, LABEL_THRESHOLD,
      input->threshold);
00498
00499   // Setting the experimental data
00500   array = json_array_new ();
00501   for (i = 0; i < input->nexperiments; ++i)
00502     {
00503       child = json_node_new (JSON_NODE_OBJECT);
00504       object = json_node_get_object (child);
00505       json_object_set_string_member (object, LABEL_NAME,
00506                                      input->experiment[i].name);
00507       if (input->experiment[i].weight != 1.)
00508         json_object_set_float (object, LABEL_WEIGHT,
00509                                input->experiment[i].weight);
00510       for (j = 0; j < input->experiment->ninputs; ++j)
00511         json_object_set_string_member (object, template[j],
00512                                        input->experiment[i].
      template[j]);
00513       json_array_add_element (array, child);
00514     }
00515   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00516
00517   // Setting the variables data
00518   array = json_array_new ();
00519   for (i = 0; i < input->nvariables; ++i)
00520     {
00521       child = json_node_new (JSON_NODE_OBJECT);
00522       object = json_node_get_object (child);
00523       json_object_set_string_member (object, LABEL_NAME,
00524                                      input->variable[i].name);
00525       json_object_set_float (object, LABEL_MINIMUM,
00526                              input->variable[i].rangemin);
00527       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00528         json_object_set_float (object,
      LABEL_ABSOLUTE_MINIMUM,
00529                                input->variable[i].rangeminabs);
00530       json_object_set_float (object, LABEL_MAXIMUM,
00531                              input->variable[i].rangemax);
00532       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00533         json_object_set_float (object,
      LABEL_ABSOLUTE_MAXIMUM,
00534                                input->variable[i].rangemaxabs);
00535       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00536         json_object_set_uint (object, LABEL_PRECISION,
00537                               input->variable[i].precision);
00538       if (input->algorithm == ALGORITHM_SWEEP)
00539         json_object_set_uint (object, LABEL_NSWEEPS,
00540                               input->variable[i].nsweeps);
00541       else if (input->algorithm == ALGORITHM_GENETIC)
00542         json_object_set_uint (object, LABEL_NBITS,
      input->variable[i].nbits);
00543       if (input->nsteps)
00544         json_object_set_float (object, LABEL_STEP,
      input->variable[i].step);
00545       json_array_add_element (array, child);
00546     }
00547   json_object_set_array_member (object, LABEL_VARIABLES, array);
00548
00549   // Saving the error norm
00550   switch (input->norm)
00551     {
00552     case ERROR_NORM_MAXIMUM:
00553       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00554       break;
00555     case ERROR_NORM_P:
00556       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00557       json_object_set_float (object, LABEL_P, input->
      p);
00558       break;
00559     case ERROR_NORM_TAXICAB:
00560       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00561     }
00562
00563 #if DEBUG_INTERFACE
```

```
00564   fprintf (stderr, "input_save_json: end\n");
00565 #endif
00566 }
```

Here is the call graph for this function:

#### 4.11.2.5 void input_save_xml ( xmlDoc * *doc* )

Function to save the input file in XML format.

**Parameters**

| | |
|---|---|
| *doc* | xmlDoc struct. |

Definition at line 242 of file interface.c.

```
00243 {
00244   unsigned int i, j;
00245   char *buffer;
00246   xmlNode *node, *child;
00247   GFile *file, *file2;
00248
00249 #if DEBUG_INTERFACE
00250   fprintf (stderr, "input_save_xml: start\n");
00251 #endif
00252
00253   // Setting root XML node
00254   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255   xmlDocSetRootElement (doc, node);
00256
00257   // Adding properties to the root XML node
00258   if (xmlStrcmp
00259       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                 (xmlChar *) input->result);
00262   if (xmlStrcmp
00263       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00264     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00265                 (xmlChar *) input->variables);
00266   file = g_file_new_for_path (input->directory);
00267   file2 = g_file_new_for_path (input->simulator);
00268   buffer = g_file_get_relative_path (file, file2);
00269   g_object_unref (file2);
00270   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00271   g_free (buffer);
00272   if (input->evaluator)
00273     {
00274       file2 = g_file_new_for_path (input->evaluator);
00275       buffer = g_file_get_relative_path (file, file2);
00276       g_object_unref (file2);
00277       if (xmlStrlen ((xmlChar *) buffer))
00278         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00279                     (xmlChar *) buffer);
00280       g_free (buffer);
00281     }
00282   if (input->seed != DEFAULT_RANDOM_SEED)
00283     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00284
00285   // Setting the algorithm
00286   buffer = (char *) g_slice_alloc (64);
00287   switch (input->algorithm)
00288     {
00289     case ALGORITHM_MONTE_CARLO:
00290       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00291                   (const xmlChar *) LABEL_MONTE_CARLO);
00292       snprintf (buffer, 64, "%u", input->nsimulations);
00293       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00294                   (xmlChar *) buffer);
00295       snprintf (buffer, 64, "%u", input->niterations);
00296       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00297                   (xmlChar *) buffer);
```

```
00298       snprintf (buffer, 64, "%.3lg", input->tolerance);
00299       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00300       snprintf (buffer, 64, "%u", input->nbest);
00301       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00302       input_save_direction_xml (node);
00303       break;
00304     case ALGORITHM_SWEEP:
00305       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306                   (const xmlChar *) LABEL_SWEEP);
00307       snprintf (buffer, 64, "%u", input->niterations);
00308       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309                   (xmlChar *) buffer);
00310       snprintf (buffer, 64, "%.3lg", input->tolerance);
00311       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%u", input->nbest);
00313       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314       input_save_direction_xml (node);
00315       break;
00316     default:
00317       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318                   (const xmlChar *) LABEL_GENETIC);
00319       snprintf (buffer, 64, "%u", input->nsimulations);
00320       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321                   (xmlChar *) buffer);
00322       snprintf (buffer, 64, "%u", input->niterations);
00323       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324                   (xmlChar *) buffer);
00325       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329                   (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332       break;
00333     }
00334   g_slice_free1 (64, buffer);
00335   if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00337                         input->threshold);
00338
00339   // Setting the experimental data
00340   for (i = 0; i < input->nexperiments; ++i)
00341     {
00342       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344                   (xmlChar *) input->experiment[i].name);
00345       if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00347                             input->experiment[i].weight);
00348       for (j = 0; j < input->experiment->ninputs; ++j)
00349         xmlSetProp (child, (const xmlChar *) template[j],
00350                     (xmlChar *) input->experiment[i].template[j]);
00351     }
00352
00353   // Setting the variables data
00354   for (i = 0; i < input->nvariables; ++i)
00355     {
00356       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358                   (xmlChar *) input->variable[i].name);
00359       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00360                           input->variable[i].rangemin);
00361       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00362         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00363                             input->variable[i].rangeminabs);
00364       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00365                           input->variable[i].rangemax);
00366       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00367         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00368                             input->variable[i].rangemaxabs);
00369       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00370         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00371                            input->variable[i].precision);
00372       if (input->algorithm == ALGORITHM_SWEEP)
00373         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00374                            input->variable[i].nsweeps);
00375       else if (input->algorithm == ALGORITHM_GENETIC)
```

```
00376         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00377                            input->variable[i].nbits);
00378       if (input->nsteps)
00379         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00380                             input->variable[i].step);
00381     }
00382
00383   // Saving the error norm
00384   switch (input->norm)
00385     {
00386     case ERROR_NORM_MAXIMUM:
00387       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00388                   (const xmlChar *) LABEL_MAXIMUM);
00389       break;
00390     case ERROR_NORM_P:
00391       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                   (const xmlChar *) LABEL_P);
00393       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00394       break;
00395     case ERROR_NORM_TAXICAB:
00396       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                   (const xmlChar *) LABEL_TAXICAB);
00398     }
00399
00400 #if DEBUG_INTERFACE
00401   fprintf (stderr, "input_save: end\n");
00402 #endif
00403 }
```

Here is the call graph for this function:

#### 4.11.2.6 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

**Returns**

Stochastic algorithm number.

Definition at line 725 of file interface.c.

```
00726 {
00727   unsigned int i;
00728 #if DEBUG_INTERFACE
00729   fprintf (stderr, "window_get_algorithm: start\n");
00730 #endif
00731   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00732 #if DEBUG_INTERFACE
00733   fprintf (stderr, "window_get_algorithm: %u\n", i);
00734   fprintf (stderr, "window_get_algorithm: end\n");
00735 #endif
00736   return i;
00737 }
```

Here is the call graph for this function:

#### 4.11.2.7 unsigned int window_get_direction ( )

Function to get the direction search method number.

**Returns**

Direction search method number.

Definition at line 745 of file interface.c.

```
00746 {
00747   unsigned int i;
00748 #if DEBUG_INTERFACE
00749   fprintf (stderr, "window_get_direction: start\n");
00750 #endif
00751   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00752 #if DEBUG_INTERFACE
00753   fprintf (stderr, "window_get_direction: %u\n", i);
00754   fprintf (stderr, "window_get_direction: end\n");
00755 #endif
00756   return i;
00757 }
```

Here is the call graph for this function:

#### 4.11.2.8 unsigned int window_get_norm ( )

Function to get the norm method number.

**Returns**

Norm method number.

Definition at line 765 of file interface.c.

```
00766 {
00767   unsigned int i;
00768 #if DEBUG_INTERFACE
00769   fprintf (stderr, "window_get_norm: start\n");
00770 #endif
00771   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00772 #if DEBUG_INTERFACE
00773   fprintf (stderr, "window_get_norm: %u\n", i);
00774   fprintf (stderr, "window_get_norm: end\n");
00775 #endif
00776   return i;
00777 }
```

Here is the call graph for this function:

#### 4.11.2.9 void window_new ( GtkApplication * *application* )

Function to open the main window.

**Parameters**

| | |
|---|---|
| *application* | GtkApplication struct. |

Definition at line 2075 of file interface.c.

```
02076 {
02077   unsigned int i;
02078   char *buffer, *buffer2, buffer3[64];
02079   char *label_algorithm[NALGORITHMS] = {
02080     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02081   };
02082   char *tip_algorithm[NALGORITHMS] = {
02083     _("Monte-Carlo brute force algorithm"),
02084     _("Sweep brute force algorithm"),
02085     _("Genetic algorithm")
02086   };
02087   char *label_direction[NDIRECTIONS] = {
02088     _("_Coordinates descent"), _("_Random")
02089   };
02090   char *tip_direction[NDIRECTIONS] = {
02091     _("Coordinates direction estimate method"),
02092     _("Random direction estimate method")
02093   };
02094   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02095   char *tip_norm[NNORMS] = {
02096     _("Euclidean error norm (L2)"),
02097     _("Maximum error norm (L)"),
02098     _("P error norm (Lp)"),
02099     _("Taxicab error norm (L1)")
02100   };
02101
02102 #if DEBUG_INTERFACE
02103   fprintf (stderr, "window_new: start\n");
02104 #endif
02105
02106   // Creating the window
02107   window->window = main_window
02108     = (GtkWindow *) gtk_application_window_new (application);
02109
02110   // Finish when closing the window
02111   g_signal_connect_swapped (window->window, "delete-event",
02112                             G_CALLBACK (g_application_quit),
02113                             G_APPLICATION (application));
02114
02115   // Setting the window title
02116   gtk_window_set_title (window->window, "MPCOTool");
02117
02118   // Creating the open button
02119   window->button_open = (GtkToolButton *) gtk_tool_button_new
02120     (gtk_image_new_from_icon_name ("document-open",
02121                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02122   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02123
02124   // Creating the save button
02125   window->button_save = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("document-save",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02128   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02129                     NULL);
02130
02131   // Creating the run button
02132   window->button_run = (GtkToolButton *) gtk_tool_button_new
02133     (gtk_image_new_from_icon_name ("system-run",
02134                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02135   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02136
02137   // Creating the options button
02138   window->button_options = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("preferences-system",
02140                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02141   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02142
02143   // Creating the help button
02144   window->button_help = (GtkToolButton *) gtk_tool_button_new
02145     (gtk_image_new_from_icon_name ("help-browser",
02146                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02147   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02148
02149   // Creating the about button
02150   window->button_about = (GtkToolButton *) gtk_tool_button_new
02151     (gtk_image_new_from_icon_name ("help-about",
02152                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02153   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02154
02155   // Creating the exit button
02156   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02157     (gtk_image_new_from_icon_name ("application-exit",
02158                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02159   g_signal_connect_swapped (window->button_exit, "clicked",
```

```
02160                             G_CALLBACK (g_application_quit),
02161                             G_APPLICATION (application));
02162
02163   // Creating the buttons bar
02164   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02165   gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02167   gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02169   gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
02171   gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02173   gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02175   gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02177   gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02179   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02180
02181   // Creating the simulator program label and entry
02182   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02183   window->button_simulator = (GtkFileChooserButton *)
02184     gtk_file_chooser_button_new (_("Simulator program"),
02185                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02186   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02187                                _("Simulator program executable file"));
02188   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02189
02190   // Creating the evaluator program label and entry
02191   window->check_evaluator = (GtkCheckButton *)
02192     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02193   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02194   window->button_evaluator = (GtkFileChooserButton *)
02195     gtk_file_chooser_button_new (_("Evaluator program"),
02196                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->button_evaluator),
02199      _("Optional evaluator program executable file"));
02200
02201   // Creating the results files labels and entries
02202   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02203   window->entry_result = (GtkEntry *) gtk_entry_new ();
02204   gtk_widget_set_tooltip_text
02205     (GTK_WIDGET (window->entry_result), _("Best results file"));
02206   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02207   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02208   gtk_widget_set_tooltip_text
02209     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02210
02211   // Creating the files grid and attaching widgets
02212   window->grid_files = (GtkGrid *) gtk_grid_new ();
02213   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02214                    0, 0, 1, 1);
02215   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02216                    1, 0, 1, 1);
02217   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02218                    0, 1, 1, 1);
02219   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02220                    1, 1, 1, 1);
02221   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02222                    0, 2, 1, 1);
02223   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02224                    1, 2, 1, 1);
02225   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02226                    0, 3, 1, 1);
02227   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02228                    1, 3, 1, 1);
02229
02230   // Creating the algorithm properties
```

```
02231   window->label_simulations = (GtkLabel *) gtk_label_new
02232     (_("Simulations number"));
02233   window->spin_simulations
02234     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02235   gtk_widget_set_tooltip_text
02236     (GTK_WIDGET (window->spin_simulations),
02237      _("Number of simulations to perform for each iteration"));
02238   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02239   window->label_iterations = (GtkLabel *)
02240     gtk_label_new (_("Iterations number"));
02241   window->spin_iterations
02242     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02245   g_signal_connect
02246     (window->spin_iterations, "value-changed",
      window_update, NULL);
02247   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02248   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02249   window->spin_tolerance =
02250     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02251   gtk_widget_set_tooltip_text
02252     (GTK_WIDGET (window->spin_tolerance),
02253      _("Tolerance to set the variable interval on the next iteration"));
02254   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02255   window->spin_bests
02256     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02257   gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_bests),
02259      _("Number of best simulations used to set the variable interval "
02260        "on the next iteration"));
02261   window->label_population
02262     = (GtkLabel *) gtk_label_new (_("Population number"));
02263   window->spin_population
02264     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02265   gtk_widget_set_tooltip_text
02266     (GTK_WIDGET (window->spin_population),
02267      _("Number of population for the genetic algorithm"));
02268   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02269   window->label_generations
02270     = (GtkLabel *) gtk_label_new (_("Generations number"));
02271   window->spin_generations
02272     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02273   gtk_widget_set_tooltip_text
02274     (GTK_WIDGET (window->spin_generations),
02275      _("Number of generations for the genetic algorithm"));
02276   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02277   window->spin_mutation
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279   gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_mutation),
02281      _("Ratio of mutation for the genetic algorithm"));
02282   window->label_reproduction
02283     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02284   window->spin_reproduction
02285     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286   gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_reproduction),
02288      _("Ratio of reproduction for the genetic algorithm"));
02289   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02290   window->spin_adaptation
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_adaptation),
02294      _("Ratio of adaptation for the genetic algorithm"));
02295   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02296   window->spin_threshold = (GtkSpinButton *)
02297     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02298                                     precision[DEFAULT_PRECISION]);
02299   gtk_widget_set_tooltip_text
02300     (GTK_WIDGET (window->spin_threshold),
02301      _("Threshold in the objective function to finish the simulations"));
02302   window->scrolled_threshold =
02303     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02304   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02305                      GTK_WIDGET (window->spin_threshold));
02306 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02307 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02308 //                         GTK_ALIGN_FILL);
02309
02310   // Creating the direction search method properties
02311   window->check_direction = (GtkCheckButton *)
02312     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02313   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02314   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02315   window->button_direction[0] = (GtkRadioButton *)
```

```
02316     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02317   gtk_grid_attach (window->grid_direction,
02318                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02319   g_signal_connect (window->button_direction[0], "clicked",
    window_update,
02320                     NULL);
02321   for (i = 0; ++i < NDIRECTIONS;)
02322     {
02323       window->button_direction[i] = (GtkRadioButton *)
02324         gtk_radio_button_new_with_mnemonic
02325         (gtk_radio_button_get_group (window->button_direction[0]),
02326          label_direction[i]);
02327       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02328                                    tip_direction[i]);
02329       gtk_grid_attach (window->grid_direction,
02330                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02331       g_signal_connect (window->button_direction[i], "clicked",
02332                         window_update, NULL);
02333     }
02334   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02335   window->spin_steps = (GtkSpinButton *)
02336     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02337   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02338   window->label_estimates
02339     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02340   window->spin_estimates = (GtkSpinButton *)
02341     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02342   window->label_relaxation
02343     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02344   window->spin_relaxation = (GtkSpinButton *)
02345     gtk_spin_button_new_with_range (0., 2., 0.001);
02346   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->label_steps),
02347                    0, NDIRECTIONS, 1, 1);
02348   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
    window->spin_steps),
02349                    1, NDIRECTIONS, 1, 1);
02350   gtk_grid_attach (window->grid_direction,
02351                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02352                    1, 1);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02355                    1);
02356   gtk_grid_attach (window->grid_direction,
02357                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02358                    1, 1);
02359   gtk_grid_attach (window->grid_direction,
02360                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02361                    1, 1);
02362
02363   // Creating the array of algorithms
02364   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02365   window->button_algorithm[0] = (GtkRadioButton *)
02366     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02367   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02368                                tip_algorithm[0]);
02369   gtk_grid_attach (window->grid_algorithm,
02370                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02371   g_signal_connect (window->button_algorithm[0], "clicked",
02372                     window_set_algorithm, NULL);
02373   for (i = 0; ++i < NALGORITHMS;)
02374     {
02375       window->button_algorithm[i] = (GtkRadioButton *)
02376         gtk_radio_button_new_with_mnemonic
02377         (gtk_radio_button_get_group (window->button_algorithm[0]),
02378          label_algorithm[i]);
02379       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02380                                    tip_algorithm[i]);
02381       gtk_grid_attach (window->grid_algorithm,
02382                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02383       g_signal_connect (window->button_algorithm[i], "clicked",
02384                         window_set_algorithm, NULL);
02385     }
02386   gtk_grid_attach (window->grid_algorithm,
02387                    GTK_WIDGET (window->label_simulations), 0,
02388                    NALGORITHMS, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm,
02390                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm,
02392                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02393                    1, 1);
02394   gtk_grid_attach (window->grid_algorithm,
02395                    GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02396                    1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02399                    1, 1);
```

```
02400   gtk_grid_attach (window->grid_algorithm,
02401                    GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02402                    1);
02403   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests),
02404                    0, NALGORITHMS + 3, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02406                    NALGORITHMS + 3, 1, 1);
02407   gtk_grid_attach (window->grid_algorithm,
02408                    GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02409                    1, 1);
02410   gtk_grid_attach (window->grid_algorithm,
02411                    GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02412                    1, 1);
02413   gtk_grid_attach (window->grid_algorithm,
02414                    GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02415                    1, 1);
02416   gtk_grid_attach (window->grid_algorithm,
02417                    GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02418                    1, 1);
02419   gtk_grid_attach (window->grid_algorithm,
02420                    GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02421                    1);
02422   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02423                    1, NALGORITHMS + 6, 1, 1);
02424   gtk_grid_attach (window->grid_algorithm,
02425                    GTK_WIDGET (window->label_reproduction), 0,
02426                    NALGORITHMS + 7, 1, 1);
02427   gtk_grid_attach (window->grid_algorithm,
02428                    GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02429                    1, 1);
02430   gtk_grid_attach (window->grid_algorithm,
02431                    GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02432                    1, 1);
02433   gtk_grid_attach (window->grid_algorithm,
02434                    GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02435                    1, 1);
02436   gtk_grid_attach (window->grid_algorithm,
02437                    GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02438                    2, 1);
02439   gtk_grid_attach (window->grid_algorithm,
02440                    GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02441                    2, 1);
02442   gtk_grid_attach (window->grid_algorithm,
02443                    GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02444                    1, 1);
02445   gtk_grid_attach (window->grid_algorithm,
02446                    GTK_WIDGET (window->scrolled_threshold), 1,
02447                    NALGORITHMS + 11, 1, 1);
02448   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02449   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02450                      GTK_WIDGET (window->grid_algorithm));
02451
02452   // Creating the variable widgets
02453   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02456   window->id_variable = g_signal_connect
02457     (window->combo_variable, "changed", window_set_variable, NULL);
02458   window->button_add_variable
02459     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02460                                                    GTK_ICON_SIZE_BUTTON);
02461   g_signal_connect
02462     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02465   window->button_remove_variable
02466     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02467                                                    GTK_ICON_SIZE_BUTTON);
02468   g_signal_connect
02469     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02470   gtk_widget_set_tooltip_text
02471     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02472   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02473   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02474   gtk_widget_set_tooltip_text
02475     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02476   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02477   window->id_variable_label = g_signal_connect
02478     (window->entry_variable, "changed",
      window_label_variable, NULL);
02479   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02480   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
```

```
02481     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482   gtk_widget_set_tooltip_text
02483     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02484   window->scrolled_min
02485     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02487                      GTK_WIDGET (window->spin_min));
02488   g_signal_connect (window->spin_min, "value-changed",
02489                     window_rangemin_variable, NULL);
02490   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02491   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02492     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02495   window->scrolled_max
02496     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02497   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02498                      GTK_WIDGET (window->spin_max));
02499   g_signal_connect (window->spin_max, "value-changed",
02500                     window_rangemax_variable, NULL);
02501   window->check_minabs = (GtkCheckButton *)
02502     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02503   g_signal_connect (window->check_minabs, "toggled",
    window_update, NULL);
02504   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02505     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_minabs),
02508      _("Minimum allowed value of the variable"));
02509   window->scrolled_minabs
02510     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02511   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02512                      GTK_WIDGET (window->spin_minabs));
02513   g_signal_connect (window->spin_minabs, "value-changed",
02514                     window_rangeminabs_variable, NULL);
02515   window->check_maxabs = (GtkCheckButton *)
02516     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02517   g_signal_connect (window->check_maxabs, "toggled",
    window_update, NULL);
02518   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02519     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02520   gtk_widget_set_tooltip_text
02521     (GTK_WIDGET (window->spin_maxabs),
02522      _("Maximum allowed value of the variable"));
02523   window->scrolled_maxabs
02524     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02525   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02526                      GTK_WIDGET (window->spin_maxabs));
02527   g_signal_connect (window->spin_maxabs, "value-changed",
02528                     window_rangemaxabs_variable, NULL);
02529   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02530   window->spin_precision = (GtkSpinButton *)
02531     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_precision),
02534      _("Number of precision floating point digits\n"
02535        "0 is for integer numbers"));
02536   g_signal_connect (window->spin_precision, "value-changed",
02537                     window_precision_variable, NULL);
02538   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02539   window->spin_sweeps =
02540     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02541   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02542                                _("Number of steps sweeping the variable"));
02543   g_signal_connect (window->spin_sweeps, "value-changed",
02544                     window_update_variable, NULL);
02545   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02546   window->spin_bits
02547     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02548   gtk_widget_set_tooltip_text
02549     (GTK_WIDGET (window->spin_bits),
02550      _("Number of bits to encode the variable"));
02551   g_signal_connect
02552     (window->spin_bits, "value-changed", window_update_variable, NULL)
    ;
02553   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02554   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02555     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02556   gtk_widget_set_tooltip_text
02557     (GTK_WIDGET (window->spin_step),
02558      _("Initial step size for the direction search method"));
02559   window->scrolled_step
02560     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02561   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02562                      GTK_WIDGET (window->spin_step));
02563   g_signal_connect
02564     (window->spin_step, "value-changed", window_step_variable, NULL);
```

```
02565   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02592   gtk_grid_attach (window->grid_variable,
02593                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02594   gtk_grid_attach (window->grid_variable,
02595                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02596   gtk_grid_attach (window->grid_variable,
02597                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02598   gtk_grid_attach (window->grid_variable,
02599                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02600   gtk_grid_attach (window->grid_variable,
02601                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02602   gtk_grid_attach (window->grid_variable,
02603                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02604   gtk_grid_attach (window->grid_variable,
02605                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02608   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02609   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02610                      GTK_WIDGET (window->grid_variable));
02611
02612   // Creating the experiment widgets
02613   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02615                                _("Experiment selector"));
02616   window->id_experiment = g_signal_connect
02617     (window->combo_experiment, "changed",
      window_set_experiment, NULL);
02618   window->button_add_experiment
02619     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02620                                                    GTK_ICON_SIZE_BUTTON);
02621   g_signal_connect
02622     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02623   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02624                                _("Add experiment"));
02625   window->button_remove_experiment
02626     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02627                                                    GTK_ICON_SIZE_BUTTON);
02628   g_signal_connect (window->button_remove_experiment, "clicked",
02629                     window_remove_experiment, NULL);
02630   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02631                                _("Remove experiment"));
02632   window->label_experiment
02633     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02634   window->button_experiment = (GtkFileChooserButton *)
02635     gtk_file_chooser_button_new (_("Experimental data file"),
02636                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02637   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02638                                _("Experimental data file"));
02639   window->id_experiment_name
02640     = g_signal_connect (window->button_experiment, "selection-changed",
02641                         window_name_experiment, NULL);
02642   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02643   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02644   window->spin_weight
02645     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02646   gtk_widget_set_tooltip_text
02647     (GTK_WIDGET (window->spin_weight),
02648      _("Weight factor to build the objective function"));
```

```
02649   g_signal_connect
02650     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02651   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02652   gtk_grid_attach (window->grid_experiment,
02653                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02654   gtk_grid_attach (window->grid_experiment,
02655                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02656   gtk_grid_attach (window->grid_experiment,
02657                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02658   gtk_grid_attach (window->grid_experiment,
02659                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02660   gtk_grid_attach (window->grid_experiment,
02661                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02662   gtk_grid_attach (window->grid_experiment,
02663                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02664   gtk_grid_attach (window->grid_experiment,
02665                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02666   for (i = 0; i < MAX_NINPUTS; ++i)
02667     {
02668       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02669       window->check_template[i] = (GtkCheckButton *)
02670         gtk_check_button_new_with_label (buffer3);
02671       window->id_template[i]
02672         = g_signal_connect (window->check_template[i], "toggled",
02673                             window_inputs_experiment, NULL);
02674       gtk_grid_attach (window->grid_experiment,
02675                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02676       window->button_template[i] =
02677         (GtkFileChooserButton *)
02678         gtk_file_chooser_button_new (_("Input template"),
02679                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02680       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02681                                    _("Experimental input template file"));
02682       window->id_input[i] =
02683         g_signal_connect_swapped (window->button_template[i],
02684                                   "selection-changed",
02685                                   (void (*)) window_template_experiment,
02686                                   (void *) (size_t) i);
02687       gtk_grid_attach (window->grid_experiment,
02688                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02689     }
02690   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02691   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02692                      GTK_WIDGET (window->grid_experiment));
02693
02694   // Creating the error norm widgets
02695   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02696   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02697   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02698                      GTK_WIDGET (window->grid_norm));
02699   window->button_norm[0] = (GtkRadioButton *)
02700     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02702                                tip_norm[0]);
02703   gtk_grid_attach (window->grid_norm,
02704                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02705   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02706   for (i = 0; ++i < NNORMS;)
02707     {
02708       window->button_norm[i] = (GtkRadioButton *)
02709         gtk_radio_button_new_with_mnemonic
02710         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02711       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02712                                    tip_norm[i]);
02713       gtk_grid_attach (window->grid_norm,
02714                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02715       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02716     }
02717   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02718   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1, 1);
02719   window->spin_p =
02720     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02721                                                       G_MAXDOUBLE, 0.01);
02722   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02723                                _("P parameter for the P error norm"));
02724   window->scrolled_p =
02725     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02726   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02727                      GTK_WIDGET (window->spin_p));
02728   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02729   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02730   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
```

```
    scrolled_p),
02731                     1, 2, 1, 2);
02732
02733   // Creating the grid and attaching the widgets to the grid
02734   window->grid = (GtkGrid *) gtk_grid_new ();
02735   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02736   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02737   gtk_grid_attach (window->grid,
02738                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02739   gtk_grid_attach (window->grid,
02740                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02741   gtk_grid_attach (window->grid,
02742                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02743   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02744   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
    window->grid));
02745
02746   // Setting the window logo
02747   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02748   gtk_window_set_icon (window->window, window->logo);
02749
02750   // Showing the window
02751   gtk_widget_show_all (GTK_WIDGET (window->window));
02752
02753   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02754 #if GTK_MINOR_VERSION >= 16
02755   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02756   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02757   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02758   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02759   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02760   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02761   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02762 #endif
02763
02764   // Reading initial example
02765   input_new ();
02766   buffer2 = g_get_current_dir ();
02767   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02768   g_free (buffer2);
02769   window_read (buffer);
02770   g_free (buffer);
02771
02772 #if DEBUG_INTERFACE
02773   fprintf (stderr, "window_new: start\n");
02774 #endif
02775 }
```

Here is the call graph for this function:

#### 4.11.2.10 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 1873 of file interface.c.

```
01874 {
01875   unsigned int i;
01876   char *buffer;
01877 #if DEBUG_INTERFACE
01878   fprintf (stderr, "window_read: start\n");
01879 #endif
```

```
01880
01881   // Reading new input file
01882   input_free ();
01883   if (!input_open (filename))
01884     {
01885 #if DEBUG_INTERFACE
01886       fprintf (stderr, "window_read: end\n");
01887 #endif
01888       return 0;
01889     }
01890
01891   // Setting GTK+ widgets data
01892   gtk_entry_set_text (window->entry_result, input->result);
01893   gtk_entry_set_text (window->entry_variables, input->
      variables);
01894   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01895   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01896                                  (window->button_simulator), buffer);
01897   g_free (buffer);
01898   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01899                                 (size_t) input->evaluator);
01900   if (input->evaluator)
01901     {
01902       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01903       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01904                                      (window->button_evaluator), buffer);
01905       g_free (buffer);
01906     }
01907   gtk_toggle_button_set_active
01908     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01909   switch (input->algorithm)
01910     {
01911     case ALGORITHM_MONTE_CARLO:
01912       gtk_spin_button_set_value (window->spin_simulations,
01913                                  (gdouble) input->nsimulations);
01914     case ALGORITHM_SWEEP:
01915       gtk_spin_button_set_value (window->spin_iterations,
01916                                  (gdouble) input->niterations);
01917       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01918       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01919       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01920                                     (window->check_direction),
      input->nsteps);
01921       if (input->nsteps)
01922         {
01923           gtk_toggle_button_set_active
01924             (GTK_TOGGLE_BUTTON (window->button_direction
01925                                 [input->direction]), TRUE);
01926           gtk_spin_button_set_value (window->spin_steps,
01927                                      (gdouble) input->nsteps);
01928           gtk_spin_button_set_value (window->spin_relaxation,
01929                                      (gdouble) input->relaxation);
01930           switch (input->direction)
01931             {
01932             case DIRECTION_METHOD_RANDOM:
01933               gtk_spin_button_set_value (window->spin_estimates,
01934                                          (gdouble) input->nestimates);
01935             }
01936         }
01937       break;
01938     default:
01939       gtk_spin_button_set_value (window->spin_population,
01940                                  (gdouble) input->nsimulations);
01941       gtk_spin_button_set_value (window->spin_generations,
01942                                  (gdouble) input->niterations);
01943       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01944       gtk_spin_button_set_value (window->spin_reproduction,
01945                                  input->reproduction_ratio);
01946       gtk_spin_button_set_value (window->spin_adaptation,
01947                                  input->adaptation_ratio);
01948     }
01949   gtk_toggle_button_set_active
01950     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01951   gtk_spin_button_set_value (window->spin_p, input->p);
01952   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01953   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01954   g_signal_handler_block (window->button_experiment,
01955                           window->id_experiment_name);
01956   gtk_combo_box_text_remove_all (window->combo_experiment);
```

```
01957   for (i = 0; i < input->nexperiments; ++i)
01958     gtk_combo_box_text_append_text (window->combo_experiment,
01959                                     input->experiment[i].name);
01960   g_signal_handler_unblock
01961     (window->button_experiment, window->
      id_experiment_name);
01962   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01963   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01964   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01965   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01966   gtk_combo_box_text_remove_all (window->combo_variable);
01967   for (i = 0; i < input->nvariables; ++i)
01968     gtk_combo_box_text_append_text (window->combo_variable,
01969                                     input->variable[i].name);
01970   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01971   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01972   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01973   window_set_variable ();
01974   window_update ();
01975
01976 #if DEBUG_INTERFACE
01977   fprintf (stderr, "window_read: end\n");
01978 #endif
01979   return 1;
01980 }
```

Here is the call graph for this function:

#### 4.11.2.11 int window_save ( )

Function to save the input file.

**Returns**

1 on OK, 0 on Cancel.

Definition at line 818 of file interface.c.

```
00819 {
00820   GtkFileChooserDialog *dlg;
00821   GtkFileFilter *filter1, *filter2;
00822   char *buffer;
00823
00824 #if DEBUG_INTERFACE
00825   fprintf (stderr, "window_save: start\n");
00826 #endif
00827
00828   // Opening the saving dialog
00829   dlg = (GtkFileChooserDialog *)
00830     gtk_file_chooser_dialog_new (_("Save file"),
00831                                  window->window,
00832                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00833                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00834                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00835   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836   buffer = g_build_filename (input->directory, input->name, NULL);
00837   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838   g_free (buffer);
00839
00840   // Adding XML filter
00841   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842   gtk_file_filter_set_name (filter1, "XML");
00843   gtk_file_filter_add_pattern (filter1, "*.xml");
00844   gtk_file_filter_add_pattern (filter1, "*.XML");
00845   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847   // Adding JSON filter
00848   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter2, "JSON");
00850   gtk_file_filter_add_pattern (filter2, "*.json");
```

```
00851   gtk_file_filter_add_pattern (filter2, "*.JSON");
00852   gtk_file_filter_add_pattern (filter2, "*.js");
00853   gtk_file_filter_add_pattern (filter2, "*.JS");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856   if (input->type == INPUT_TYPE_XML)
00857     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858   else
00859     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861   // If OK response then saving
00862   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863     {
00864       // Setting input file type
00865       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866       buffer = (char *) gtk_file_filter_get_name (filter1);
00867       if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869       else
00870         input->type = INPUT_TYPE_JSON;
00871
00872       // Adding properties to the root XML node
00873       input->simulator = gtk_file_chooser_get_filename
00874         (GTK_FILE_CHOOSER (window->button_simulator));
00875       if (gtk_toggle_button_get_active
00876           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877         input->evaluator = gtk_file_chooser_get_filename
00878           (GTK_FILE_CHOOSER (window->button_evaluator));
00879       else
00880         input->evaluator = NULL;
00881       if (input->type == INPUT_TYPE_XML)
00882         {
00883           input->result
00884             = (char *) xmlStrdup ((const xmlChar *)
00885                                   gtk_entry_get_text (window->entry_result));
00886           input->variables
00887             = (char *) xmlStrdup ((const xmlChar *)
00888                                   gtk_entry_get_text (window->
    entry_variables));
00889         }
00890       else
00891         {
00892           input->result = g_strdup (gtk_entry_get_text (window->
    entry_result));
00893           input->variables =
00894             g_strdup (gtk_entry_get_text (window->entry_variables));
00895         }
00896
00897       // Setting the algorithm
00898       switch (window_get_algorithm ())
00899         {
00900         case ALGORITHM_MONTE_CARLO:
00901           input->algorithm = ALGORITHM_MONTE_CARLO;
00902           input->nsimulations
00903             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00904           input->niterations
00905             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00906           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00907           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00908           window_save_direction ();
00909           break;
00910         case ALGORITHM_SWEEP:
00911           input->algorithm = ALGORITHM_SWEEP;
00912           input->niterations
00913             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00914           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00915           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00916           window_save_direction ();
00917           break;
00918         default:
00919           input->algorithm = ALGORITHM_GENETIC;
00920           input->nsimulations
00921             = gtk_spin_button_get_value_as_int (window->spin_population);
00922           input->niterations
00923             = gtk_spin_button_get_value_as_int (window->spin_generations);
00924           input->mutation_ratio
00925             = gtk_spin_button_get_value (window->spin_mutation);
00926           input->reproduction_ratio
00927             = gtk_spin_button_get_value (window->spin_reproduction);
00928           input->adaptation_ratio
00929             = gtk_spin_button_get_value (window->spin_adaptation);
00930           break;
00931         }
```

```
00932       input->norm = window_get_norm ();
00933       input->p = gtk_spin_button_get_value (window->spin_p);
00934       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00935
00936       // Saving the XML file
00937       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00938       input_save (buffer);
00939
00940       // Closing and freeing memory
00941       g_free (buffer);
00942       gtk_widget_destroy (GTK_WIDGET (dlg));
00943 #if DEBUG_INTERFACE
00944       fprintf (stderr, "window_save: end\n");
00945 #endif
00946       return 1;
00947     }
00948
00949   // Closing and freeing memory
00950   gtk_widget_destroy (GTK_WIDGET (dlg));
00951 #if DEBUG_INTERFACE
00952   fprintf (stderr, "window_save: end\n");
00953 #endif
00954   return 0;
00955 }
```

Here is the call graph for this function:

#### 4.11.2.12 void window_template_experiment ( void * *data* )

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

**Parameters**

| | |
|---|---|
| *data* | Callback data (i-th input template). |

Definition at line 1517 of file interface.c.

```
01518 {
01519   unsigned int i, j;
01520   char *buffer;
01521   GFile *file1, *file2;
01522 #if DEBUG_INTERFACE
01523   fprintf (stderr, "window_template_experiment: start\n");
01524 #endif
01525   i = (size_t) data;
01526   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01527   file1
01528     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01529   file2 = g_file_new_for_path (input->directory);
01530   buffer = g_file_get_relative_path (file2, file1);
01531   if (input->type == INPUT_TYPE_XML)
01532     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01533   else
01534     input->experiment[j].template[i] = g_strdup (buffer);
01535   g_free (buffer);
01536   g_object_unref (file2);
01537   g_object_unref (file1);
01538 #if DEBUG_INTERFACE
01539   fprintf (stderr, "window_template_experiment: end\n");
01540 #endif
01541 }
```

## 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 2012-2017, 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,
      LABEL_RANDOM);
00227           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00228         }
00229     }
00230 #if DEBUG_INTERFACE
00231   fprintf (stderr, "input_save_direction_json: end\n");
00232 #endif
00233 }
00234
00241 void
00242 input_save_xml (xmlDoc * doc)
00243 {
00244   unsigned int i, j;
00245   char *buffer;
00246   xmlNode *node, *child;
00247   GFile *file, *file2;
00248
00249 #if DEBUG_INTERFACE
00250   fprintf (stderr, "input_save_xml: start\n");
00251 #endif
00252
00253   // Setting root XML node
00254   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255   xmlDocSetRootElement (doc, node);
00256
00257   // Adding properties to the root XML node
00258   if (xmlStrcmp
00259       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                 (xmlChar *) input->result);
00262   if (xmlStrcmp
00263       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00264     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00265                 (xmlChar *) input->variables);
00266   file = g_file_new_for_path (input->directory);
00267   file2 = g_file_new_for_path (input->simulator);
00268   buffer = g_file_get_relative_path (file, file2);
00269   g_object_unref (file2);
00270   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00271   g_free (buffer);
00272   if (input->evaluator)
00273     {
00274       file2 = g_file_new_for_path (input->evaluator);
00275       buffer = g_file_get_relative_path (file, file2);
00276       g_object_unref (file2);
00277       if (xmlStrlen ((xmlChar *) buffer))
00278         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00279                     (xmlChar *) buffer);
00280       g_free (buffer);
00281     }
00282   if (input->seed != DEFAULT_RANDOM_SEED)
00283     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00284
00285   // Setting the algorithm
```

```
00286   buffer = (char *) g_slice_alloc (64);
00287   switch (input->algorithm)
00288     {
00289     case ALGORITHM_MONTE_CARLO:
00290       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00291                   (const xmlChar *) LABEL_MONTE_CARLO);
00292       snprintf (buffer, 64, "%u", input->nsimulations);
00293       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00294                   (xmlChar *) buffer);
00295       snprintf (buffer, 64, "%u", input->niterations);
00296       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00297                   (xmlChar *) buffer);
00298       snprintf (buffer, 64, "%.3lg", input->tolerance);
00299       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00300       snprintf (buffer, 64, "%u", input->nbest);
00301       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00302       input_save_direction_xml (node);
00303       break;
00304     case ALGORITHM_SWEEP:
00305       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306                   (const xmlChar *) LABEL_SWEEP);
00307       snprintf (buffer, 64, "%u", input->niterations);
00308       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309                   (xmlChar *) buffer);
00310       snprintf (buffer, 64, "%.3lg", input->tolerance);
00311       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%u", input->nbest);
00313       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314       input_save_direction_xml (node);
00315       break;
00316     default:
00317       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318                   (const xmlChar *) LABEL_GENETIC);
00319       snprintf (buffer, 64, "%u", input->nsimulations);
00320       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321                   (xmlChar *) buffer);
00322       snprintf (buffer, 64, "%u", input->niterations);
00323       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324                   (xmlChar *) buffer);
00325       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329                   (xmlChar *) buffer);
00330       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332       break;
00333     }
00334   g_slice_free1 (64, buffer);
00335   if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00337                         input->threshold);
00338
00339   // Setting the experimental data
00340   for (i = 0; i < input->nexperiments; ++i)
00341     {
00342       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344                   (xmlChar *) input->experiment[i].name);
00345       if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00347                             input->experiment[i].weight);
00348       for (j = 0; j < input->experiment->ninputs; ++j)
00349         xmlSetProp (child, (const xmlChar *) template[j],
00350                     (xmlChar *) input->experiment[i].template[j]);
00351     }
00352
00353   // Setting the variables data
00354   for (i = 0; i < input->nvariables; ++i)
00355     {
00356       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358                   (xmlChar *) input->variable[i].name);
00359       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00360                           input->variable[i].rangemin);
00361       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00362         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00363                             input->variable[i].rangeminabs);
00364       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00365                           input->variable[i].rangemax);
00366       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00367         xml_node_set_float (child, (const xmlChar *)
```

```
      LABEL_ABSOLUTE_MAXIMUM,
00368                              input->variable[i].rangemaxabs);
00369       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00370         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00371                            input->variable[i].precision);
00372       if (input->algorithm == ALGORITHM_SWEEP)
00373         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00374                            input->variable[i].nsweeps);
00375       else if (input->algorithm == ALGORITHM_GENETIC)
00376         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00377                            input->variable[i].nbits);
00378       if (input->nsteps)
00379         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00380                             input->variable[i].step);
00381     }
00382
00383   // Saving the error norm
00384   switch (input->norm)
00385     {
00386     case ERROR_NORM_MAXIMUM:
00387       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00388                   (const xmlChar *) LABEL_MAXIMUM);
00389       break;
00390     case ERROR_NORM_P:
00391       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                   (const xmlChar *) LABEL_P);
00393       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00394       break;
00395     case ERROR_NORM_TAXICAB:
00396       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                   (const xmlChar *) LABEL_TAXICAB);
00398     }
00399
00400 #if DEBUG_INTERFACE
00401   fprintf (stderr, "input_save: end\n");
00402 #endif
00403 }
00404
00411 void
00412 input_save_json (JsonGenerator * generator)
00413 {
00414   unsigned int i, j;
00415   char *buffer;
00416   JsonNode *node, *child;
00417   JsonObject *object;
00418   JsonArray *array;
00419   GFile *file, *file2;
00420
00421 #if DEBUG_INTERFACE
00422   fprintf (stderr, "input_save_json: start\n");
00423 #endif
00424
00425   // Setting root JSON node
00426   node = json_node_new (JSON_NODE_OBJECT);
00427   object = json_node_get_object (node);
00428   json_generator_set_root (generator, node);
00429
00430   // Adding properties to the root JSON node
00431   if (strcmp (input->result, result_name))
00432     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00433   if (strcmp (input->variables, variables_name))
00434     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00435                                    input->variables);
00436   file = g_file_new_for_path (input->directory);
00437   file2 = g_file_new_for_path (input->simulator);
00438   buffer = g_file_get_relative_path (file, file2);
00439   g_object_unref (file2);
00440   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00441   g_free (buffer);
00442   if (input->evaluator)
00443     {
00444       file2 = g_file_new_for_path (input->evaluator);
00445       buffer = g_file_get_relative_path (file, file2);
00446       g_object_unref (file2);
00447       if (strlen (buffer))
00448         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00449       g_free (buffer);
00450     }
00451   if (input->seed != DEFAULT_RANDOM_SEED)
00452     json_object_set_uint (object, LABEL_SEED,
      input->seed);
```

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

```
00534                                 input->variable[i].rangemaxabs);
00535       if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00536         json_object_set_uint (object, LABEL_PRECISION,
00537                               input->variable[i].precision);
00538       if (input->algorithm == ALGORITHM_SWEEP)
00539         json_object_set_uint (object, LABEL_NSWEEPS,
00540                               input->variable[i].nsweeps);
00541       else if (input->algorithm == ALGORITHM_GENETIC)
00542         json_object_set_uint (object, LABEL_NBITS,
    input->variable[i].nbits);
00543       if (input->nsteps)
00544         json_object_set_float (object, LABEL_STEP,
    input->variable[i].step);
00545       json_array_add_element (array, child);
00546     }
00547   json_object_set_array_member (object, LABEL_VARIABLES, array);
00548
00549   // Saving the error norm
00550   switch (input->norm)
00551     {
00552     case ERROR_NORM_MAXIMUM:
00553       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00554       break;
00555     case ERROR_NORM_P:
00556       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00557       json_object_set_float (object, LABEL_P, input->
    p);
00558       break;
00559     case ERROR_NORM_TAXICAB:
00560       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00561     }
00562
00563 #if DEBUG_INTERFACE
00564   fprintf (stderr, "input_save_json: end\n");
00565 #endif
00566 }
00567
00574 void
00575 input_save (char *filename)
00576 {
00577   xmlDoc *doc;
00578   JsonGenerator *generator;
00579
00580 #if DEBUG_INTERFACE
00581   fprintf (stderr, "input_save: start\n");
00582 #endif
00583
00584   // Getting the input file directory
00585   input->name = g_path_get_basename (filename);
00586   input->directory = g_path_get_dirname (filename);
00587
00588   if (input->type == INPUT_TYPE_XML)
00589     {
00590       // Opening the input file
00591       doc = xmlNewDoc ((const xmlChar *) "1.0");
00592       input_save_xml (doc);
00593
00594       // Saving the XML file
00595       xmlSaveFormatFile (filename, doc, 1);
00596
00597       // Freeing memory
00598       xmlFreeDoc (doc);
00599     }
00600   else
00601     {
00602       // Opening the input file
00603       generator = json_generator_new ();
00604       json_generator_set_pretty (generator, TRUE);
00605       input_save_json (generator);
00606
00607       // Saving the JSON file
00608       json_generator_to_file (generator, filename, NULL);
00609
00610       // Freeing memory
00611       g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615   fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
00618
00623 void
00624 options_new ()
00625 {
00626 #if DEBUG_INTERFACE
```

```
00627   fprintf (stderr, "options_new: start\n");
00628 #endif
00629   options->label_seed = (GtkLabel *)
00630     gtk_label_new (_("Pseudo-random numbers generator seed"));
00631   options->spin_seed = (GtkSpinButton *)
00632     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00633   gtk_widget_set_tooltip_text
00634     (GTK_WIDGET (options->spin_seed),
00635      _("Seed to init the pseudo-random numbers generator"));
00636   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00637   options->label_threads = (GtkLabel *)
00638     gtk_label_new (_("Threads number for the stochastic algorithm"));
00639   options->spin_threads
00640     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00641   gtk_widget_set_tooltip_text
00642     (GTK_WIDGET (options->spin_threads),
00643      _("Number of threads to perform the calibration/optimization for "
00644        "the stochastic algorithm"));
00645   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00646   options->label_direction = (GtkLabel *)
00647     gtk_label_new (_("Threads number for the direction search method"));
00648   options->spin_direction =
00649     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650   gtk_widget_set_tooltip_text (GTK_WIDGET (options->spin_direction),
00651                                _
00652                                ("Number of threads to perform the calibration/optimization for "
00653                                 "the direction search method"));
00654   gtk_spin_button_set_value (options->spin_direction,
00655                              (gdouble) nthreads_direction);
00656   options->grid = (GtkGrid *) gtk_grid_new ();
00657   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00658   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00659   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00660                    1, 1);
00661   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00662                    1);
00663   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00664                    1, 1);
00665   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00666                    1, 1);
00667   gtk_widget_show_all (GTK_WIDGET (options->grid));
00668   options->dialog = (GtkDialog *)
00669     gtk_dialog_new_with_buttons (_("Options"),
00670                                  window->window,
00671                                  GTK_DIALOG_MODAL,
00672                                  _("_OK"), GTK_RESPONSE_OK,
00673                                  _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00674   gtk_container_add
00675     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00676      GTK_WIDGET (options->grid));
00677   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00678     {
00679       input->seed
00680         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00681       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00682       nthreads_direction
00683         = gtk_spin_button_get_value_as_int (options->spin_direction);
00684     }
00685   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00686 #if DEBUG_INTERFACE
00687   fprintf (stderr, "options_new: end\n");
00688 #endif
00689 }
00690
00695 void
00696 running_new ()
00697 {
00698 #if DEBUG_INTERFACE
00699   fprintf (stderr, "running_new: start\n");
00700 #endif
00701   running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00702   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00703   running->grid = (GtkGrid *) gtk_grid_new ();
00704   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00705   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00706   running->dialog = (GtkDialog *)
00707     gtk_dialog_new_with_buttons (_("Calculating"),
00708                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00709   gtk_container_add (GTK_CONTAINER
00710                      (gtk_dialog_get_content_area (running->dialog)),
00711                      GTK_WIDGET (running->grid));
00712   gtk_spinner_start (running->spinner);
00713   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00714 #if DEBUG_INTERFACE
00715   fprintf (stderr, "running_new: end\n");
```

```
00716 #endif
00717 }
00718
00724 unsigned int
00725 window_get_algorithm ()
00726 {
00727   unsigned int i;
00728 #if DEBUG_INTERFACE
00729   fprintf (stderr, "window_get_algorithm: start\n");
00730 #endif
00731   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00732 #if DEBUG_INTERFACE
00733   fprintf (stderr, "window_get_algorithm: %u\n", i);
00734   fprintf (stderr, "window_get_algorithm: end\n");
00735 #endif
00736   return i;
00737 }
00738
00744 unsigned int
00745 window_get_direction ()
00746 {
00747   unsigned int i;
00748 #if DEBUG_INTERFACE
00749   fprintf (stderr, "window_get_direction: start\n");
00750 #endif
00751   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00752 #if DEBUG_INTERFACE
00753   fprintf (stderr, "window_get_direction: %u\n", i);
00754   fprintf (stderr, "window_get_direction: end\n");
00755 #endif
00756   return i;
00757 }
00758
00764 unsigned int
00765 window_get_norm ()
00766 {
00767   unsigned int i;
00768 #if DEBUG_INTERFACE
00769   fprintf (stderr, "window_get_norm: start\n");
00770 #endif
00771   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00772 #if DEBUG_INTERFACE
00773   fprintf (stderr, "window_get_norm: %u\n", i);
00774   fprintf (stderr, "window_get_norm: end\n");
00775 #endif
00776   return i;
00777 }
00778
00783 void
00784 window_save_direction ()
00785 {
00786 #if DEBUG_INTERFACE
00787   fprintf (stderr, "window_save_direction: start\n");
00788 #endif
00789   if (gtk_toggle_button_get_active
00790       (GTK_TOGGLE_BUTTON (window->check_direction)))
00791     {
00792       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00793       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00794       switch (window_get_direction ())
00795         {
00796         case DIRECTION_METHOD_COORDINATES:
00797           input->direction = DIRECTION_METHOD_COORDINATES;
00798           break;
00799         default:
00800           input->direction = DIRECTION_METHOD_RANDOM;
00801           input->nestimates
00802             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00803         }
00804     }
00805   else
00806     input->nsteps = 0;
00807 #if DEBUG_INTERFACE
00808   fprintf (stderr, "window_save_direction: end\n");
00809 #endif
00810 }
00811
00817 int
00818 window_save ()
00819 {
00820   GtkFileChooserDialog *dlg;
00821   GtkFileFilter *filter1, *filter2;
```

```
00822   char *buffer;
00823
00824 #if DEBUG_INTERFACE
00825   fprintf (stderr, "window_save: start\n");
00826 #endif
00827
00828   // Opening the saving dialog
00829   dlg = (GtkFileChooserDialog *)
00830     gtk_file_chooser_dialog_new (_("Save file"),
00831                                  window->window,
00832                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00833                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00834                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00835   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836   buffer = g_build_filename (input->directory, input->name, NULL);
00837   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838   g_free (buffer);
00839
00840   // Adding XML filter
00841   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842   gtk_file_filter_set_name (filter1, "XML");
00843   gtk_file_filter_add_pattern (filter1, "*.xml");
00844   gtk_file_filter_add_pattern (filter1, "*.XML");
00845   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847   // Adding JSON filter
00848   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter2, "JSON");
00850   gtk_file_filter_add_pattern (filter2, "*.json");
00851   gtk_file_filter_add_pattern (filter2, "*.JSON");
00852   gtk_file_filter_add_pattern (filter2, "*.js");
00853   gtk_file_filter_add_pattern (filter2, "*.JS");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856   if (input->type == INPUT_TYPE_XML)
00857     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858   else
00859     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861   // If OK response then saving
00862   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863     {
00864       // Setting input file type
00865       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866       buffer = (char *) gtk_file_filter_get_name (filter1);
00867       if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869       else
00870         input->type = INPUT_TYPE_JSON;
00871
00872       // Adding properties to the root XML node
00873       input->simulator = gtk_file_chooser_get_filename
00874         (GTK_FILE_CHOOSER (window->button_simulator));
00875       if (gtk_toggle_button_get_active
00876           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877         input->evaluator = gtk_file_chooser_get_filename
00878           (GTK_FILE_CHOOSER (window->button_evaluator));
00879       else
00880         input->evaluator = NULL;
00881       if (input->type == INPUT_TYPE_XML)
00882         {
00883           input->result
00884             = (char *) xmlStrdup ((const xmlChar *)
00885                                   gtk_entry_get_text (window->entry_result));
00886           input->variables
00887             = (char *) xmlStrdup ((const xmlChar *)
00888                                   gtk_entry_get_text (window->entry_variables));
00889         }
00890       else
00891         {
00892           input->result = g_strdup (gtk_entry_get_text (window->
      entry_result));
00893           input->variables =
00894             g_strdup (gtk_entry_get_text (window->entry_variables));
00895         }
00896
00897       // Setting the algorithm
00898       switch (window_get_algorithm ())
00899         {
00900         case ALGORITHM_MONTE_CARLO:
00901           input->algorithm = ALGORITHM_MONTE_CARLO;
00902           input->nsimulations
00903             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00904           input->niterations
00905             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00906           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
```

```
00907           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00908           window_save_direction ();
00909           break;
00910         case ALGORITHM_SWEEP:
00911           input->algorithm = ALGORITHM_SWEEP;
00912           input->niterations
00913             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00914           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00915           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00916           window_save_direction ();
00917           break;
00918         default:
00919           input->algorithm = ALGORITHM_GENETIC;
00920           input->nsimulations
00921             = gtk_spin_button_get_value_as_int (window->spin_population);
00922           input->niterations
00923             = gtk_spin_button_get_value_as_int (window->spin_generations);
00924           input->mutation_ratio
00925             = gtk_spin_button_get_value (window->spin_mutation);
00926           input->reproduction_ratio
00927             = gtk_spin_button_get_value (window->spin_reproduction);
00928           input->adaptation_ratio
00929             = gtk_spin_button_get_value (window->spin_adaptation);
00930           break;
00931         }
00932       input->norm = window_get_norm ();
00933       input->p = gtk_spin_button_get_value (window->spin_p);
00934       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00935
00936       // Saving the XML file
00937       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00938       input_save (buffer);
00939
00940       // Closing and freeing memory
00941       g_free (buffer);
00942       gtk_widget_destroy (GTK_WIDGET (dlg));
00943 #if DEBUG_INTERFACE
00944       fprintf (stderr, "window_save: end\n");
00945 #endif
00946       return 1;
00947     }
00948
00949   // Closing and freeing memory
00950   gtk_widget_destroy (GTK_WIDGET (dlg));
00951 #if DEBUG_INTERFACE
00952   fprintf (stderr, "window_save: end\n");
00953 #endif
00954   return 0;
00955 }
00956
00961 void
00962 window_run ()
00963 {
00964   unsigned int i;
00965   char *msg, *msg2, buffer[64], buffer2[64];
00966 #if DEBUG_INTERFACE
00967   fprintf (stderr, "window_run: start\n");
00968 #endif
00969   if (!window_save ())
00970     {
00971 #if DEBUG_INTERFACE
00972       fprintf (stderr, "window_run: end\n");
00973 #endif
00974       return;
00975     }
00976   running_new ();
00977   while (gtk_events_pending ())
00978     gtk_main_iteration ();
00979   optimize_open ();
00980 #if DEBUG_INTERFACE
00981   fprintf (stderr, "window_run: closing running dialog\n");
00982 #endif
00983   gtk_spinner_stop (running->spinner);
00984   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00985 #if DEBUG_INTERFACE
00986   fprintf (stderr, "window_run: displaying results\n");
00987 #endif
00988   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00989   msg2 = g_strdup (buffer);
00990   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00991     {
00992       snprintf (buffer, 64, "%s = %s\n",
00993                 input->variable[i].name, format[input->
```

```
      variable[i].precision]);
00994       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00995       msg = g_strconcat (msg2, buffer2, NULL);
00996       g_free (msg2);
00997     }
00998   snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
00999             optimize->calculation_time);
01000   msg = g_strconcat (msg2, buffer, NULL);
01001   g_free (msg2);
01002   show_message (_("Best result"), msg, INFO_TYPE);
01003   g_free (msg);
01004 #if DEBUG_INTERFACE
01005   fprintf (stderr, "window_run: freeing memory\n");
01006 #endif
01007   optimize_free ();
01008 #if DEBUG_INTERFACE
01009   fprintf (stderr, "window_run: end\n");
01010 #endif
01011 }
01012
01017 void
01018 window_help ()
01019 {
01020   char *buffer, *buffer2;
01021 #if DEBUG_INTERFACE
01022   fprintf (stderr, "window_help: start\n");
01023 #endif
01024   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01025                               _("user-manual.pdf"), NULL);
01026   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01027   g_free (buffer2);
01028   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01029 #if DEBUG_INTERFACE
01030   fprintf (stderr, "window_help: uri=%s\n", buffer);
01031 #endif
01032   g_free (buffer);
01033 #if DEBUG_INTERFACE
01034   fprintf (stderr, "window_help: end\n");
01035 #endif
01036 }
01037
01042 void
01043 window_about ()
01044 {
01045   static const gchar *authors[] = {
01046     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01047     "Borja Latorre Garcés <borja.latorre@csic.es>",
01048     NULL
01049   };
01050 #if DEBUG_INTERFACE
01051   fprintf (stderr, "window_about: start\n");
01052 #endif
01053   gtk_show_about_dialog
01054     (window->window,
01055      "program_name", "MPCOTool",
01056      "comments",
01057      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01058        "A software to perform calibrations or optimizations of empirical"
01059        " parameters"),
01060      "authors", authors,
01061      "translator-credits",
01062      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01063      "(english, french and spanish)\n"
01064      "Uğur Çayoğlu (german)",
01065      "version", "3.4.2",
01066      "copyright", "Copyright 2012-2017 Javier Burguete Tolosa",
01067      "logo", window->logo,
01068      "website", "https://github.com/jburguete/mpcotool",
01069      "license-type", GTK_LICENSE_BSD, NULL);
01070 #if DEBUG_INTERFACE
01071   fprintf (stderr, "window_about: end\n");
01072 #endif
01073 }
01074
01080 void
01081 window_update_direction ()
01082 {
01083 #if DEBUG_INTERFACE
01084   fprintf (stderr, "window_update_direction: start\n");
01085 #endif
01086   gtk_widget_show (GTK_WIDGET (window->check_direction));
01087   if (gtk_toggle_button_get_active
01088       (GTK_TOGGLE_BUTTON (window->check_direction)))
01089     {
01090       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01091       gtk_widget_show (GTK_WIDGET (window->label_step));
01092       gtk_widget_show (GTK_WIDGET (window->spin_step));
```

```
01093     }
01094   switch (window_get_direction ())
01095     {
01096     case DIRECTION_METHOD_COORDINATES:
01097       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01098       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01099       break;
01100     default:
01101       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01102       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01103     }
01104 #if DEBUG_INTERFACE
01105   fprintf (stderr, "window_update_direction: end\n");
01106 #endif
01107 }
01108
01113 void
01114 window_update ()
01115 {
01116   unsigned int i;
01117 #if DEBUG_INTERFACE
01118   fprintf (stderr, "window_update: start\n");
01119 #endif
01120   gtk_widget_set_sensitive
01121     (GTK_WIDGET (window->button_evaluator),
01122      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01123                                    (window->check_evaluator)));
01124   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01125   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01126   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01127   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01128   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01129   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01130   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01131   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01132   gtk_widget_hide (GTK_WIDGET (window->label_population));
01133   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01134   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01135   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01136   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01137   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01138   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01139   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01140   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01142   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01144   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01146   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01147   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01148   gtk_widget_hide (GTK_WIDGET (window->label_step));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01150   gtk_widget_hide (GTK_WIDGET (window->label_p));
01151   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01152   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01153   switch (window_get_algorithm ())
01154     {
01155     case ALGORITHM_MONTE_CARLO:
01156       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01157       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01158       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01159       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01160       if (i > 1)
01161         {
01162           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01163           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01164           gtk_widget_show (GTK_WIDGET (window->label_bests));
01165           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01166         }
01167       window_update_direction ();
01168       break;
01169     case ALGORITHM_SWEEP:
01170       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01171       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01172       if (i > 1)
01173         {
01174           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01175           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01176           gtk_widget_show (GTK_WIDGET (window->label_bests));
01177           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01178         }
01179       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01180       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01181       gtk_widget_show (GTK_WIDGET (window->check_direction));
01182       window_update_direction ();
01183       break;
```

```
01184     default:
01185       gtk_widget_show (GTK_WIDGET (window->label_population));
01186       gtk_widget_show (GTK_WIDGET (window->spin_population));
01187       gtk_widget_show (GTK_WIDGET (window->label_generations));
01188       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01189       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01190       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01191       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01192       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01193       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01194       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01195       gtk_widget_show (GTK_WIDGET (window->label_bits));
01196       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01197     }
01198   gtk_widget_set_sensitive
01199     (GTK_WIDGET (window->button_remove_experiment),
      input->nexperiments > 1);
01200   gtk_widget_set_sensitive
01201     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
01202   for (i = 0; i < input->experiment->ninputs; ++i)
01203     {
01204       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01205       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01206       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01207       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01208       g_signal_handler_block
01209         (window->check_template[i], window->id_template[i]);
01210       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01211       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01212                                     (window->check_template[i]), 1);
01213       g_signal_handler_unblock (window->button_template[i],
01214                                 window->id_input[i]);
01215       g_signal_handler_unblock (window->check_template[i],
01216                                 window->id_template[i]);
01217     }
01218   if (i > 0)
01219     {
01220       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01221       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01222                                 gtk_toggle_button_get_active
01223                                 GTK_TOGGLE_BUTTON (window->check_template
01224                                                    [i - 1]));
01225     }
01226   if (i < MAX_NINPUTS)
01227     {
01228       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01229       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01230       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01231       gtk_widget_set_sensitive
01232         (GTK_WIDGET (window->button_template[i]),
01233          gtk_toggle_button_get_active
01234          GTK_TOGGLE_BUTTON (window->check_template[i]));
01235       g_signal_handler_block
01236         (window->check_template[i], window->id_template[i]);
01237       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01238       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01239                                     (window->check_template[i]), 0);
01240       g_signal_handler_unblock (window->button_template[i],
01241                                 window->id_input[i]);
01242       g_signal_handler_unblock (window->check_template[i],
01243                                 window->id_template[i]);
01244     }
01245   while (++i < MAX_NINPUTS)
01246     {
01247       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01248       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01249     }
01250   gtk_widget_set_sensitive
01251     (GTK_WIDGET (window->spin_minabs),
01252      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01253   gtk_widget_set_sensitive
01254     (GTK_WIDGET (window->spin_maxabs),
01255      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01256   if (window_get_norm () == ERROR_NORM_P)
01257     {
01258       gtk_widget_show (GTK_WIDGET (window->label_p));
01259       gtk_widget_show (GTK_WIDGET (window->spin_p));
01260     }
01261 #if DEBUG_INTERFACE
01262   fprintf (stderr, "window_update: end\n");
01263 #endif
01264 }
01265
01270 void
```

```
01271 window_set_algorithm ()
01272 {
01273   int i;
01274 #if DEBUG_INTERFACE
01275   fprintf (stderr, "window_set_algorithm: start\n");
01276 #endif
01277   i = window_get_algorithm ();
01278   switch (i)
01279     {
01280     case ALGORITHM_SWEEP:
01281       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01282       if (i < 0)
01283         i = 0;
01284       gtk_spin_button_set_value (window->spin_sweeps,
01285                                  (gdouble) input->variable[i].
      nsweeps);
01286       break;
01287     case ALGORITHM_GENETIC:
01288       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01289       if (i < 0)
01290         i = 0;
01291       gtk_spin_button_set_value (window->spin_bits,
01292                                  (gdouble) input->variable[i].nbits);
01293     }
01294   window_update ();
01295 #if DEBUG_INTERFACE
01296   fprintf (stderr, "window_set_algorithm: end\n");
01297 #endif
01298 }
01299
01304 void
01305 window_set_experiment ()
01306 {
01307   unsigned int i, j;
01308   char *buffer1, *buffer2;
01309 #if DEBUG_INTERFACE
01310   fprintf (stderr, "window_set_experiment: start\n");
01311 #endif
01312   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01313   gtk_spin_button_set_value (window->spin_weight, input->
      experiment[i].weight);
01314   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01315   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01316   g_free (buffer1);
01317   g_signal_handler_block
01318     (window->button_experiment, window->id_experiment_name);
01319   gtk_file_chooser_set_filename
01320     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01321   g_signal_handler_unblock
01322     (window->button_experiment, window->id_experiment_name);
01323   g_free (buffer2);
01324   for (j = 0; j < input->experiment->ninputs; ++j)
01325     {
01326       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01327       buffer2 =
01328         g_build_filename (input->directory, input->experiment[i].
      template[j],
01329                           NULL);
01330       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01331                                      (window->button_template[j]), buffer2);
01332       g_free (buffer2);
01333       g_signal_handler_unblock
01334         (window->button_template[j], window->id_input[j]);
01335     }
01336 #if DEBUG_INTERFACE
01337   fprintf (stderr, "window_set_experiment: end\n");
01338 #endif
01339 }
01340
01345 void
01346 window_remove_experiment ()
01347 {
01348   unsigned int i, j;
01349 #if DEBUG_INTERFACE
01350   fprintf (stderr, "window_remove_experiment: start\n");
01351 #endif
01352   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01353   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01354   gtk_combo_box_text_remove (window->combo_experiment, i);
01355   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01356   experiment_free (input->experiment + i, input->
      type);
01357   --input->nexperiments;
01358   for (j = i; j < input->nexperiments; ++j)
```

```
01359     memcpy (input->experiment + j, input->experiment + j + 1,
01360             sizeof (Experiment));
01361   j = input->nexperiments - 1;
01362   if (i > j)
01363     i = j;
01364   for (j = 0; j < input->experiment->ninputs; ++j)
01365     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01366   g_signal_handler_block
01367     (window->button_experiment, window->id_experiment_name);
01368   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01369   g_signal_handler_unblock
01370     (window->button_experiment, window->id_experiment_name);
01371   for (j = 0; j < input->experiment->ninputs; ++j)
01372     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01373   window_update ();
01374 #if DEBUG_INTERFACE
01375   fprintf (stderr, "window_remove_experiment: end\n");
01376 #endif
01377 }
01378
01383 void
01384 window_add_experiment ()
01385 {
01386   unsigned int i, j;
01387 #if DEBUG_INTERFACE
01388   fprintf (stderr, "window_add_experiment: start\n");
01389 #endif
01390   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01391   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01392   gtk_combo_box_text_insert_text
01393     (window->combo_experiment, i, input->experiment[i].
      name);
01394   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01395   input->experiment = (Experiment *) g_realloc
01396     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01397   for (j = input->nexperiments - 1; j > i; --j)
01398     memcpy (input->experiment + j + 1, input->experiment + j,
01399             sizeof (Experiment));
01400   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01401   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01402   if (input->type == INPUT_TYPE_XML)
01403     {
01404       input->experiment[j + 1].name
01405         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01406       for (j = 0; j < input->experiment->ninputs; ++j)
01407         input->experiment[i + 1].template[j]
01408           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      template[j]);
01409     }
01410   else
01411     {
01412       input->experiment[j + 1].name = g_strdup (input->
      experiment[j].name);
01413       for (j = 0; j < input->experiment->ninputs; ++j)
01414         input->experiment[i + 1].template[j]
01415           = g_strdup (input->experiment[i].template[j]);
01416     }
01417   ++input->nexperiments;
01418   for (j = 0; j < input->experiment->ninputs; ++j)
01419     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01420   g_signal_handler_block
01421     (window->button_experiment, window->id_experiment_name);
01422   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01423   g_signal_handler_unblock
01424     (window->button_experiment, window->id_experiment_name);
01425   for (j = 0; j < input->experiment->ninputs; ++j)
01426     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01427   window_update ();
01428 #if DEBUG_INTERFACE
01429   fprintf (stderr, "window_add_experiment: end\n");
01430 #endif
01431 }
01432
01437 void
01438 window_name_experiment ()
01439 {
01440   unsigned int i;
```

```
01441   char *buffer;
01442   GFile *file1, *file2;
01443 #if DEBUG_INTERFACE
01444   fprintf (stderr, "window_name_experiment: start\n");
01445 #endif
01446   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01447   file1
01448     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01449   file2 = g_file_new_for_path (input->directory);
01450   buffer = g_file_get_relative_path (file2, file1);
01451   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01452   gtk_combo_box_text_remove (window->combo_experiment, i);
01453   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01454   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01455   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01456   g_free (buffer);
01457   g_object_unref (file2);
01458   g_object_unref (file1);
01459 #if DEBUG_INTERFACE
01460   fprintf (stderr, "window_name_experiment: end\n");
01461 #endif
01462 }
01463
01468 void
01469 window_weight_experiment ()
01470 {
01471   unsigned int i;
01472 #if DEBUG_INTERFACE
01473   fprintf (stderr, "window_weight_experiment: start\n");
01474 #endif
01475   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01476   input->experiment[i].weight = gtk_spin_button_get_value (window->
      spin_weight);
01477 #if DEBUG_INTERFACE
01478   fprintf (stderr, "window_weight_experiment: end\n");
01479 #endif
01480 }
01481
01487 void
01488 window_inputs_experiment ()
01489 {
01490   unsigned int j;
01491 #if DEBUG_INTERFACE
01492   fprintf (stderr, "window_inputs_experiment: start\n");
01493 #endif
01494   j = input->experiment->ninputs - 1;
01495   if (j
01496       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01497                                         (window->check_template[j])))
01498     --input->experiment->ninputs;
01499   if (input->experiment->ninputs < MAX_NINPUTS
01500       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01501                                        (window->check_template[j])))
01502     ++input->experiment->ninputs;
01503   window_update ();
01504 #if DEBUG_INTERFACE
01505   fprintf (stderr, "window_inputs_experiment: end\n");
01506 #endif
01507 }
01508
01516 void
01517 window_template_experiment (void *data)
01518 {
01519   unsigned int i, j;
01520   char *buffer;
01521   GFile *file1, *file2;
01522 #if DEBUG_INTERFACE
01523   fprintf (stderr, "window_template_experiment: start\n");
01524 #endif
01525   i = (size_t) data;
01526   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01527   file1
01528     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01529   file2 = g_file_new_for_path (input->directory);
01530   buffer = g_file_get_relative_path (file2, file1);
01531   if (input->type == INPUT_TYPE_XML)
01532     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01533   else
01534     input->experiment[j].template[i] = g_strdup (buffer);
01535   g_free (buffer);
01536   g_object_unref (file2);
01537   g_object_unref (file1);
01538 #if DEBUG_INTERFACE
01539   fprintf (stderr, "window_template_experiment: end\n");
01540 #endif
```

```
01541 }
01542
01547 void
01548 window_set_variable ()
01549 {
01550   unsigned int i;
01551 #if DEBUG_INTERFACE
01552   fprintf (stderr, "window_set_variable: start\n");
01553 #endif
01554   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01555   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01556   gtk_entry_set_text (window->entry_variable, input->variable[i].
      name);
01557   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01558   gtk_spin_button_set_value (window->spin_min, input->variable[i].
      rangemin);
01559   gtk_spin_button_set_value (window->spin_max, input->variable[i].
      rangemax);
01560   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01561     {
01562       gtk_spin_button_set_value (window->spin_minabs,
01563                                  input->variable[i].rangeminabs);
01564       gtk_toggle_button_set_active
01565         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01566     }
01567   else
01568     {
01569       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01570       gtk_toggle_button_set_active
01571         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01572     }
01573   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01574     {
01575       gtk_spin_button_set_value (window->spin_maxabs,
01576                                  input->variable[i].rangemaxabs);
01577       gtk_toggle_button_set_active
01578         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01579     }
01580   else
01581     {
01582       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01583       gtk_toggle_button_set_active
01584         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01585     }
01586   gtk_spin_button_set_value (window->spin_precision,
01587                              input->variable[i].precision);
01588   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01589   if (input->nsteps)
01590     gtk_spin_button_set_value (window->spin_step, input->variable[i].
      step);
01591 #if DEBUG_INTERFACE
01592   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01593            input->variable[i].precision);
01594 #endif
01595   switch (window_get_algorithm ())
01596     {
01597     case ALGORITHM_SWEEP:
01598       gtk_spin_button_set_value (window->spin_sweeps,
01599                                  (gdouble) input->variable[i].
      nsweeps);
01600 #if DEBUG_INTERFACE
01601       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01602                input->variable[i].nsweeps);
01603 #endif
01604       break;
01605     case ALGORITHM_GENETIC:
01606       gtk_spin_button_set_value (window->spin_bits,
01607                                  (gdouble) input->variable[i].nbits);
01608 #if DEBUG_INTERFACE
01609       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01610                input->variable[i].nbits);
01611 #endif
01612       break;
01613     }
01614   window_update ();
01615 #if DEBUG_INTERFACE
01616   fprintf (stderr, "window_set_variable: end\n");
01617 #endif
01618 }
01619
01624 void
01625 window_remove_variable ()
01626 {
01627   unsigned int i, j;
```

```
01628 #if DEBUG_INTERFACE
01629   fprintf (stderr, "window_remove_variable: start\n");
01630 #endif
01631   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01632   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01633   gtk_combo_box_text_remove (window->combo_variable, i);
01634   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01635   xmlFree (input->variable[i].name);
01636   --input->nvariables;
01637   for (j = i; j < input->nvariables; ++j)
01638     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01639   j = input->nvariables - 1;
01640   if (i > j)
01641     i = j;
01642   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01643   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01644   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01645   window_update ();
01646 #if DEBUG_INTERFACE
01647   fprintf (stderr, "window_remove_variable: end\n");
01648 #endif
01649 }
01650
01655 void
01656 window_add_variable ()
01657 {
01658   unsigned int i, j;
01659 #if DEBUG_INTERFACE
01660   fprintf (stderr, "window_add_variable: start\n");
01661 #endif
01662   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01663   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01664   gtk_combo_box_text_insert_text (window->combo_variable, i,
01665                                   input->variable[i].name);
01666   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01667   input->variable = (Variable *) g_realloc
01668     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01669   for (j = input->nvariables - 1; j > i; --j)
01670     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01671   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01672   if (input->type == INPUT_TYPE_XML)
01673     input->variable[j + 1].name
01674       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01675   else
01676     input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
01677   ++input->nvariables;
01678   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01679   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01680   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01681   window_update ();
01682 #if DEBUG_INTERFACE
01683   fprintf (stderr, "window_add_variable: end\n");
01684 #endif
01685 }
01686
01691 void
01692 window_label_variable ()
01693 {
01694   unsigned int i;
01695   const char *buffer;
01696 #if DEBUG_INTERFACE
01697   fprintf (stderr, "window_label_variable: start\n");
01698 #endif
01699   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01700   buffer = gtk_entry_get_text (window->entry_variable);
01701   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01702   gtk_combo_box_text_remove (window->combo_variable, i);
01703   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01704   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01705   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01706 #if DEBUG_INTERFACE
01707   fprintf (stderr, "window_label_variable: end\n");
```

```
01708 #endif
01709 }
01710
01715 void
01716 window_precision_variable ()
01717 {
01718   unsigned int i;
01719 #if DEBUG_INTERFACE
01720   fprintf (stderr, "window_precision_variable: start\n");
01721 #endif
01722   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01723   input->variable[i].precision
01724     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01725   gtk_spin_button_set_digits (window->spin_min, input->variable[i].
      precision);
01726   gtk_spin_button_set_digits (window->spin_max, input->variable[i].
      precision);
01727   gtk_spin_button_set_digits (window->spin_minabs,
01728                               input->variable[i].precision);
01729   gtk_spin_button_set_digits (window->spin_maxabs,
01730                               input->variable[i].precision);
01731 #if DEBUG_INTERFACE
01732   fprintf (stderr, "window_precision_variable: end\n");
01733 #endif
01734 }
01735
01740 void
01741 window_rangemin_variable ()
01742 {
01743   unsigned int i;
01744 #if DEBUG_INTERFACE
01745   fprintf (stderr, "window_rangemin_variable: start\n");
01746 #endif
01747   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01748   input->variable[i].rangemin = gtk_spin_button_get_value (window->
      spin_min);
01749 #if DEBUG_INTERFACE
01750   fprintf (stderr, "window_rangemin_variable: end\n");
01751 #endif
01752 }
01753
01758 void
01759 window_rangemax_variable ()
01760 {
01761   unsigned int i;
01762 #if DEBUG_INTERFACE
01763   fprintf (stderr, "window_rangemax_variable: start\n");
01764 #endif
01765   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01766   input->variable[i].rangemax = gtk_spin_button_get_value (window->
      spin_max);
01767 #if DEBUG_INTERFACE
01768   fprintf (stderr, "window_rangemax_variable: end\n");
01769 #endif
01770 }
01771
01776 void
01777 window_rangeminabs_variable ()
01778 {
01779   unsigned int i;
01780 #if DEBUG_INTERFACE
01781   fprintf (stderr, "window_rangeminabs_variable: start\n");
01782 #endif
01783   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01784   input->variable[i].rangeminabs
01785     = gtk_spin_button_get_value (window->spin_minabs);
01786 #if DEBUG_INTERFACE
01787   fprintf (stderr, "window_rangeminabs_variable: end\n");
01788 #endif
01789 }
01790
01795 void
01796 window_rangemaxabs_variable ()
01797 {
01798   unsigned int i;
01799 #if DEBUG_INTERFACE
01800   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01801 #endif
01802   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01803   input->variable[i].rangemaxabs
01804     = gtk_spin_button_get_value (window->spin_maxabs);
01805 #if DEBUG_INTERFACE
01806   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01807 #endif
01808 }
01809
01814 void
```

```
01815 window_step_variable ()
01816 {
01817   unsigned int i;
01818 #if DEBUG_INTERFACE
01819   fprintf (stderr, "window_step_variable: start\n");
01820 #endif
01821   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01822   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01823 #if DEBUG_INTERFACE
01824   fprintf (stderr, "window_step_variable: end\n");
01825 #endif
01826 }
01827
01832 void
01833 window_update_variable ()
01834 {
01835   int i;
01836 #if DEBUG_INTERFACE
01837   fprintf (stderr, "window_update_variable: start\n");
01838 #endif
01839   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01840   if (i < 0)
01841     i = 0;
01842   switch (window_get_algorithm ())
01843     {
01844     case ALGORITHM_SWEEP:
01845       input->variable[i].nsweeps
01846         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01847 #if DEBUG_INTERFACE
01848       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01849                input->variable[i].nsweeps);
01850 #endif
01851       break;
01852     case ALGORITHM_GENETIC:
01853       input->variable[i].nbits
01854         = gtk_spin_button_get_value_as_int (window->spin_bits);
01855 #if DEBUG_INTERFACE
01856       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01857                input->variable[i].nbits);
01858 #endif
01859     }
01860 #if DEBUG_INTERFACE
01861   fprintf (stderr, "window_update_variable: end\n");
01862 #endif
01863 }
01864
01872 int
01873 window_read (char *filename)
01874 {
01875   unsigned int i;
01876   char *buffer;
01877 #if DEBUG_INTERFACE
01878   fprintf (stderr, "window_read: start\n");
01879 #endif
01880
01881   // Reading new input file
01882   input_free ();
01883   if (!input_open (filename))
01884     {
01885 #if DEBUG_INTERFACE
01886       fprintf (stderr, "window_read: end\n");
01887 #endif
01888       return 0;
01889     }
01890
01891   // Setting GTK+ widgets data
01892   gtk_entry_set_text (window->entry_result, input->result);
01893   gtk_entry_set_text (window->entry_variables, input->
      variables);
01894   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01895   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01896                                  (window->button_simulator), buffer);
01897   g_free (buffer);
01898   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01899                                 (size_t) input->evaluator);
01900   if (input->evaluator)
01901     {
01902       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01903       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01904                                      (window->button_evaluator), buffer);
01905       g_free (buffer);
01906     }
01907   gtk_toggle_button_set_active
01908     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
```

```
      algorithm]), TRUE);
01909   switch (input->algorithm)
01910     {
01911     case ALGORITHM_MONTE_CARLO:
01912       gtk_spin_button_set_value (window->spin_simulations,
01913                                  (gdouble) input->nsimulations);
01914     case ALGORITHM_SWEEP:
01915       gtk_spin_button_set_value (window->spin_iterations,
01916                                  (gdouble) input->niterations);
01917       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01918       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01919       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01920                                     (window->check_direction),
      input->nsteps);
01921       if (input->nsteps)
01922         {
01923           gtk_toggle_button_set_active
01924             (GTK_TOGGLE_BUTTON (window->button_direction
01925                                 [input->direction]), TRUE);
01926           gtk_spin_button_set_value (window->spin_steps,
01927                                      (gdouble) input->nsteps);
01928           gtk_spin_button_set_value (window->spin_relaxation,
01929                                      (gdouble) input->relaxation);
01930           switch (input->direction)
01931             {
01932             case DIRECTION_METHOD_RANDOM:
01933               gtk_spin_button_set_value (window->spin_estimates,
01934                                          (gdouble) input->nestimates);
01935             }
01936         }
01937       break;
01938     default:
01939       gtk_spin_button_set_value (window->spin_population,
01940                                  (gdouble) input->nsimulations);
01941       gtk_spin_button_set_value (window->spin_generations,
01942                                  (gdouble) input->niterations);
01943       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01944       gtk_spin_button_set_value (window->spin_reproduction,
01945                                  input->reproduction_ratio);
01946       gtk_spin_button_set_value (window->spin_adaptation,
01947                                  input->adaptation_ratio);
01948     }
01949   gtk_toggle_button_set_active
01950     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01951   gtk_spin_button_set_value (window->spin_p, input->p);
01952   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01953   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01954   g_signal_handler_block (window->button_experiment,
01955                           window->id_experiment_name);
01956   gtk_combo_box_text_remove_all (window->combo_experiment);
01957   for (i = 0; i < input->nexperiments; ++i)
01958     gtk_combo_box_text_append_text (window->combo_experiment,
01959                                     input->experiment[i].name);
01960   g_signal_handler_unblock
01961     (window->button_experiment, window->id_experiment_name);
01962   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01963   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01964   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01965   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01966   gtk_combo_box_text_remove_all (window->combo_variable);
01967   for (i = 0; i < input->nvariables; ++i)
01968     gtk_combo_box_text_append_text (window->combo_variable,
01969                                     input->variable[i].name);
01970   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01971   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01972   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01973   window_set_variable ();
01974   window_update ();
01975
01976 #if DEBUG_INTERFACE
01977   fprintf (stderr, "window_read: end\n");
01978 #endif
01979   return 1;
01980 }
01981
01986 void
01987 window_open ()
```

```
01988 {
01989   GtkFileChooserDialog *dlg;
01990   GtkFileFilter *filter;
01991   char *buffer, *directory, *name;
01992
01993 #if DEBUG_INTERFACE
01994   fprintf (stderr, "window_open: start\n");
01995 #endif
01996
01997   // Saving a backup of the current input file
01998   directory = g_strdup (input->directory);
01999   name = g_strdup (input->name);
02000
02001   // Opening dialog
02002   dlg = (GtkFileChooserDialog *)
02003     gtk_file_chooser_dialog_new (_("Open input file"),
02004                                  window->window,
02005                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02006                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
02007                                  _("_OK"), GTK_RESPONSE_OK, NULL);
02008
02009   // Adding XML filter
02010   filter = (GtkFileFilter *) gtk_file_filter_new ();
02011   gtk_file_filter_set_name (filter, "XML");
02012   gtk_file_filter_add_pattern (filter, "*.xml");
02013   gtk_file_filter_add_pattern (filter, "*.XML");
02014   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02015
02016   // Adding JSON filter
02017   filter = (GtkFileFilter *) gtk_file_filter_new ();
02018   gtk_file_filter_set_name (filter, "JSON");
02019   gtk_file_filter_add_pattern (filter, "*.json");
02020   gtk_file_filter_add_pattern (filter, "*.JSON");
02021   gtk_file_filter_add_pattern (filter, "*.js");
02022   gtk_file_filter_add_pattern (filter, "*.JS");
02023   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02024
02025   // If OK saving
02026   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02027     {
02028
02029       // Traying to open the input file
02030       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02031       if (!window_read (buffer))
02032         {
02033 #if DEBUG_INTERFACE
02034           fprintf (stderr, "window_open: error reading input file\n");
02035 #endif
02036           g_free (buffer);
02037
02038           // Reading backup file on error
02039           buffer = g_build_filename (directory, name, NULL);
02040           if (!input_open (buffer))
02041             {
02042
02043               // Closing on backup file reading error
02044 #if DEBUG_INTERFACE
02045               fprintf (stderr, "window_read: error reading backup file\n");
02046 #endif
02047               g_free (buffer);
02048               break;
02049             }
02050           g_free (buffer);
02051         }
02052       else
02053         {
02054           g_free (buffer);
02055           break;
02056         }
02057     }
02058
02059   // Freeing and closing
02060   g_free (name);
02061   g_free (directory);
02062   gtk_widget_destroy (GTK_WIDGET (dlg));
02063 #if DEBUG_INTERFACE
02064   fprintf (stderr, "window_open: end\n");
02065 #endif
02066 }
02067
02074 void
02075 window_new (GtkApplication * application)
02076 {
02077   unsigned int i;
02078   char *buffer, *buffer2, buffer3[64];
02079   char *label_algorithm[NALGORITHMS] = {
02080     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
```

```
02081   };
02082   char *tip_algorithm[NALGORITHMS] = {
02083     _("Monte-Carlo brute force algorithm"),
02084     _("Sweep brute force algorithm"),
02085     _("Genetic algorithm")
02086   };
02087   char *label_direction[NDIRECTIONS] = {
02088     _("_Coordinates descent"), _("_Random")
02089   };
02090   char *tip_direction[NDIRECTIONS] = {
02091     _("Coordinates direction estimate method"),
02092     _("Random direction estimate method")
02093   };
02094   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02095   char *tip_norm[NNORMS] = {
02096     _("Euclidean error norm (L2)"),
02097     _("Maximum error norm (L)"),
02098     _("P error norm (Lp)"),
02099     _("Taxicab error norm (L1)")
02100   };
02101
02102 #if DEBUG_INTERFACE
02103   fprintf (stderr, "window_new: start\n");
02104 #endif
02105
02106   // Creating the window
02107   window->window = main_window
02108     = (GtkWindow *) gtk_application_window_new (application);
02109
02110   // Finish when closing the window
02111   g_signal_connect_swapped (window->window, "delete-event",
02112                             G_CALLBACK (g_application_quit),
02113                             G_APPLICATION (application));
02114
02115   // Setting the window title
02116   gtk_window_set_title (window->window, "MPCOTool");
02117
02118   // Creating the open button
02119   window->button_open = (GtkToolButton *) gtk_tool_button_new
02120     (gtk_image_new_from_icon_name ("document-open",
02121                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02122   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02123
02124   // Creating the save button
02125   window->button_save = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("document-save",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02128   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02129                     NULL);
02130
02131   // Creating the run button
02132   window->button_run = (GtkToolButton *) gtk_tool_button_new
02133     (gtk_image_new_from_icon_name ("system-run",
02134                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02135   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02136
02137   // Creating the options button
02138   window->button_options = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("preferences-system",
02140                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02141   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02142
02143   // Creating the help button
02144   window->button_help = (GtkToolButton *) gtk_tool_button_new
02145     (gtk_image_new_from_icon_name ("help-browser",
02146                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02147   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02148
02149   // Creating the about button
02150   window->button_about = (GtkToolButton *) gtk_tool_button_new
02151     (gtk_image_new_from_icon_name ("help-about",
02152                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02153   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02154
02155   // Creating the exit button
02156   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02157     (gtk_image_new_from_icon_name ("application-exit",
02158                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02159   g_signal_connect_swapped (window->button_exit, "clicked",
02160                             G_CALLBACK (g_application_quit),
02161                             G_APPLICATION (application));
02162
02163   // Creating the buttons bar
02164   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02165   gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
```

```
02167   gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02169   gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02171   gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02173   gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02175   gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02177   gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02179   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02180
02181   // Creating the simulator program label and entry
02182   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02183   window->button_simulator = (GtkFileChooserButton *)
02184     gtk_file_chooser_button_new (_("Simulator program"),
02185                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02186   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02187                                _("Simulator program executable file"));
02188   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02189
02190   // Creating the evaluator program label and entry
02191   window->check_evaluator = (GtkCheckButton *)
02192     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02193   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02194   window->button_evaluator = (GtkFileChooserButton *)
02195     gtk_file_chooser_button_new (_("Evaluator program"),
02196                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->button_evaluator),
02199      _("Optional evaluator program executable file"));
02200
02201   // Creating the results files labels and entries
02202   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02203   window->entry_result = (GtkEntry *) gtk_entry_new ();
02204   gtk_widget_set_tooltip_text
02205     (GTK_WIDGET (window->entry_result), _("Best results file"));
02206   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02207   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02208   gtk_widget_set_tooltip_text
02209     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02210
02211   // Creating the files grid and attaching widgets
02212   window->grid_files = (GtkGrid *) gtk_grid_new ();
02213   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02214                    0, 0, 1, 1);
02215   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02216                    1, 0, 1, 1);
02217   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02218                    0, 1, 1, 1);
02219   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02220                    1, 1, 1, 1);
02221   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02222                    0, 2, 1, 1);
02223   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02224                    1, 2, 1, 1);
02225   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02226                    0, 3, 1, 1);
02227   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02228                    1, 3, 1, 1);
02229
02230   // Creating the algorithm properties
02231   window->label_simulations = (GtkLabel *) gtk_label_new
02232     (_("Simulations number"));
02233   window->spin_simulations
02234     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02235   gtk_widget_set_tooltip_text
02236     (GTK_WIDGET (window->spin_simulations),
02237      _("Number of simulations to perform for each iteration"));
02238   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02239   window->label_iterations = (GtkLabel *)
02240     gtk_label_new (_("Iterations number"));
02241   window->spin_iterations
02242     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
```

```
02245   g_signal_connect
02246     (window->spin_iterations, "value-changed", window_update, NULL);
02247   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02248   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02249   window->spin_tolerance =
02250     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02251   gtk_widget_set_tooltip_text
02252     (GTK_WIDGET (window->spin_tolerance),
02253      _("Tolerance to set the variable interval on the next iteration"));
02254   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02255   window->spin_bests
02256     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02257   gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_bests),
02259      _("Number of best simulations used to set the variable interval "
02260        "on the next iteration"));
02261   window->label_population
02262     = (GtkLabel *) gtk_label_new (_("Population number"));
02263   window->spin_population
02264     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02265   gtk_widget_set_tooltip_text
02266     (GTK_WIDGET (window->spin_population),
02267      _("Number of population for the genetic algorithm"));
02268   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02269   window->label_generations
02270     = (GtkLabel *) gtk_label_new (_("Generations number"));
02271   window->spin_generations
02272     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02273   gtk_widget_set_tooltip_text
02274     (GTK_WIDGET (window->spin_generations),
02275      _("Number of generations for the genetic algorithm"));
02276   window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02277   window->spin_mutation
02278     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279   gtk_widget_set_tooltip_text
02280     (GTK_WIDGET (window->spin_mutation),
02281      _("Ratio of mutation for the genetic algorithm"));
02282   window->label_reproduction
02283     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02284   window->spin_reproduction
02285     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286   gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_reproduction),
02288      _("Ratio of reproduction for the genetic algorithm"));
02289   window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02290   window->spin_adaptation
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292   gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_adaptation),
02294      _("Ratio of adaptation for the genetic algorithm"));
02295   window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02296   window->spin_threshold = (GtkSpinButton *)
02297     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02298                                     precision[DEFAULT_PRECISION]);
02299   gtk_widget_set_tooltip_text
02300     (GTK_WIDGET (window->spin_threshold),
02301      _("Threshold in the objective function to finish the simulations"));
02302   window->scrolled_threshold =
02303     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02304   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02305                      GTK_WIDGET (window->spin_threshold));
02306 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02307 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02308 //                         GTK_ALIGN_FILL);
02309
02310   // Creating the direction search method properties
02311   window->check_direction = (GtkCheckButton *)
02312     gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02313   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02314   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02315   window->button_direction[0] = (GtkRadioButton *)
02316     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02317   gtk_grid_attach (window->grid_direction,
02318                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02319   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02320                     NULL);
02321   for (i = 0; ++i < NDIRECTIONS;)
02322     {
02323       window->button_direction[i] = (GtkRadioButton *)
02324         gtk_radio_button_new_with_mnemonic
02325         (gtk_radio_button_get_group (window->button_direction[0]),
02326          label_direction[i]);
02327       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02328                                    tip_direction[i]);
02329       gtk_grid_attach (window->grid_direction,
```

```
02330                       GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02331       g_signal_connect (window->button_direction[i], "clicked",
02332                         window_update, NULL);
02333     }
02334   window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02335   window->spin_steps = (GtkSpinButton *)
02336     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02337   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02338   window->label_estimates
02339     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02340   window->spin_estimates = (GtkSpinButton *)
02341     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02342   window->label_relaxation
02343     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02344   window->spin_relaxation = (GtkSpinButton *)
02345     gtk_spin_button_new_with_range (0., 2., 0.001);
02346   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02347                    0, NDIRECTIONS, 1, 1);
02348   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02349                    1, NDIRECTIONS, 1, 1);
02350   gtk_grid_attach (window->grid_direction,
02351                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02352                    1, 1);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02355                    1);
02356   gtk_grid_attach (window->grid_direction,
02357                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02358                    1, 1);
02359   gtk_grid_attach (window->grid_direction,
02360                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02361                    1, 1);
02362
02363   // Creating the array of algorithms
02364   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02365   window->button_algorithm[0] = (GtkRadioButton *)
02366     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02367   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02368                                tip_algorithm[0]);
02369   gtk_grid_attach (window->grid_algorithm,
02370                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02371   g_signal_connect (window->button_algorithm[0], "clicked",
02372                     window_set_algorithm, NULL);
02373   for (i = 0; ++i < NALGORITHMS;)
02374     {
02375       window->button_algorithm[i] = (GtkRadioButton *)
02376         gtk_radio_button_new_with_mnemonic
02377         (gtk_radio_button_get_group (window->button_algorithm[0]),
02378          label_algorithm[i]);
02379       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02380                                    tip_algorithm[i]);
02381       gtk_grid_attach (window->grid_algorithm,
02382                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02383       g_signal_connect (window->button_algorithm[i], "clicked",
02384                         window_set_algorithm, NULL);
02385     }
02386   gtk_grid_attach (window->grid_algorithm,
02387                    GTK_WIDGET (window->label_simulations), 0,
02388                    NALGORITHMS, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm,
02390                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm,
02392                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02393                    1, 1);
02394   gtk_grid_attach (window->grid_algorithm,
02395                    GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02396                    1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02399                    1, 1);
02400   gtk_grid_attach (window->grid_algorithm,
02401                    GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02402                    1);
02403   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_bests),
02404                    0, NALGORITHMS + 3, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_bests), 1,
02406                    NALGORITHMS + 3, 1, 1);
02407   gtk_grid_attach (window->grid_algorithm,
02408                    GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02409                    1, 1);
02410   gtk_grid_attach (window->grid_algorithm,
02411                    GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02412                    1, 1);
```

```
02413   gtk_grid_attach (window->grid_algorithm,
02414                    GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02415                    1, 1);
02416   gtk_grid_attach (window->grid_algorithm,
02417                    GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02418                    1, 1);
02419   gtk_grid_attach (window->grid_algorithm,
02420                    GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02421                    1);
02422   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      spin_mutation),
02423                    1, NALGORITHMS + 6, 1, 1);
02424   gtk_grid_attach (window->grid_algorithm,
02425                    GTK_WIDGET (window->label_reproduction), 0,
02426                    NALGORITHMS + 7, 1, 1);
02427   gtk_grid_attach (window->grid_algorithm,
02428                    GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02429                    1, 1);
02430   gtk_grid_attach (window->grid_algorithm,
02431                    GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02432                    1, 1);
02433   gtk_grid_attach (window->grid_algorithm,
02434                    GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02435                    1, 1);
02436   gtk_grid_attach (window->grid_algorithm,
02437                    GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02438                    2, 1);
02439   gtk_grid_attach (window->grid_algorithm,
02440                    GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02441                    2, 1);
02442   gtk_grid_attach (window->grid_algorithm,
02443                    GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02444                    1, 1);
02445   gtk_grid_attach (window->grid_algorithm,
02446                    GTK_WIDGET (window->scrolled_threshold), 1,
02447                    NALGORITHMS + 11, 1, 1);
02448   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02449   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02450                      GTK_WIDGET (window->grid_algorithm));
02451
02452   // Creating the variable widgets
02453   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02456   window->id_variable = g_signal_connect
02457     (window->combo_variable, "changed", window_set_variable, NULL);
02458   window->button_add_variable
02459     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02460                                                    GTK_ICON_SIZE_BUTTON);
02461   g_signal_connect
02462     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02465   window->button_remove_variable
02466     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02467                                                    GTK_ICON_SIZE_BUTTON);
02468   g_signal_connect
02469     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02470   gtk_widget_set_tooltip_text
02471     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02472   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02473   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02474   gtk_widget_set_tooltip_text
02475     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02476   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02477   window->id_variable_label = g_signal_connect
02478     (window->entry_variable, "changed", window_label_variable, NULL);
02479   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02480   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02481     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482   gtk_widget_set_tooltip_text
02483     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02484   window->scrolled_min
02485     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02487                      GTK_WIDGET (window->spin_min));
02488   g_signal_connect (window->spin_min, "value-changed",
02489                     window_rangemin_variable, NULL);
02490   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02491   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02492     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02495   window->scrolled_max
02496     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
```

```
02497   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02498                      GTK_WIDGET (window->spin_max));
02499   g_signal_connect (window->spin_max, "value-changed",
02500                     window_rangemax_variable, NULL);
02501   window->check_minabs = (GtkCheckButton *)
02502     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02503   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02504   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02505     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_minabs),
02508      _("Minimum allowed value of the variable"));
02509   window->scrolled_minabs
02510     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02511   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02512                      GTK_WIDGET (window->spin_minabs));
02513   g_signal_connect (window->spin_minabs, "value-changed",
02514                     window_rangeminabs_variable, NULL);
02515   window->check_maxabs = (GtkCheckButton *)
02516     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02517   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02518   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02519     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02520   gtk_widget_set_tooltip_text
02521     (GTK_WIDGET (window->spin_maxabs),
02522      _("Maximum allowed value of the variable"));
02523   window->scrolled_maxabs
02524     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02525   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02526                      GTK_WIDGET (window->spin_maxabs));
02527   g_signal_connect (window->spin_maxabs, "value-changed",
02528                     window_rangemaxabs_variable, NULL);
02529   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02530   window->spin_precision = (GtkSpinButton *)
02531     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_precision),
02534      _("Number of precision floating point digits\n"
02535        "0 is for integer numbers"));
02536   g_signal_connect (window->spin_precision, "value-changed",
02537                     window_precision_variable, NULL);
02538   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02539   window->spin_sweeps =
02540     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02541   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02542                                _("Number of steps sweeping the variable"));
02543   g_signal_connect (window->spin_sweeps, "value-changed",
02544                     window_update_variable, NULL);
02545   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02546   window->spin_bits
02547     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02548   gtk_widget_set_tooltip_text
02549     (GTK_WIDGET (window->spin_bits),
02550      _("Number of bits to encode the variable"));
02551   g_signal_connect
02552     (window->spin_bits, "value-changed", window_update_variable, NULL);
02553   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02554   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02555     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02556   gtk_widget_set_tooltip_text
02557     (GTK_WIDGET (window->spin_step),
02558      _("Initial step size for the direction search method"));
02559   window->scrolled_step
02560     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02561   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02562                      GTK_WIDGET (window->spin_step));
02563   g_signal_connect
02564     (window->spin_step, "value-changed", window_step_variable, NULL);
02565   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
```

```
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02592   gtk_grid_attach (window->grid_variable,
02593                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02594   gtk_grid_attach (window->grid_variable,
02595                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02596   gtk_grid_attach (window->grid_variable,
02597                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02598   gtk_grid_attach (window->grid_variable,
02599                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02600   gtk_grid_attach (window->grid_variable,
02601                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02602   gtk_grid_attach (window->grid_variable,
02603                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02604   gtk_grid_attach (window->grid_variable,
02605                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02608   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02609   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02610                      GTK_WIDGET (window->grid_variable));
02611
02612   // Creating the experiment widgets
02613   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02615                                _("Experiment selector"));
02616   window->id_experiment = g_signal_connect
02617     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
02618   window->button_add_experiment
02619     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02620                                                    GTK_ICON_SIZE_BUTTON);
02621   g_signal_connect
02622     (window->button_add_experiment, "clicked",
       window_add_experiment, NULL);
02623   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02624                                _("Add experiment"));
02625   window->button_remove_experiment
02626     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02627                                                    GTK_ICON_SIZE_BUTTON);
02628   g_signal_connect (window->button_remove_experiment, "clicked",
02629                     window_remove_experiment, NULL);
02630   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02631                                _("Remove experiment"));
02632   window->label_experiment
02633     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02634   window->button_experiment = (GtkFileChooserButton *)
02635     gtk_file_chooser_button_new (_("Experimental data file"),
02636                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02637   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02638                                _("Experimental data file"));
02639   window->id_experiment_name
02640     = g_signal_connect (window->button_experiment, "selection-changed",
02641                         window_name_experiment, NULL);
02642   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02643   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02644   window->spin_weight
02645     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02646   gtk_widget_set_tooltip_text
02647     (GTK_WIDGET (window->spin_weight),
02648      _("Weight factor to build the objective function"));
02649   g_signal_connect
02650     (window->spin_weight, "value-changed", window_weight_experiment,
       NULL);
02651   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02652   gtk_grid_attach (window->grid_experiment,
02653                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02654   gtk_grid_attach (window->grid_experiment,
02655                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02656   gtk_grid_attach (window->grid_experiment,
02657                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02658   gtk_grid_attach (window->grid_experiment,
02659                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02660   gtk_grid_attach (window->grid_experiment,
02661                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02662   gtk_grid_attach (window->grid_experiment,
02663                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02664   gtk_grid_attach (window->grid_experiment,
02665                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02666   for (i = 0; i < MAX_NINPUTS; ++i)
02667     {
```

```
02668       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02669       window->check_template[i] = (GtkCheckButton *)
02670         gtk_check_button_new_with_label (buffer3);
02671       window->id_template[i]
02672         = g_signal_connect (window->check_template[i], "toggled",
02673                             window_inputs_experiment, NULL);
02674       gtk_grid_attach (window->grid_experiment,
02675                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02676       window->button_template[i] =
02677         (GtkFileChooserButton *)
02678         gtk_file_chooser_button_new (_("Input template"),
02679                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02680       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02681                                    _("Experimental input template file"));
02682       window->id_input[i] =
02683         g_signal_connect_swapped (window->button_template[i],
02684                                   "selection-changed",
02685                                   (void (*)) window_template_experiment,
02686                                   (void *) (size_t) i);
02687       gtk_grid_attach (window->grid_experiment,
02688                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02689     }
02690   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02691   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02692                      GTK_WIDGET (window->grid_experiment));
02693
02694   // Creating the error norm widgets
02695   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02696   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02697   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02698                      GTK_WIDGET (window->grid_norm));
02699   window->button_norm[0] = (GtkRadioButton *)
02700     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02702                                tip_norm[0]);
02703   gtk_grid_attach (window->grid_norm,
02704                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02705   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02706   for (i = 0; ++i < NNORMS;)
02707     {
02708       window->button_norm[i] = (GtkRadioButton *)
02709         gtk_radio_button_new_with_mnemonic
02710         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02711       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02712                                    tip_norm[i]);
02713       gtk_grid_attach (window->grid_norm,
02714                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02715       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02716     }
02717   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02718   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02719   window->spin_p =
02720     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02721                                                       G_MAXDOUBLE, 0.01);
02722   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02723                                _("P parameter for the P error norm"));
02724   window->scrolled_p =
02725     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02726   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02727                      GTK_WIDGET (window->spin_p));
02728   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02729   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02730   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02731                    1, 2, 1, 2);
02732
02733   // Creating the grid and attaching the widgets to the grid
02734   window->grid = (GtkGrid *) gtk_grid_new ();
02735   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02736   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02737   gtk_grid_attach (window->grid,
02738                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02739   gtk_grid_attach (window->grid,
02740                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02741   gtk_grid_attach (window->grid,
02742                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02743   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02744   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
02745
02746   // Setting the window logo
02747   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02748   gtk_window_set_icon (window->window, window->logo);
02749
02750   // Showing the window
02751   gtk_widget_show_all (GTK_WIDGET (window->window));
02752
```

```
02753   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02754 #if GTK_MINOR_VERSION >= 16
02755   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02756   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02757   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02758   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02759   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02760   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02761   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02762 #endif
02763
02764   // Reading initial example
02765   input_new ();
02766   buffer2 = g_get_current_dir ();
02767   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02768   g_free (buffer2);
02769   window_read (buffer);
02770   g_free (buffer);
02771
02772 #if DEBUG_INTERFACE
02773   fprintf (stderr, "window_new: start\n");
02774 #endif
02775 }
```

## 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 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 565 of file utils.c.

```
00566 {
00567   unsigned int i;
00568   for (i = 0; i < n; ++i)
00569     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570       break;
00571   return i;
00572 }
```

#### 4.13.2.2 void input_save ( char * *filename* )

Function to save the input file.

**Parameters**

| | |
|---|---|
| *filename* | Input file name. |

Definition at line 575 of file interface.c.

```
00576 {
00577   xmlDoc *doc;
00578   JsonGenerator *generator;
00579
00580 #if DEBUG_INTERFACE
00581   fprintf (stderr, "input_save: start\n");
00582 #endif
00583
00584   // Getting the input file directory
00585   input->name = g_path_get_basename (filename);
00586   input->directory = g_path_get_dirname (filename);
00587
00588   if (input->type == INPUT_TYPE_XML)
00589     {
00590       // Opening the input file
00591       doc = xmlNewDoc ((const xmlChar *) "1.0");
00592       input_save_xml (doc);
00593
00594       // Saving the XML file
00595       xmlSaveFormatFile (filename, doc, 1);
00596
00597       // Freeing memory
00598       xmlFreeDoc (doc);
00599     }
00600   else
00601     {
00602       // Opening the input file
00603       generator = json_generator_new ();
00604       json_generator_set_pretty (generator, TRUE);
00605       input_save_json (generator);
00606
00607       // Saving the JSON file
00608       json_generator_to_file (generator, filename, NULL);
00609
00610       // Freeing memory
00611       g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615   fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
```

Here is the call graph for this function:

#### 4.13.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

**Returns**

Stochastic algorithm number.

Definition at line 725 of file interface.c.

```
00726 {
00727   unsigned int i;
00728 #if DEBUG_INTERFACE
00729   fprintf (stderr, "window_get_algorithm: start\n");
00730 #endif
00731   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00732 #if DEBUG_INTERFACE
00733   fprintf (stderr, "window_get_algorithm: %u\n", i);
00734   fprintf (stderr, "window_get_algorithm: end\n");
00735 #endif
00736   return i;
00737 }
```

Here is the call graph for this function:

#### 4.13.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

**Returns**

Direction search method number.

Definition at line 745 of file interface.c.

```
00746 {
00747   unsigned int i;
00748 #if DEBUG_INTERFACE
00749   fprintf (stderr, "window_get_direction: start\n");
00750 #endif
00751   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00752 #if DEBUG_INTERFACE
00753   fprintf (stderr, "window_get_direction: %u\n", i);
00754   fprintf (stderr, "window_get_direction: end\n");
00755 #endif
00756   return i;
00757 }
```

Here is the call graph for this function:

#### 4.13.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

**Returns**

Norm method number.

Definition at line 765 of file interface.c.

```
00766 {
00767   unsigned int i;
00768 #if DEBUG_INTERFACE
00769   fprintf (stderr, "window_get_norm: start\n");
00770 #endif
00771   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00772 #if DEBUG_INTERFACE
00773   fprintf (stderr, "window_get_norm: %u\n", i);
00774   fprintf (stderr, "window_get_norm: end\n");
00775 #endif
00776   return i;
00777 }
```

Here is the call graph for this function:

#### 4.13.2.6 void window_new ( GtkApplication * *application* )

Function to open the main window.

**Parameters**

| | |
|---|---|
| *application* | GtkApplication struct. |

Definition at line 2075 of file interface.c.

```
02076 {
02077   unsigned int i;
02078   char *buffer, *buffer2, buffer3[64];
02079   char *label_algorithm[NALGORITHMS] = {
02080     "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02081   };
02082   char *tip_algorithm[NALGORITHMS] = {
02083     _("Monte-Carlo brute force algorithm"),
02084     _("Sweep brute force algorithm"),
02085     _("Genetic algorithm")
02086   };
02087   char *label_direction[NDIRECTIONS] = {
02088     _("_Coordinates descent"), _("_Random")
02089   };
02090   char *tip_direction[NDIRECTIONS] = {
02091     _("Coordinates direction estimate method"),
02092     _("Random direction estimate method")
02093   };
02094   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02095   char *tip_norm[NNORMS] = {
02096     _("Euclidean error norm (L2)"),
02097     _("Maximum error norm (L)"),
02098     _("P error norm (Lp)"),
02099     _("Taxicab error norm (L1)")
02100   };
02101
02102 #if DEBUG_INTERFACE
02103   fprintf (stderr, "window_new: start\n");
02104 #endif
```

```
02105
02106   // Creating the window
02107   window->window = main_window
02108     = (GtkWindow *) gtk_application_window_new (application);
02109
02110   // Finish when closing the window
02111   g_signal_connect_swapped (window->window, "delete-event",
02112                             G_CALLBACK (g_application_quit),
02113                             G_APPLICATION (application));
02114
02115   // Setting the window title
02116   gtk_window_set_title (window->window, "MPCOTool");
02117
02118   // Creating the open button
02119   window->button_open = (GtkToolButton *) gtk_tool_button_new
02120     (gtk_image_new_from_icon_name ("document-open",
02121                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02122   g_signal_connect (window->button_open, "clicked", window_open, NULL);
02123
02124   // Creating the save button
02125   window->button_save = (GtkToolButton *) gtk_tool_button_new
02126     (gtk_image_new_from_icon_name ("document-save",
02127                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02128   g_signal_connect (window->button_save, "clicked", (void (*))
      window_save,
02129                     NULL);
02130
02131   // Creating the run button
02132   window->button_run = (GtkToolButton *) gtk_tool_button_new
02133     (gtk_image_new_from_icon_name ("system-run",
02134                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02135   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02136
02137   // Creating the options button
02138   window->button_options = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("preferences-system",
02140                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Options"));
02141   g_signal_connect (window->button_options, "clicked",
      options_new, NULL);
02142
02143   // Creating the help button
02144   window->button_help = (GtkToolButton *) gtk_tool_button_new
02145     (gtk_image_new_from_icon_name ("help-browser",
02146                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02147   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02148
02149   // Creating the about button
02150   window->button_about = (GtkToolButton *) gtk_tool_button_new
02151     (gtk_image_new_from_icon_name ("help-about",
02152                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02153   g_signal_connect (window->button_about, "clicked",
      window_about, NULL);
02154
02155   // Creating the exit button
02156   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02157     (gtk_image_new_from_icon_name ("application-exit",
02158                                    GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02159   g_signal_connect_swapped (window->button_exit, "clicked",
02160                             G_CALLBACK (g_application_quit),
02161                             G_APPLICATION (application));
02162
02163   // Creating the buttons bar
02164   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02165   gtk_toolbar_insert
02166     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_open), 0);
02167   gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_save), 1);
02169   gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_run), 2);
02171   gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_options), 3);
02173   gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_help), 4);
02175   gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_about), 5);
02177   gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->
      button_exit), 6);
02179   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02180
02181   // Creating the simulator program label and entry
```

```
02182   window->label_simulator = (GtkLabel *) gtk_label_new (_("Simulator program"));
02183   window->button_simulator = (GtkFileChooserButton *)
02184     gtk_file_chooser_button_new (_("Simulator program"),
02185                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02186   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02187                                _("Simulator program executable file"));
02188   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02189
02190   // Creating the evaluator program label and entry
02191   window->check_evaluator = (GtkCheckButton *)
02192     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));
02193   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02194   window->button_evaluator = (GtkFileChooserButton *)
02195     gtk_file_chooser_button_new (_("Evaluator program"),
02196                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02197   gtk_widget_set_tooltip_text
02198     (GTK_WIDGET (window->button_evaluator),
02199      _("Optional evaluator program executable file"));
02200
02201   // Creating the results files labels and entries
02202   window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02203   window->entry_result = (GtkEntry *) gtk_entry_new ();
02204   gtk_widget_set_tooltip_text
02205     (GTK_WIDGET (window->entry_result), _("Best results file"));
02206   window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02207   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02208   gtk_widget_set_tooltip_text
02209     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02210
02211   // Creating the files grid and attaching widgets
02212   window->grid_files = (GtkGrid *) gtk_grid_new ();
02213   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02214                    0, 0, 1, 1);
02215   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02216                    1, 0, 1, 1);
02217   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02218                    0, 1, 1, 1);
02219   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02220                    1, 1, 1, 1);
02221   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02222                    0, 2, 1, 1);
02223   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02224                    1, 2, 1, 1);
02225   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02226                    0, 3, 1, 1);
02227   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02228                    1, 3, 1, 1);
02229
02230   // Creating the algorithm properties
02231   window->label_simulations = (GtkLabel *) gtk_label_new
02232     (_("Simulations number"));
02233   window->spin_simulations
02234     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02235   gtk_widget_set_tooltip_text
02236     (GTK_WIDGET (window->spin_simulations),
02237      _("Number of simulations to perform for each iteration"));
02238   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02239   window->label_iterations = (GtkLabel *)
02240     gtk_label_new (_("Iterations number"));
02241   window->spin_iterations
02242     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02243   gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02245   g_signal_connect
02246     (window->spin_iterations, "value-changed",
      window_update, NULL);
02247   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02248   window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02249   window->spin_tolerance =
02250     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02251   gtk_widget_set_tooltip_text
02252     (GTK_WIDGET (window->spin_tolerance),
02253      _("Tolerance to set the variable interval on the next iteration"));
02254   window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02255   window->spin_bests
02256     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02257   gtk_widget_set_tooltip_text
02258     (GTK_WIDGET (window->spin_bests),
```

```
02259       _("Number of best simulations used to set the variable interval "
02260         "on the next iteration"));
02261     window->label_population
02262       = (GtkLabel *) gtk_label_new (_("Population number"));
02263     window->spin_population
02264       = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02265     gtk_widget_set_tooltip_text
02266       (GTK_WIDGET (window->spin_population),
02267        _("Number of population for the genetic algorithm"));
02268     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02269     window->label_generations
02270       = (GtkLabel *) gtk_label_new (_("Generations number"));
02271     window->spin_generations
02272       = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02273     gtk_widget_set_tooltip_text
02274       (GTK_WIDGET (window->spin_generations),
02275        _("Number of generations for the genetic algorithm"));
02276     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02277     window->spin_mutation
02278       = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02279     gtk_widget_set_tooltip_text
02280       (GTK_WIDGET (window->spin_mutation),
02281        _("Ratio of mutation for the genetic algorithm"));
02282     window->label_reproduction
02283       = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02284     window->spin_reproduction
02285       = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286     gtk_widget_set_tooltip_text
02287       (GTK_WIDGET (window->spin_reproduction),
02288        _("Ratio of reproduction for the genetic algorithm"));
02289     window->label_adaptation = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02290     window->spin_adaptation
02291       = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292     gtk_widget_set_tooltip_text
02293       (GTK_WIDGET (window->spin_adaptation),
02294        _("Ratio of adaptation for the genetic algorithm"));
02295     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02296     window->spin_threshold = (GtkSpinButton *)
02297       gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02298                                       precision[DEFAULT_PRECISION]);
02299     gtk_widget_set_tooltip_text
02300       (GTK_WIDGET (window->spin_threshold),
02301        _("Threshold in the objective function to finish the simulations"));
02302     window->scrolled_threshold =
02303       (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02304     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02305                        GTK_WIDGET (window->spin_threshold));
02306 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02307 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02308 //                         GTK_ALIGN_FILL);
02309
02310     // Creating the direction search method properties
02311     window->check_direction = (GtkCheckButton *)
02312       gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02313     g_signal_connect (window->check_direction, "clicked",
    window_update, NULL);
02314     window->grid_direction = (GtkGrid *) gtk_grid_new ();
02315     window->button_direction[0] = (GtkRadioButton *)
02316       gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02317     gtk_grid_attach (window->grid_direction,
02318                      GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02319     g_signal_connect (window->button_direction[0], "clicked",
    window_update,
02320                       NULL);
02321     for (i = 0; ++i < NDIRECTIONS;)
02322       {
02323         window->button_direction[i] = (GtkRadioButton *)
02324           gtk_radio_button_new_with_mnemonic
02325           (gtk_radio_button_get_group (window->button_direction[0]),
02326            label_direction[i]);
02327         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02328                                      tip_direction[i]);
02329         gtk_grid_attach (window->grid_direction,
02330                          GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02331         g_signal_connect (window->button_direction[i], "clicked",
02332                           window_update, NULL);
02333       }
02334     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02335     window->spin_steps = (GtkSpinButton *)
02336       gtk_spin_button_new_with_range (1., 1.e12, 1.);
02337     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02338     window->label_estimates
02339       = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02340     window->spin_estimates = (GtkSpinButton *)
02341       gtk_spin_button_new_with_range (1., 1.e3, 1.);
02342     window->label_relaxation
02343       = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
```

```
02344   window->spin_relaxation = (GtkSpinButton *)
02345     gtk_spin_button_new_with_range (0., 2., 0.001);
02346   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->label_steps),
02347                    0, NDIRECTIONS, 1, 1);
02348   gtk_grid_attach (window->grid_direction, GTK_WIDGET (
      window->spin_steps),
02349                    1, NDIRECTIONS, 1, 1);
02350   gtk_grid_attach (window->grid_direction,
02351                    GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02352                    1, 1);
02353   gtk_grid_attach (window->grid_direction,
02354                    GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02355                    1);
02356   gtk_grid_attach (window->grid_direction,
02357                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02358                    1, 1);
02359   gtk_grid_attach (window->grid_direction,
02360                    GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02361                    1, 1);
02362
02363   // Creating the array of algorithms
02364   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02365   window->button_algorithm[0] = (GtkRadioButton *)
02366     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02367   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02368                                tip_algorithm[0]);
02369   gtk_grid_attach (window->grid_algorithm,
02370                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02371   g_signal_connect (window->button_algorithm[0], "clicked",
02372                     window_set_algorithm, NULL);
02373   for (i = 0; ++i < NALGORITHMS;)
02374     {
02375       window->button_algorithm[i] = (GtkRadioButton *)
02376         gtk_radio_button_new_with_mnemonic
02377         (gtk_radio_button_get_group (window->button_algorithm[0]),
02378          label_algorithm[i]);
02379       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02380                                    tip_algorithm[i]);
02381       gtk_grid_attach (window->grid_algorithm,
02382                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02383       g_signal_connect (window->button_algorithm[i], "clicked",
02384                         window_set_algorithm, NULL);
02385     }
02386   gtk_grid_attach (window->grid_algorithm,
02387                    GTK_WIDGET (window->label_simulations), 0,
02388                    NALGORITHMS, 1, 1);
02389   gtk_grid_attach (window->grid_algorithm,
02390                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02391   gtk_grid_attach (window->grid_algorithm,
02392                    GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02393                    1, 1);
02394   gtk_grid_attach (window->grid_algorithm,
02395                    GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02396                    1, 1);
02397   gtk_grid_attach (window->grid_algorithm,
02398                    GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02399                    1, 1);
02400   gtk_grid_attach (window->grid_algorithm,
02401                    GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02402                    1);
02403   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->label_bests),
02404                    0, NALGORITHMS + 3, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_bests), 1,
02406                    NALGORITHMS + 3, 1, 1);
02407   gtk_grid_attach (window->grid_algorithm,
02408                    GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02409                    1, 1);
02410   gtk_grid_attach (window->grid_algorithm,
02411                    GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02412                    1, 1);
02413   gtk_grid_attach (window->grid_algorithm,
02414                    GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02415                    1, 1);
02416   gtk_grid_attach (window->grid_algorithm,
02417                    GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02418                    1, 1);
02419   gtk_grid_attach (window->grid_algorithm,
02420                    GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02421                    1);
02422   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
      window->spin_mutation),
02423                    1, NALGORITHMS + 6, 1, 1);
02424   gtk_grid_attach (window->grid_algorithm,
02425                    GTK_WIDGET (window->label_reproduction), 0,
```

```
02426                     NALGORITHMS + 7, 1, 1);
02427   gtk_grid_attach (window->grid_algorithm,
02428                    GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02429                    1, 1);
02430   gtk_grid_attach (window->grid_algorithm,
02431                    GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02432                    1, 1);
02433   gtk_grid_attach (window->grid_algorithm,
02434                    GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02435                    1, 1);
02436   gtk_grid_attach (window->grid_algorithm,
02437                    GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02438                    2, 1);
02439   gtk_grid_attach (window->grid_algorithm,
02440                    GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02441                    2, 1);
02442   gtk_grid_attach (window->grid_algorithm,
02443                    GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02444                    1, 1);
02445   gtk_grid_attach (window->grid_algorithm,
02446                    GTK_WIDGET (window->scrolled_threshold), 1,
02447                    NALGORITHMS + 11, 1, 1);
02448   window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02449   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02450                      GTK_WIDGET (window->grid_algorithm));
02451
02452   // Creating the variable widgets
02453   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02454   gtk_widget_set_tooltip_text
02455     (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02456   window->id_variable = g_signal_connect
02457     (window->combo_variable, "changed", window_set_variable, NULL);
02458   window->button_add_variable
02459     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02460                                                    GTK_ICON_SIZE_BUTTON);
02461   g_signal_connect
02462     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02463   gtk_widget_set_tooltip_text
02464     (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02465   window->button_remove_variable
02466     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02467                                                    GTK_ICON_SIZE_BUTTON);
02468   g_signal_connect
02469     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02470   gtk_widget_set_tooltip_text
02471     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02472   window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02473   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02474   gtk_widget_set_tooltip_text
02475     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02476   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02477   window->id_variable_label = g_signal_connect
02478     (window->entry_variable, "changed",
      window_label_variable, NULL);
02479   window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02480   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02481     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02482   gtk_widget_set_tooltip_text
02483     (GTK_WIDGET (window->spin_min), _("Minimum initial value of the variable"));
02484   window->scrolled_min
02485     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02486   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02487                      GTK_WIDGET (window->spin_min));
02488   g_signal_connect (window->spin_min, "value-changed",
02489                     window_rangemin_variable, NULL);
02490   window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02491   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02492     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02493   gtk_widget_set_tooltip_text
02494     (GTK_WIDGET (window->spin_max), _("Maximum initial value of the variable"));
02495   window->scrolled_max
02496     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02497   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02498                      GTK_WIDGET (window->spin_max));
02499   g_signal_connect (window->spin_max, "value-changed",
02500                     window_rangemax_variable, NULL);
02501   window->check_minabs = (GtkCheckButton *)
02502     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02503   g_signal_connect (window->check_minabs, "toggled",
      window_update, NULL);
02504   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02505     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02506   gtk_widget_set_tooltip_text
02507     (GTK_WIDGET (window->spin_minabs),
02508      _("Minimum allowed value of the variable"));
```

```
02509   window->scrolled_minabs
02510     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02511   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02512                      GTK_WIDGET (window->spin_minabs));
02513   g_signal_connect (window->spin_minabs, "value-changed",
02514                     window_rangeminabs_variable, NULL);
02515   window->check_maxabs = (GtkCheckButton *)
02516     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02517   g_signal_connect (window->check_maxabs, "toggled",
      window_update, NULL);
02518   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02519     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02520   gtk_widget_set_tooltip_text
02521     (GTK_WIDGET (window->spin_maxabs),
02522      _("Maximum allowed value of the variable"));
02523   window->scrolled_maxabs
02524     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02525   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02526                      GTK_WIDGET (window->spin_maxabs));
02527   g_signal_connect (window->spin_maxabs, "value-changed",
02528                     window_rangemaxabs_variable, NULL);
02529   window->label_precision = (GtkLabel *) gtk_label_new (_("Precision digits"));
02530   window->spin_precision = (GtkSpinButton *)
02531     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02532   gtk_widget_set_tooltip_text
02533     (GTK_WIDGET (window->spin_precision),
02534      _("Number of precision floating point digits\n"
02535        "0 is for integer numbers"));
02536   g_signal_connect (window->spin_precision, "value-changed",
02537                     window_precision_variable, NULL);
02538   window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02539   window->spin_sweeps =
02540     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02541   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02542                                _("Number of steps sweeping the variable"));
02543   g_signal_connect (window->spin_sweeps, "value-changed",
02544                     window_update_variable, NULL);
02545   window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02546   window->spin_bits
02547     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02548   gtk_widget_set_tooltip_text
02549     (GTK_WIDGET (window->spin_bits),
02550      _("Number of bits to encode the variable"));
02551   g_signal_connect
02552     (window->spin_bits, "value-changed", window_update_variable, NULL)
    ;
02553   window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02554   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02555     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02556   gtk_widget_set_tooltip_text
02557     (GTK_WIDGET (window->spin_step),
02558      _("Initial step size for the direction search method"));
02559   window->scrolled_step
02560     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02561   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02562                      GTK_WIDGET (window->spin_step));
02563   g_signal_connect
02564     (window->spin_step, "value-changed", window_step_variable, NULL);
02565   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02566   gtk_grid_attach (window->grid_variable,
02567                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02568   gtk_grid_attach (window->grid_variable,
02569                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02570   gtk_grid_attach (window->grid_variable,
02571                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02572   gtk_grid_attach (window->grid_variable,
02573                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02574   gtk_grid_attach (window->grid_variable,
02575                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02576   gtk_grid_attach (window->grid_variable,
02577                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02578   gtk_grid_attach (window->grid_variable,
02579                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02580   gtk_grid_attach (window->grid_variable,
02581                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02582   gtk_grid_attach (window->grid_variable,
02583                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02584   gtk_grid_attach (window->grid_variable,
02585                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02586   gtk_grid_attach (window->grid_variable,
02587                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02588   gtk_grid_attach (window->grid_variable,
02589                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02590   gtk_grid_attach (window->grid_variable,
02591                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02592   gtk_grid_attach (window->grid_variable,
02593                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
```

```
02594   gtk_grid_attach (window->grid_variable,
02595                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02596   gtk_grid_attach (window->grid_variable,
02597                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02598   gtk_grid_attach (window->grid_variable,
02599                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02600   gtk_grid_attach (window->grid_variable,
02601                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02602   gtk_grid_attach (window->grid_variable,
02603                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02604   gtk_grid_attach (window->grid_variable,
02605                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02606   gtk_grid_attach (window->grid_variable,
02607                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02608   window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02609   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02610                      GTK_WIDGET (window->grid_variable));
02611
02612   // Creating the experiment widgets
02613   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02614   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02615                                _("Experiment selector"));
02616   window->id_experiment = g_signal_connect
02617     (window->combo_experiment, "changed",
      window_set_experiment, NULL);
02618   window->button_add_experiment
02619     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02620                                                    GTK_ICON_SIZE_BUTTON);
02621   g_signal_connect
02622     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02623   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02624                                _("Add experiment"));
02625   window->button_remove_experiment
02626     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02627                                                    GTK_ICON_SIZE_BUTTON);
02628   g_signal_connect (window->button_remove_experiment, "clicked",
02629                     window_remove_experiment, NULL);
02630   gtk_widget_set_tooltip_text (GTK_WIDGET (window->
      button_remove_experiment),
02631                                _("Remove experiment"));
02632   window->label_experiment
02633     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02634   window->button_experiment = (GtkFileChooserButton *)
02635     gtk_file_chooser_button_new (_("Experimental data file"),
02636                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02637   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02638                                _("Experimental data file"));
02639   window->id_experiment_name
02640     = g_signal_connect (window->button_experiment, "selection-changed",
02641                         window_name_experiment, NULL);
02642   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02643   window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02644   window->spin_weight
02645     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02646   gtk_widget_set_tooltip_text
02647     (GTK_WIDGET (window->spin_weight),
02648      _("Weight factor to build the objective function"));
02649   g_signal_connect
02650     (window->spin_weight, "value-changed",
      window_weight_experiment, NULL);
02651   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02652   gtk_grid_attach (window->grid_experiment,
02653                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02654   gtk_grid_attach (window->grid_experiment,
02655                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02656   gtk_grid_attach (window->grid_experiment,
02657                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1)
      ;
02658   gtk_grid_attach (window->grid_experiment,
02659                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02660   gtk_grid_attach (window->grid_experiment,
02661                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02662   gtk_grid_attach (window->grid_experiment,
02663                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02664   gtk_grid_attach (window->grid_experiment,
02665                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02666   for (i = 0; i < MAX_NINPUTS; ++i)
02667     {
02668       snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02669       window->check_template[i] = (GtkCheckButton *)
02670         gtk_check_button_new_with_label (buffer3);
02671       window->id_template[i]
02672         = g_signal_connect (window->check_template[i], "toggled",
02673                             window_inputs_experiment, NULL);
02674       gtk_grid_attach (window->grid_experiment,
02675                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
```

```
02676       window->button_template[i] =
02677         (GtkFileChooserButton *)
02678         gtk_file_chooser_button_new (_("Input template"),
02679                                      GTK_FILE_CHOOSER_ACTION_OPEN);
02680       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02681                                    _("Experimental input template file"));
02682       window->id_input[i] =
02683         g_signal_connect_swapped (window->button_template[i],
02684                                   "selection-changed",
02685                                   (void (*)) window_template_experiment,
02686                                   (void *) (size_t) i);
02687       gtk_grid_attach (window->grid_experiment,
02688                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02689     }
02690   window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02691   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02692                      GTK_WIDGET (window->grid_experiment));
02693
02694   // Creating the error norm widgets
02695   window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02696   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02697   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02698                      GTK_WIDGET (window->grid_norm));
02699   window->button_norm[0] = (GtkRadioButton *)
02700     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02701   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02702                                tip_norm[0]);
02703   gtk_grid_attach (window->grid_norm,
02704                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02705   g_signal_connect (window->button_norm[0], "clicked",
      window_update, NULL);
02706   for (i = 0; ++i < NNORMS;)
02707     {
02708       window->button_norm[i] = (GtkRadioButton *)
02709         gtk_radio_button_new_with_mnemonic
02710         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02711       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02712                                    tip_norm[i]);
02713       gtk_grid_attach (window->grid_norm,
02714                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02715       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02716     }
02717   window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02718   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      label_p), 1, 1, 1, 1);
02719   window->spin_p =
02720     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02721                                                       G_MAXDOUBLE, 0.01);
02722   gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02723                                _("P parameter for the P error norm"));
02724   window->scrolled_p =
02725     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02726   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02727                      GTK_WIDGET (window->spin_p));
02728   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02729   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02730   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
      scrolled_p),
02731                    1, 2, 1, 2);
02732
02733   // Creating the grid and attaching the widgets to the grid
02734   window->grid = (GtkGrid *) gtk_grid_new ();
02735   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02736   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02737   gtk_grid_attach (window->grid,
02738                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02739   gtk_grid_attach (window->grid,
02740                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02741   gtk_grid_attach (window->grid,
02742                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02743   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02744   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (
      window->grid));
02745
02746   // Setting the window logo
02747   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02748   gtk_window_set_icon (window->window, window->logo);
02749
02750   // Showing the window
02751   gtk_widget_show_all (GTK_WIDGET (window->window));
02752
02753   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02754 #if GTK_MINOR_VERSION >= 16
02755   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02756   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02757   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
```

```
02758   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02759   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02760   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02761   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02762 #endif
02763
02764   // Reading initial example
02765   input_new ();
02766   buffer2 = g_get_current_dir ();
02767   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02768   g_free (buffer2);
02769   window_read (buffer);
02770   g_free (buffer);
02771
02772 #if DEBUG_INTERFACE
02773   fprintf (stderr, "window_new: start\n");
02774 #endif
02775 }
```

Here is the call graph for this function:

#### 4.13.2.7 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 1873 of file interface.c.

```
01874 {
01875   unsigned int i;
01876   char *buffer;
01877 #if DEBUG_INTERFACE
01878   fprintf (stderr, "window_read: start\n");
01879 #endif
01880
01881   // Reading new input file
01882   input_free ();
01883   if (!input_open (filename))
01884     {
01885 #if DEBUG_INTERFACE
01886       fprintf (stderr, "window_read: end\n");
01887 #endif
01888       return 0;
01889     }
01890
01891   // Setting GTK+ widgets data
01892   gtk_entry_set_text (window->entry_result, input->result);
01893   gtk_entry_set_text (window->entry_variables, input->
      variables);
01894   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01895   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01896                                  (window->button_simulator), buffer);
01897   g_free (buffer);
01898   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01899                                 (size_t) input->evaluator);
01900   if (input->evaluator)
01901     {
01902       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01903       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01904                                      (window->button_evaluator), buffer);
01905       g_free (buffer);
```

```
01906     }
01907   gtk_toggle_button_set_active
01908     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01909   switch (input->algorithm)
01910     {
01911     case ALGORITHM_MONTE_CARLO:
01912       gtk_spin_button_set_value (window->spin_simulations,
01913                                  (gdouble) input->nsimulations);
01914     case ALGORITHM_SWEEP:
01915       gtk_spin_button_set_value (window->spin_iterations,
01916                                  (gdouble) input->niterations);
01917       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01918       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01919       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01920                                     (window->check_direction),
      input->nsteps);
01921       if (input->nsteps)
01922         {
01923           gtk_toggle_button_set_active
01924             (GTK_TOGGLE_BUTTON (window->button_direction
01925                                 [input->direction]), TRUE);
01926           gtk_spin_button_set_value (window->spin_steps,
01927                                      (gdouble) input->nsteps);
01928           gtk_spin_button_set_value (window->spin_relaxation,
01929                                      (gdouble) input->relaxation);
01930           switch (input->direction)
01931             {
01932             case DIRECTION_METHOD_RANDOM:
01933               gtk_spin_button_set_value (window->spin_estimates,
01934                                          (gdouble) input->nestimates);
01935             }
01936         }
01937       break;
01938     default:
01939       gtk_spin_button_set_value (window->spin_population,
01940                                  (gdouble) input->nsimulations);
01941       gtk_spin_button_set_value (window->spin_generations,
01942                                  (gdouble) input->niterations);
01943       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01944       gtk_spin_button_set_value (window->spin_reproduction,
01945                                  input->reproduction_ratio);
01946       gtk_spin_button_set_value (window->spin_adaptation,
01947                                  input->adaptation_ratio);
01948     }
01949   gtk_toggle_button_set_active
01950     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01951   gtk_spin_button_set_value (window->spin_p, input->p);
01952   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01953   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01954   g_signal_handler_block (window->button_experiment,
01955                           window->id_experiment_name);
01956   gtk_combo_box_text_remove_all (window->combo_experiment);
01957   for (i = 0; i < input->nexperiments; ++i)
01958     gtk_combo_box_text_append_text (window->combo_experiment,
01959                                     input->experiment[i].name);
01960   g_signal_handler_unblock
01961     (window->button_experiment, window->
      id_experiment_name);
01962   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01963   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01964   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01965   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01966   gtk_combo_box_text_remove_all (window->combo_variable);
01967   for (i = 0; i < input->nvariables; ++i)
01968     gtk_combo_box_text_append_text (window->combo_variable,
01969                                     input->variable[i].name);
01970   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01971   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01972   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01973   window_set_variable ();
01974   window_update ();
01975
01976 #if DEBUG_INTERFACE
01977   fprintf (stderr, "window_read: end\n");
01978 #endif
01979   return 1;
```

```
01980 }
```

Here is the call graph for this function:

#### 4.13.2.8 int window_save ( )

Function to save the input file.

**Returns**

1 on OK, 0 on Cancel.

Definition at line 818 of file interface.c.

```
00819 {
00820   GtkFileChooserDialog *dlg;
00821   GtkFileFilter *filter1, *filter2;
00822   char *buffer;
00823
00824 #if DEBUG_INTERFACE
00825   fprintf (stderr, "window_save: start\n");
00826 #endif
00827
00828   // Opening the saving dialog
00829   dlg = (GtkFileChooserDialog *)
00830     gtk_file_chooser_dialog_new (_("Save file"),
00831                                  window->window,
00832                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00833                                  _("_Cancel"), GTK_RESPONSE_CANCEL,
00834                                  _("_OK"), GTK_RESPONSE_OK, NULL);
00835   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00836   buffer = g_build_filename (input->directory, input->name, NULL);
00837   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00838   g_free (buffer);
00839
00840   // Adding XML filter
00841   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00842   gtk_file_filter_set_name (filter1, "XML");
00843   gtk_file_filter_add_pattern (filter1, "*.xml");
00844   gtk_file_filter_add_pattern (filter1, "*.XML");
00845   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00846
00847   // Adding JSON filter
00848   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter2, "JSON");
00850   gtk_file_filter_add_pattern (filter2, "*.json");
00851   gtk_file_filter_add_pattern (filter2, "*.JSON");
00852   gtk_file_filter_add_pattern (filter2, "*.js");
00853   gtk_file_filter_add_pattern (filter2, "*.JS");
00854   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00855
00856   if (input->type == INPUT_TYPE_XML)
00857     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00858   else
00859     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00860
00861   // If OK response then saving
00862   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00863     {
00864       // Setting input file type
00865       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00866       buffer = (char *) gtk_file_filter_get_name (filter1);
00867       if (!strcmp (buffer, "XML"))
00868         input->type = INPUT_TYPE_XML;
00869       else
00870         input->type = INPUT_TYPE_JSON;
00871
00872       // Adding properties to the root XML node
00873       input->simulator = gtk_file_chooser_get_filename
00874         (GTK_FILE_CHOOSER (window->button_simulator));
00875       if (gtk_toggle_button_get_active
00876           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00877         input->evaluator = gtk_file_chooser_get_filename
00878           (GTK_FILE_CHOOSER (window->button_evaluator));
00879       else
```

```
00880         input->evaluator = NULL;
00881       if (input->type == INPUT_TYPE_XML)
00882         {
00883           input->result
00884             = (char *) xmlStrdup ((const xmlChar *)
00885                                   gtk_entry_get_text (window->entry_result));
00886           input->variables
00887             = (char *) xmlStrdup ((const xmlChar *)
00888                                   gtk_entry_get_text (window->
      entry_variables));
00889         }
00890       else
00891         {
00892           input->result = g_strdup (gtk_entry_get_text (window->
      entry_result));
00893           input->variables =
00894             g_strdup (gtk_entry_get_text (window->entry_variables));
00895         }
00896
00897       // Setting the algorithm
00898       switch (window_get_algorithm ())
00899         {
00900         case ALGORITHM_MONTE_CARLO:
00901           input->algorithm = ALGORITHM_MONTE_CARLO;
00902           input->nsimulations
00903             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00904           input->niterations
00905             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00906           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00907           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00908           window_save_direction ();
00909           break;
00910         case ALGORITHM_SWEEP:
00911           input->algorithm = ALGORITHM_SWEEP;
00912           input->niterations
00913             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00914           input->tolerance = gtk_spin_button_get_value (window->
      spin_tolerance);
00915           input->nbest = gtk_spin_button_get_value_as_int (window->
      spin_bests);
00916           window_save_direction ();
00917           break;
00918         default:
00919           input->algorithm = ALGORITHM_GENETIC;
00920           input->nsimulations
00921             = gtk_spin_button_get_value_as_int (window->spin_population);
00922           input->niterations
00923             = gtk_spin_button_get_value_as_int (window->spin_generations);
00924           input->mutation_ratio
00925             = gtk_spin_button_get_value (window->spin_mutation);
00926           input->reproduction_ratio
00927             = gtk_spin_button_get_value (window->spin_reproduction);
00928           input->adaptation_ratio
00929             = gtk_spin_button_get_value (window->spin_adaptation);
00930           break;
00931         }
00932       input->norm = window_get_norm ();
00933       input->p = gtk_spin_button_get_value (window->spin_p);
00934       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00935
00936       // Saving the XML file
00937       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00938       input_save (buffer);
00939
00940       // Closing and freeing memory
00941       g_free (buffer);
00942       gtk_widget_destroy (GTK_WIDGET (dlg));
00943 #if DEBUG_INTERFACE
00944       fprintf (stderr, "window_save: end\n");
00945 #endif
00946       return 1;
00947     }
00948
00949   // Closing and freeing memory
00950   gtk_widget_destroy (GTK_WIDGET (dlg));
00951 #if DEBUG_INTERFACE
00952   fprintf (stderr, "window_save: end\n");
00953 #endif
00954   return 0;
00955 }
```

Here is the call graph for this function:

#### 4.13.2.9 void window_template_experiment ( void * *data* )

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

**Parameters**

| | |
|---|---|
| *data* | Callback data (i-th input template). |

Definition at line 1517 of file interface.c.

```
01518 {
01519   unsigned int i, j;
01520   char *buffer;
01521   GFile *file1, *file2;
01522 #if DEBUG_INTERFACE
01523   fprintf (stderr, "window_template_experiment: start\n");
01524 #endif
01525   i = (size_t) data;
01526   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01527   file1
01528     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01529   file2 = g_file_new_for_path (input->directory);
01530   buffer = g_file_get_relative_path (file2, file1);
01531   if (input->type == INPUT_TYPE_XML)
01532     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01533   else
01534     input->experiment[j].template[i] = g_strdup (buffer);
01535   g_free (buffer);
01536   g_object_unref (file2);
01537   g_object_unref (file1);
01538 #if DEBUG_INTERFACE
01539   fprintf (stderr, "window_template_experiment: end\n");
01540 #endif
01541 }
```

## 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 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
```

```
00048 typedef struct
00049 {
00050   GtkDialog *dialog;
00051   GtkGrid *grid;
00052   GtkLabel *label_seed;
00054   GtkSpinButton *spin_seed;
00056   GtkLabel *label_threads;
00057   GtkSpinButton *spin_threads;
00058   GtkLabel *label_direction;
00059   GtkSpinButton *spin_direction;
00061 } Options;
00062
00067 typedef struct
00068 {
00069   GtkDialog *dialog;
00070   GtkLabel *label;
00071   GtkSpinner *spinner;
00072   GtkGrid *grid;
00073 } Running;
00074
00079 typedef struct
00080 {
00081   GtkWindow *window;
00082   GtkGrid *grid;
00083   GtkToolbar *bar_buttons;
00084   GtkToolButton *button_open;
00085   GtkToolButton *button_save;
00086   GtkToolButton *button_run;
00087   GtkToolButton *button_options;
00088   GtkToolButton *button_help;
00089   GtkToolButton *button_about;
00090   GtkToolButton *button_exit;
00091   GtkGrid *grid_files;
00092   GtkLabel *label_simulator;
00093   GtkFileChooserButton *button_simulator;
00095   GtkCheckButton *check_evaluator;
00096   GtkFileChooserButton *button_evaluator;
00098   GtkLabel *label_result;
00099   GtkEntry *entry_result;
00100   GtkLabel *label_variables;
00101   GtkEntry *entry_variables;
00102   GtkFrame *frame_norm;
00103   GtkGrid *grid_norm;
00104   GtkRadioButton *button_norm[NNORMS];
00106   GtkLabel *label_p;
00107   GtkSpinButton *spin_p;
00108   GtkScrolledWindow *scrolled_p;
00110   GtkFrame *frame_algorithm;
00111   GtkGrid *grid_algorithm;
00112   GtkRadioButton *button_algorithm[NALGORITHMS];
00114   GtkLabel *label_simulations;
00115   GtkSpinButton *spin_simulations;
00117   GtkLabel *label_iterations;
00118   GtkSpinButton *spin_iterations;
00120   GtkLabel *label_tolerance;
00121   GtkSpinButton *spin_tolerance;
00122   GtkLabel *label_bests;
00123   GtkSpinButton *spin_bests;
00124   GtkLabel *label_population;
00125   GtkSpinButton *spin_population;
00127   GtkLabel *label_generations;
00128   GtkSpinButton *spin_generations;
00130   GtkLabel *label_mutation;
00131   GtkSpinButton *spin_mutation;
00132   GtkLabel *label_reproduction;
00133   GtkSpinButton *spin_reproduction;
00135   GtkLabel *label_adaptation;
00136   GtkSpinButton *spin_adaptation;
00138   GtkCheckButton *check_direction;
00140   GtkGrid *grid_direction;
00142   GtkRadioButton *button_direction[NDIRECTIONS];
00144   GtkLabel *label_steps;
00145   GtkSpinButton *spin_steps;
00146   GtkLabel *label_estimates;
00147   GtkSpinButton *spin_estimates;
00149   GtkLabel *label_relaxation;
00151   GtkSpinButton *spin_relaxation;
00153   GtkLabel *label_threshold;
00154   GtkSpinButton *spin_threshold;
00155   GtkScrolledWindow *scrolled_threshold;
00157   GtkFrame *frame_variable;
00158   GtkGrid *grid_variable;
00159   GtkComboBoxText *combo_variable;
00161   GtkButton *button_add_variable;
00162   GtkButton *button_remove_variable;
00163   GtkLabel *label_variable;
00164   GtkEntry *entry_variable;
```

```
00165   GtkLabel *label_min;
00166   GtkSpinButton *spin_min;
00167   GtkScrolledWindow *scrolled_min;
00168   GtkLabel *label_max;
00169   GtkSpinButton *spin_max;
00170   GtkScrolledWindow *scrolled_max;
00171   GtkCheckButton *check_minabs;
00172   GtkSpinButton *spin_minabs;
00173   GtkScrolledWindow *scrolled_minabs;
00174   GtkCheckButton *check_maxabs;
00175   GtkSpinButton *spin_maxabs;
00176   GtkScrolledWindow *scrolled_maxabs;
00177   GtkLabel *label_precision;
00178   GtkSpinButton *spin_precision;
00179   GtkLabel *label_sweeps;
00180   GtkSpinButton *spin_sweeps;
00181   GtkLabel *label_bits;
00182   GtkSpinButton *spin_bits;
00183   GtkLabel *label_step;
00184   GtkSpinButton *spin_step;
00185   GtkScrolledWindow *scrolled_step;
00186   GtkFrame *frame_experiment;
00187   GtkGrid *grid_experiment;
00188   GtkComboBoxText *combo_experiment;
00189   GtkButton *button_add_experiment;
00190   GtkButton *button_remove_experiment;
00191   GtkLabel *label_experiment;
00192   GtkFileChooserButton *button_experiment;
00194   GtkLabel *label_weight;
00195   GtkSpinButton *spin_weight;
00196   GtkCheckButton *check_template[MAX_NINPUTS];
00198   GtkFileChooserButton *button_template[MAX_NINPUTS];
00200   GdkPixbuf *logo;
00201   Experiment *experiment;
00202   Variable *variable;
00203   char *application_directory;
00204   gulong id_experiment;
00205   gulong id_experiment_name;
00206   gulong id_variable;
00207   gulong id_variable_label;
00208   gulong id_template[MAX_NINPUTS];
00210   gulong id_input[MAX_NINPUTS];
00212   unsigned int nexperiments;
00213   unsigned int nvariables;
00214 } Window;
00215
00216 // Global variables
00217 extern const char *logo[];
00218 extern Options options[1];
00219 extern Running running[1];
00220 extern Window window[1];
00221
00222 // Inline functions
00223 #if GTK_MINOR_VERSION < 10
00224 static inline GtkButton *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227   GtkButton *button;
00228   GtkImage *image;
00229   button = (GtkButton *) gtk_button_new ();
00230   image = (GtkImage *) gtk_image_new_from_icon_name (name, size);
00231   gtk_button_set_image (button, GTK_WIDGET (image));
00232   return button;
00233 }
00234 #endif
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);
00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_direction ();
00243 unsigned int window_get_norm ();
00244 void window_save_direction ();
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_direction ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
```

```
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new (GtkApplication * application);
00271
00272 #endif
```

## 4.15 main.c File Reference

Main source file.

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

Include dependency graph for main.c:

### Functions

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

### 4.15.1 Detailed Description

Main source file.

**Authors**

Javier Burguete and Borja Latorre.

**Copyright**



Definition in file main.c.

## 4.16 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #if HAVE_GTK
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #endif
00060 #include "genetic/genetic.h"
00061 #include "utils.h"
00062 #include "experiment.h"
00063 #include "variable.h"
00064 #include "input.h"
00065 #include "optimize.h"
00066 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069 #include "mpcotool.h"
00070
00071 int
00072 main (int argn, char **argc)
00073 {
00074 #if HAVE_GTK
00075   show_pending = process_pending;
00076 #endif
00077   return mpcotool (argn, argc);
00078 }
```

## 4.17 optimize.c File Reference

Source file to define the optimization functions.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
```

Include dependency graph for optimize.c:

## Macros

- #define DEBUG_OPTIMIZE 0

  *Macro to debug optimize functions.*
- #define RM "rm"

  *Macro to define the shell remove command.*

## Functions

- void optimize_input (unsigned int simulation, char *input, GMappedFile *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

Copyright 2012-2017, all rights reserved.

Definition in file optimize.c.

### 4.17.2 Function Documentation

#### 4.17.2.1 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 469 of file optimize.c.

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

```
00484       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00485       for (i = optimize->nsaveds; --i;)
00486         {
00487           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00488             {
00489               j = optimize->simulation_best[i];
00490               e = optimize->error_best[i];
00491               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00492               optimize->error_best[i] = optimize->
      error_best[i - 1];
00493               optimize->simulation_best[i - 1] = j;
00494               optimize->error_best[i - 1] = e;
00495             }
00496           else
00497             break;
00498         }
00499     }
00500 #if DEBUG_OPTIMIZE
00501   fprintf (stderr, "optimize_best: end\n");
00502 #endif
00503 }
```

#### 4.17.2.2 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 794 of file optimize.c.

```
00795 {
00796 #if DEBUG_OPTIMIZE
00797   fprintf (stderr, "optimize_best_direction: start\n");
00798   fprintf (stderr,
00799            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00800            simulation, value, optimize->error_best[0]);
00801 #endif
00802   if (value < optimize->error_best[0])
00803     {
00804       optimize->error_best[0] = value;
00805       optimize->simulation_best[0] = simulation;
00806 #if DEBUG_OPTIMIZE
00807       fprintf (stderr,
00808                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00809                simulation, value);
00810 #endif
00811     }
00812 #if DEBUG_OPTIMIZE
00813   fprintf (stderr, "optimize_best_direction: end\n");
00814 #endif
00815 }
```

#### 4.17.2.3 void optimize_direction_sequential ( unsigned int *simulation* )

Function to estimate the direction search sequentially.

**Parameters**

| | |
|---|---|
| *simulation* | Simulation number. |

Definition at line 824 of file optimize.c.

```
00825 {
00826   unsigned int i, j;
00827   double e;
00828 #if DEBUG_OPTIMIZE
00829   fprintf (stderr, "optimize_direction_sequential: start\n");
00830   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00831            "nend_direction=%u\n",
00832            optimize->nstart_direction, optimize->
     nend_direction);
00833 #endif
00834   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00835     {
00836       j = simulation + i;
00837       e = optimize_norm (j);
00838       optimize_best_direction (j, e);
00839       optimize_save_variables (j, e);
00840       if (e < optimize->threshold)
00841         {
00842           optimize->stop = 1;
00843           break;
00844         }
00845 #if DEBUG_OPTIMIZE
00846       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00847 #endif
00848     }
00849 #if DEBUG_OPTIMIZE
00850   fprintf (stderr, "optimize_direction_sequential: end\n");
00851 #endif
00852 }
```

Here is the call graph for this function:

#### 4.17.2.4 void * optimize_direction_thread ( ParallelData * *data* )

Function to estimate the direction search on a thread.

**Parameters**

| | |
|---|---|
| *data* | Function data. |

**Returns**

NULL

Definition at line 862 of file optimize.c.

```
00863 {
00864   unsigned int i, thread;
00865   double e;
00866 #if DEBUG_OPTIMIZE
00867   fprintf (stderr, "optimize_direction_thread: start\n");
00868 #endif
00869   thread = data->thread;
00870 #if DEBUG_OPTIMIZE
00871   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00872            thread,
00873            optimize->thread_direction[thread],
00874            optimize->thread_direction[thread + 1]);
00875 #endif
00876   for (i = optimize->thread_direction[thread];
00877        i < optimize->thread_direction[thread + 1]; ++i)
00878     {
00879       e = optimize_norm (i);
00880       g_mutex_lock (mutex);
00881       optimize_best_direction (i, e);
00882       optimize_save_variables (i, e);
00883       if (e < optimize->threshold)
```

```
00884         optimize->stop = 1;
00885       g_mutex_unlock (mutex);
00886       if (optimize->stop)
00887         break;
00888 #if DEBUG_OPTIMIZE
00889       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00890 #endif
00891     }
00892 #if DEBUG_OPTIMIZE
00893   fprintf (stderr, "optimize_direction_thread: end\n");
00894 #endif
00895   g_thread_exit (NULL);
00896   return NULL;
00897 }
```

Here is the call graph for this function:

#### 4.17.2.5 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 936 of file optimize.c.

```
00938 {
00939   double x;
00940 #if DEBUG_OPTIMIZE
00941   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00942 #endif
00943   x = optimize->direction[variable];
00944   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00945     {
00946       if (estimate & 1)
00947         x += optimize->step[variable];
00948       else
00949         x -= optimize->step[variable];
00950     }
00951 #if DEBUG_OPTIMIZE
00952   fprintf (stderr,
00953            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00954            variable, x);
00955   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00956 #endif
00957   return x;
00958 }
```

#### 4.17.2.6 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 909 of file optimize.c.

```
00911 {
00912   double x;
00913 #if DEBUG_OPTIMIZE
00914   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00915 #endif
00916   x = optimize->direction[variable]
00917     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00918 #if DEBUG_OPTIMIZE
00919   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00920            variable, x);
00921   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00922 #endif
00923   return x;
00924 }
```

#### 4.17.2.7 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 1103 of file optimize.c.

```
01104 {
01105   unsigned int j;
01106   double objective;
01107   char buffer[64];
01108 #if DEBUG_OPTIMIZE
01109   fprintf (stderr, "optimize_genetic_objective: start\n");
01110 #endif
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       optimize->value[entity->id * optimize->nvariables + j]
01114         = genetic_get_variable (entity, optimize->genetic_variable + j);
01115     }
01116   objective = optimize_norm (entity->id);
01117   g_mutex_lock (mutex);
01118   for (j = 0; j < optimize->nvariables; ++j)
01119     {
01120       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01121       fprintf (optimize->file_variables, buffer,
01122                genetic_get_variable (entity, optimize->genetic_variable + j));
01123     }
01124   fprintf (optimize->file_variables, "%.14le\n", objective);
01125   g_mutex_unlock (mutex);
01126 #if DEBUG_OPTIMIZE
01127   fprintf (stderr, "optimize_genetic_objective: end\n");
01128 #endif
01129   return objective;
01130 }
```

Here is the call graph for this function:

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

#### 4.17.2.9 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 592 of file optimize.c.

```
00594 {
00595   unsigned int i, j, k, s[optimize->nbest];
00596   double e[optimize->nbest];
00597 #if DEBUG_OPTIMIZE
00598   fprintf (stderr, "optimize_merge: start\n");
00599 #endif
00600   i = j = k = 0;
00601   do
00602     {
00603       if (i == optimize->nsaveds)
00604         {
00605           s[k] = simulation_best[j];
00606           e[k] = error_best[j];
00607           ++j;
00608           ++k;
00609           if (j == nsaveds)
00610             break;
00611         }
00612       else if (j == nsaveds)
00613         {
00614           s[k] = optimize->simulation_best[i];
00615           e[k] = optimize->error_best[i];
00616           ++i;
00617           ++k;
00618           if (i == optimize->nsaveds)
00619             break;
00620         }
00621       else if (optimize->error_best[i] > error_best[j])
00622         {
00623           s[k] = simulation_best[j];
00624           e[k] = error_best[j];
00625           ++j;
00626           ++k;
00627         }
00628       else
00629         {
00630           s[k] = optimize->simulation_best[i];
00631           e[k] = optimize->error_best[i];
00632           ++i;
00633           ++k;
00634         }
00635     }
00636   while (k < optimize->nbest);
00637   optimize->nsaveds = k;
00638   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00639   memcpy (optimize->error_best, e, k * sizeof (double));
00640 #if DEBUG_OPTIMIZE
00641   fprintf (stderr, "optimize_merge: end\n");
00642 #endif
00643 }
```

#### 4.17.2.10 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 301 of file optimize.c.

```
00302 {
00303   double e, ei;
00304   unsigned int i;
00305 #if DEBUG_OPTIMIZE
00306   fprintf (stderr, "optimize_norm_euclidian: start\n");
00307 #endif
00308   e = 0.;
00309   for (i = 0; i < optimize->nexperiments; ++i)
00310     {
00311       ei = optimize_parse (simulation, i);
00312       e += ei * ei;
00313     }
00314   e = sqrt (e);
00315 #if DEBUG_OPTIMIZE
00316   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00317   fprintf (stderr, "optimize_norm_euclidian: end\n");
00318 #endif
00319   return e;
00320 }
```

Here is the call graph for this function:

#### 4.17.2.11 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 330 of file optimize.c.

```
00331 {
00332   double e, ei;
00333   unsigned int i;
00334 #if DEBUG_OPTIMIZE
00335   fprintf (stderr, "optimize_norm_maximum: start\n");
00336 #endif
00337   e = 0.;
00338   for (i = 0; i < optimize->nexperiments; ++i)
00339     {
00340       ei = fabs (optimize_parse (simulation, i));
00341       e = fmax (e, ei);
00342     }
00343 #if DEBUG_OPTIMIZE
00344   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00345   fprintf (stderr, "optimize_norm_maximum: end\n");
00346 #endif
00347   return e;
00348 }
```

Here is the call graph for this function:

#### 4.17.2.12 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 358 of file optimize.c.

```
00359 {
00360   double e, ei;
00361   unsigned int i;
00362 #if DEBUG_OPTIMIZE
00363   fprintf (stderr, "optimize_norm_p: start\n");
00364 #endif
00365   e = 0.;
00366   for (i = 0; i < optimize->nexperiments; ++i)
00367     {
00368       ei = fabs (optimize_parse (simulation, i));
00369       e += pow (ei, optimize->p);
00370     }
00371   e = pow (e, 1. / optimize->p);
00372 #if DEBUG_OPTIMIZE
00373   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00374   fprintf (stderr, "optimize_norm_p: end\n");
00375 #endif
00376   return e;
00377 }
```

Here is the call graph for this function:

#### 4.17.2.13 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 387 of file optimize.c.

```
00388 {
00389   double e;
00390   unsigned int i;
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: start\n");
00393 #endif
00394   e = 0.;
00395   for (i = 0; i < optimize->nexperiments; ++i)
00396     e += fabs (optimize_parse (simulation, i));
00397 #if DEBUG_OPTIMIZE
00398   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00399   fprintf (stderr, "optimize_norm_taxicab: end\n");
00400 #endif
00401   return e;
00402 }
```

Here is the call graph for this function:

#### 4.17.2.14 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 190 of file optimize.c.

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

```
00250 #endif
00251       system (buffer);
00252       file_result = g_fopen (result, "r");
00253       e = atof (fgets (buffer, 512, file_result));
00254       fclose (file_result);
00255     }
00256   else
00257     {
00258 #if DEBUG_OPTIMIZE
00259       fprintf (stderr, "optimize_parse: output=%s\n", output);
00260 #endif
00261       strcpy (result, "");
00262       file_result = g_fopen (output, "r");
00263       e = atof (fgets (buffer, 512, file_result));
00264       fclose (file_result);
00265     }
00266
00267   // Removing files
00268 #if !DEBUG_OPTIMIZE
00269   for (i = 0; i < optimize->ninputs; ++i)
00270     {
00271       if (optimize->file[i][0])
00272         {
00273           snprintf (buffer, 512, RM " %s", &input[i][0]);
00274           system (buffer);
00275         }
00276     }
00277   snprintf (buffer, 512, RM " %s %s", output, result);
00278   system (buffer);
00279 #endif
00280
00281   // Processing pending events
00282   if (show_pending)
00283     show_pending ();
00284
00285 #if DEBUG_OPTIMIZE
00286   fprintf (stderr, "optimize_parse: end\n");
00287 #endif
00288
00289   // Returning the objective function
00290   return e * optimize->weight[experiment];
00291 }
```

Here is the call graph for this function:

#### 4.17.2.15 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 440 of file optimize.c.

```
00441 {
00442   unsigned int i;
00443   char buffer[64];
00444 #if DEBUG_OPTIMIZE
00445   fprintf (stderr, "optimize_save_variables: start\n");
00446 #endif
00447   for (i = 0; i < optimize->nvariables; ++i)
00448     {
00449       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00450       fprintf (optimize->file_variables, buffer,
00451                optimize->value[simulation * optimize->
      nvariables + i]);
00452     }
00453   fprintf (optimize->file_variables, "%.14le\n", error);
00454   fflush (optimize->file_variables);
00455 #if DEBUG_OPTIMIZE
00456   fprintf (stderr, "optimize_save_variables: end\n");
00457 #endif
00458 }
```

#### 4.17.2.16 void optimize_step_direction ( unsigned int *simulation* )

Function to do a step of the direction search method.

**Parameters**

| | |
|---|---|
| *simulation* | Simulation number. |

Definition at line 967 of file optimize.c.

```
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->
      nvariables;
00979 #if DEBUG_OPTIMIZE
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011               / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
01022             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
```

Here is the call graph for this function:

#### 4.17.2.17 void * optimize_thread ( ParallelData * *data* )

Function to optimize on a thread.

**Parameters**

| | |
|---|---|
| *data* | Function data. |

**Returns**

NULL

Definition at line 546 of file optimize.c.

```
00547 {
00548   unsigned int i, thread;
00549   double e;
00550 #if DEBUG_OPTIMIZE
00551   fprintf (stderr, "optimize_thread: start\n");
00552 #endif
00553   thread = data->thread;
00554 #if DEBUG_OPTIMIZE
00555   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00556            optimize->thread[thread], optimize->thread[thread + 1]);
00557 #endif
00558   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00559     {
00560       e = optimize_norm (i);
00561       g_mutex_lock (mutex);
00562       optimize_best (i, e);
00563       optimize_save_variables (i, e);
00564       if (e < optimize->threshold)
00565         optimize->stop = 1;
00566       g_mutex_unlock (mutex);
00567       if (optimize->stop)
00568         break;
00569 #if DEBUG_OPTIMIZE
00570       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00571 #endif
00572     }
00573 #if DEBUG_OPTIMIZE
00574   fprintf (stderr, "optimize_thread: end\n");
00575 #endif
00576   g_thread_exit (NULL);
00577   return NULL;
00578 }
```

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 2012-2017, 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, content);
00125 #endif
00126   file = g_fopen (input, "w");
00127
00128   // Parsing template
```

```
00129   for (i = 0; i < optimize->nvariables; ++i)
00130     {
00131 #if DEBUG_OPTIMIZE
00132       fprintf (stderr, "optimize_input: variable=%u\n", i);
00133 #endif
00134       snprintf (buffer, 32, "@variable%u@", i + 1);
00135       regex = g_regex_new (buffer, 0, 0, NULL);
00136       if (i == 0)
00137         {
00138           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00139                                              optimize->label[i], 0, NULL);
00140 #if DEBUG_OPTIMIZE
00141           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00142 #endif
00143         }
00144       else
00145         {
00146           length = strlen (buffer3);
00147           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00148                                              optimize->label[i], 0, NULL);
00149           g_free (buffer3);
00150         }
00151       g_regex_unref (regex);
00152       length = strlen (buffer2);
00153       snprintf (buffer, 32, "@value%u@", i + 1);
00154       regex = g_regex_new (buffer, 0, 0, NULL);
00155       snprintf (value, 32, format[optimize->precision[i]],
00156                 optimize->value[simulation * optimize->nvariables + i]);
00157
00158 #if DEBUG_OPTIMIZE
00159       fprintf (stderr, "optimize_input: value=%s\n", value);
00160 #endif
00161       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00162                                          0, NULL);
00163       g_free (buffer2);
00164       g_regex_unref (regex);
00165     }
00166
00167   // Saving input file
00168   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00169   g_free (buffer3);
00170   fclose (file);
00171
00172 optimize_input_end:
00173 #if DEBUG_OPTIMIZE
00174   fprintf (stderr, "optimize_input: end\n");
00175 #endif
00176   return;
00177 }
00178
00189 double
00190 optimize_parse (unsigned int simulation, unsigned int experiment)
00191 {
00192   unsigned int i;
00193   double e;
00194   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00195     *buffer3, *buffer4;
00196   FILE *file_result;
00197
00198 #if DEBUG_OPTIMIZE
00199   fprintf (stderr, "optimize_parse: start\n");
00200   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00201            simulation, experiment);
00202 #endif
00203
00204   // Opening input files
00205   for (i = 0; i < optimize->ninputs; ++i)
00206     {
00207       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00208 #if DEBUG_OPTIMIZE
00209       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00210 #endif
00211       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00212     }
00213   for (; i < MAX_NINPUTS; ++i)
00214     strcpy (&input[i][0], "");
00215 #if DEBUG_OPTIMIZE
00216   fprintf (stderr, "optimize_parse: parsing end\n");
00217 #endif
00218
00219   // Performing the simulation
00220   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221   buffer2 = g_path_get_dirname (optimize->simulator);
00222   buffer3 = g_path_get_basename (optimize->simulator);
00223   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225             buffer4, input[0], input[1], input[2], input[3], input[4],
```

```
00226             input[5], input[6], input[7], output);
00227   g_free (buffer4);
00228   g_free (buffer3);
00229   g_free (buffer2);
00230 #if DEBUG_OPTIMIZE
00231   fprintf (stderr, "optimize_parse: %s\n", buffer);
00232 #endif
00233   system (buffer);
00234
00235   // Checking the objective value function
00236   if (optimize->evaluator)
00237     {
00238       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00239       buffer2 = g_path_get_dirname (optimize->evaluator);
00240       buffer3 = g_path_get_basename (optimize->evaluator);
00241       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00242       snprintf (buffer, 512, "\"%s\" %s %s %s",
00243                 buffer4, output, optimize->experiment[experiment], result);
00244       g_free (buffer4);
00245       g_free (buffer3);
00246       g_free (buffer2);
00247 #if DEBUG_OPTIMIZE
00248       fprintf (stderr, "optimize_parse: %s\n", buffer);
00249       fprintf (stderr, "optimize_parse: result=%s\n", result);
00250 #endif
00251       system (buffer);
00252       file_result = g_fopen (result, "r");
00253       e = atof (fgets (buffer, 512, file_result));
00254       fclose (file_result);
00255     }
00256   else
00257     {
00258 #if DEBUG_OPTIMIZE
00259       fprintf (stderr, "optimize_parse: output=%s\n", output);
00260 #endif
00261       strcpy (result, "");
00262       file_result = g_fopen (output, "r");
00263       e = atof (fgets (buffer, 512, file_result));
00264       fclose (file_result);
00265     }
00266
00267   // Removing files
00268 #if !DEBUG_OPTIMIZE
00269   for (i = 0; i < optimize->ninputs; ++i)
00270     {
00271       if (optimize->file[i][0])
00272         {
00273           snprintf (buffer, 512, RM " %s", &input[i][0]);
00274           system (buffer);
00275         }
00276     }
00277   snprintf (buffer, 512, RM " %s %s", output, result);
00278   system (buffer);
00279 #endif
00280
00281   // Processing pending events
00282   if (show_pending)
00283     show_pending ();
00284
00285 #if DEBUG_OPTIMIZE
00286   fprintf (stderr, "optimize_parse: end\n");
00287 #endif
00288
00289   // Returning the objective function
00290   return e * optimize->weight[experiment];
00291 }
00292
00300 double
00301 optimize_norm_euclidian (unsigned int simulation)
00302 {
00303   double e, ei;
00304   unsigned int i;
00305 #if DEBUG_OPTIMIZE
00306   fprintf (stderr, "optimize_norm_euclidian: start\n");
00307 #endif
00308   e = 0.;
00309   for (i = 0; i < optimize->nexperiments; ++i)
00310     {
00311       ei = optimize_parse (simulation, i);
00312       e += ei * ei;
00313     }
00314   e = sqrt (e);
00315 #if DEBUG_OPTIMIZE
00316   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00317   fprintf (stderr, "optimize_norm_euclidian: end\n");
00318 #endif
00319   return e;
```

```
00320 }
00321
00329 double
00330 optimize_norm_maximum (unsigned int simulation)
00331 {
00332   double e, ei;
00333   unsigned int i;
00334 #if DEBUG_OPTIMIZE
00335   fprintf (stderr, "optimize_norm_maximum: start\n");
00336 #endif
00337   e = 0.;
00338   for (i = 0; i < optimize->nexperiments; ++i)
00339     {
00340       ei = fabs (optimize_parse (simulation, i));
00341       e = fmax (e, ei);
00342     }
00343 #if DEBUG_OPTIMIZE
00344   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00345   fprintf (stderr, "optimize_norm_maximum: end\n");
00346 #endif
00347   return e;
00348 }
00349
00357 double
00358 optimize_norm_p (unsigned int simulation)
00359 {
00360   double e, ei;
00361   unsigned int i;
00362 #if DEBUG_OPTIMIZE
00363   fprintf (stderr, "optimize_norm_p: start\n");
00364 #endif
00365   e = 0.;
00366   for (i = 0; i < optimize->nexperiments; ++i)
00367     {
00368       ei = fabs (optimize_parse (simulation, i));
00369       e += pow (ei, optimize->p);
00370     }
00371   e = pow (e, 1. / optimize->p);
00372 #if DEBUG_OPTIMIZE
00373   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00374   fprintf (stderr, "optimize_norm_p: end\n");
00375 #endif
00376   return e;
00377 }
00378
00386 double
00387 optimize_norm_taxicab (unsigned int simulation)
00388 {
00389   double e;
00390   unsigned int i;
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: start\n");
00393 #endif
00394   e = 0.;
00395   for (i = 0; i < optimize->nexperiments; ++i)
00396     e += fabs (optimize_parse (simulation, i));
00397 #if DEBUG_OPTIMIZE
00398   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00399   fprintf (stderr, "optimize_norm_taxicab: end\n");
00400 #endif
00401   return e;
00402 }
00403
00408 void
00409 optimize_print ()
00410 {
00411   unsigned int i;
00412   char buffer[512];
00413 #if HAVE_MPI
00414   if (optimize->mpi_rank)
00415     return;
00416 #endif
00417   printf ("%s\n", _("Best result"));
00418   fprintf (optimize->file_result, "%s\n", _("Best result"));
00419   printf ("error = %.15le\n", optimize->error_old[0]);
00420   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
00421   for (i = 0; i < optimize->nvariables; ++i)
00422     {
00423       snprintf (buffer, 512, "%s = %s\n",
00424                 optimize->label[i], format[optimize->precision[i]]);
00425       printf (buffer, optimize->value_old[i]);
00426       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00427     }
00428   fflush (optimize->file_result);
00429 }
00430
```

```
00439 void
00440 optimize_save_variables (unsigned int simulation, double error)
00441 {
00442   unsigned int i;
00443   char buffer[64];
00444 #if DEBUG_OPTIMIZE
00445   fprintf (stderr, "optimize_save_variables: start\n");
00446 #endif
00447   for (i = 0; i < optimize->nvariables; ++i)
00448     {
00449       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00450       fprintf (optimize->file_variables, buffer,
00451                optimize->value[simulation * optimize->nvariables + i]);
00452     }
00453   fprintf (optimize->file_variables, "%.14le\n", error);
00454   fflush (optimize->file_variables);
00455 #if DEBUG_OPTIMIZE
00456   fprintf (stderr, "optimize_save_variables: end\n");
00457 #endif
00458 }
00459
00468 void
00469 optimize_best (unsigned int simulation, double value)
00470 {
00471   unsigned int i, j;
00472   double e;
00473 #if DEBUG_OPTIMIZE
00474   fprintf (stderr, "optimize_best: start\n");
00475   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00476            optimize->nsaveds, optimize->nbest);
00477 #endif
00478   if (optimize->nsaveds < optimize->nbest
00479       || value < optimize->error_best[optimize->nsaveds - 1])
00480     {
00481       if (optimize->nsaveds < optimize->nbest)
00482         ++optimize->nsaveds;
00483       optimize->error_best[optimize->nsaveds - 1] = value;
00484       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00485       for (i = optimize->nsaveds; --i;)
00486         {
00487           if (optimize->error_best[i] < optimize->error_best[i - 1])
00488             {
00489               j = optimize->simulation_best[i];
00490               e = optimize->error_best[i];
00491               optimize->simulation_best[i] = optimize->
    simulation_best[i - 1];
00492               optimize->error_best[i] = optimize->error_best[i - 1];
00493               optimize->simulation_best[i - 1] = j;
00494               optimize->error_best[i - 1] = e;
00495             }
00496           else
00497             break;
00498         }
00499     }
00500 #if DEBUG_OPTIMIZE
00501   fprintf (stderr, "optimize_best: end\n");
00502 #endif
00503 }
00504
00509 void
00510 optimize_sequential ()
00511 {
00512   unsigned int i;
00513   double e;
00514 #if DEBUG_OPTIMIZE
00515   fprintf (stderr, "optimize_sequential: start\n");
00516   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00517            optimize->nstart, optimize->nend);
00518 #endif
00519   for (i = optimize->nstart; i < optimize->nend; ++i)
00520     {
00521       e = optimize_norm (i);
00522       optimize_best (i, e);
00523       optimize_save_variables (i, e);
00524       if (e < optimize->threshold)
00525         {
00526           optimize->stop = 1;
00527           break;
00528         }
00529 #if DEBUG_OPTIMIZE
00530       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00531 #endif
00532     }
00533 #if DEBUG_OPTIMIZE
00534   fprintf (stderr, "optimize_sequential: end\n");
00535 #endif
00536 }
```

```
00537
00545 void *
00546 optimize_thread (ParallelData * data)
00547 {
00548   unsigned int i, thread;
00549   double e;
00550 #if DEBUG_OPTIMIZE
00551   fprintf (stderr, "optimize_thread: start\n");
00552 #endif
00553   thread = data->thread;
00554 #if DEBUG_OPTIMIZE
00555   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00556            optimize->thread[thread], optimize->thread[thread + 1]);
00557 #endif
00558   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00559     {
00560       e = optimize_norm (i);
00561       g_mutex_lock (mutex);
00562       optimize_best (i, e);
00563       optimize_save_variables (i, e);
00564       if (e < optimize->threshold)
00565         optimize->stop = 1;
00566       g_mutex_unlock (mutex);
00567       if (optimize->stop)
00568         break;
00569 #if DEBUG_OPTIMIZE
00570       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00571 #endif
00572     }
00573 #if DEBUG_OPTIMIZE
00574   fprintf (stderr, "optimize_thread: end\n");
00575 #endif
00576   g_thread_exit (NULL);
00577   return NULL;
00578 }
00579
00591 void
00592 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00593                 double *error_best)
00594 {
00595   unsigned int i, j, k, s[optimize->nbest];
00596   double e[optimize->nbest];
00597 #if DEBUG_OPTIMIZE
00598   fprintf (stderr, "optimize_merge: start\n");
00599 #endif
00600   i = j = k = 0;
00601   do
00602     {
00603       if (i == optimize->nsaveds)
00604         {
00605           s[k] = simulation_best[j];
00606           e[k] = error_best[j];
00607           ++j;
00608           ++k;
00609           if (j == nsaveds)
00610             break;
00611         }
00612       else if (j == nsaveds)
00613         {
00614           s[k] = optimize->simulation_best[i];
00615           e[k] = optimize->error_best[i];
00616           ++i;
00617           ++k;
00618           if (i == optimize->nsaveds)
00619             break;
00620         }
00621       else if (optimize->error_best[i] > error_best[j])
00622         {
00623           s[k] = simulation_best[j];
00624           e[k] = error_best[j];
00625           ++j;
00626           ++k;
00627         }
00628       else
00629         {
00630           s[k] = optimize->simulation_best[i];
00631           e[k] = optimize->error_best[i];
00632           ++i;
00633           ++k;
00634         }
00635     }
00636   while (k < optimize->nbest);
00637   optimize->nsaveds = k;
00638   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00639   memcpy (optimize->error_best, e, k * sizeof (double));
00640 #if DEBUG_OPTIMIZE
00641   fprintf (stderr, "optimize_merge: end\n");
```

```
00642 #endif
00643 }
00644
00649 #if HAVE_MPI
00650 void
00651 optimize_synchronise ()
00652 {
00653   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00654   double error_best[optimize->nbest];
00655   MPI_Status mpi_stat;
00656 #if DEBUG_OPTIMIZE
00657   fprintf (stderr, "optimize_synchronise: start\n");
00658 #endif
00659   if (optimize->mpi_rank == 0)
00660     {
00661       for (i = 1; i < ntasks; ++i)
00662         {
00663           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00664           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00665                     MPI_COMM_WORLD, &mpi_stat);
00666           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00667                     MPI_COMM_WORLD, &mpi_stat);
00668           optimize_merge (nsaveds, simulation_best, error_best);
00669           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00670           if (stop)
00671             optimize->stop = 1;
00672         }
00673       for (i = 1; i < ntasks; ++i)
00674         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00675     }
00676   else
00677     {
00678       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00679       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00680                 MPI_COMM_WORLD);
00681       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00682                 MPI_COMM_WORLD);
00683       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00684       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00685       if (stop)
00686         optimize->stop = 1;
00687     }
00688 #if DEBUG_OPTIMIZE
00689   fprintf (stderr, "optimize_synchronise: end\n");
00690 #endif
00691 }
00692 #endif
00693
00698 void
00699 optimize_sweep ()
00700 {
00701   unsigned int i, j, k, l;
00702   double e;
00703   GThread *thread[nthreads];
00704   ParallelData data[nthreads];
00705 #if DEBUG_OPTIMIZE
00706   fprintf (stderr, "optimize_sweep: start\n");
00707 #endif
00708   for (i = 0; i < optimize->nsimulations; ++i)
00709     {
00710       k = i;
00711       for (j = 0; j < optimize->nvariables; ++j)
00712         {
00713           l = k % optimize->nsweeps[j];
00714           k /= optimize->nsweeps[j];
00715           e = optimize->rangemin[j];
00716           if (optimize->nsweeps[j] > 1)
00717             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00718               / (optimize->nsweeps[j] - 1);
00719           optimize->value[i * optimize->nvariables + j] = e;
00720         }
00721     }
00722   optimize->nsaveds = 0;
00723   if (nthreads <= 1)
00724     optimize_sequential ();
00725   else
00726     {
00727       for (i = 0; i < nthreads; ++i)
00728         {
00729           data[i].thread = i;
00730           thread[i] = 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] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00771         }
00772       for (i = 0; i < nthreads; ++i)
00773         g_thread_join (thread[i]);
00774     }
00775 #if HAVE_MPI
00776   // Communicating tasks results
00777   optimize_synchronise ();
00778 #endif
00779 #if DEBUG_OPTIMIZE
00780   fprintf (stderr, "optimize_MonteCarlo: end\n");
00781 #endif
00782 }
00783
00793 void
00794 optimize_best_direction (unsigned int simulation, double value)
00795 {
00796 #if DEBUG_OPTIMIZE
00797   fprintf (stderr, "optimize_best_direction: start\n");
00798   fprintf (stderr,
00799            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00800            simulation, value, optimize->error_best[0]);
00801 #endif
00802   if (value < optimize->error_best[0])
00803     {
00804       optimize->error_best[0] = value;
00805       optimize->simulation_best[0] = simulation;
00806 #if DEBUG_OPTIMIZE
00807       fprintf (stderr,
00808                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00809                simulation, value);
00810 #endif
00811     }
00812 #if DEBUG_OPTIMIZE
00813   fprintf (stderr, "optimize_best_direction: end\n");
00814 #endif
00815 }
00816
00823 void
00824 optimize_direction_sequential (unsigned int simulation)
00825 {
00826   unsigned int i, j;
00827   double e;
00828 #if DEBUG_OPTIMIZE
00829   fprintf (stderr, "optimize_direction_sequential: start\n");
00830   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00831            "nend_direction=%u\n",
00832            optimize->nstart_direction, optimize->nend_direction);
00833 #endif
00834   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00835     {
00836       j = simulation + i;
00837       e = optimize_norm (j);
00838       optimize_best_direction (j, e);
00839       optimize_save_variables (j, e);
00840       if (e < optimize->threshold)
00841         {
00842           optimize->stop = 1;
```

```
00843           break;
00844         }
00845 #if DEBUG_OPTIMIZE
00846       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00847 #endif
00848     }
00849 #if DEBUG_OPTIMIZE
00850   fprintf (stderr, "optimize_direction_sequential: end\n");
00851 #endif
00852 }
00853
00861 void *
00862 optimize_direction_thread (ParallelData * data)
00863 {
00864   unsigned int i, thread;
00865   double e;
00866 #if DEBUG_OPTIMIZE
00867   fprintf (stderr, "optimize_direction_thread: start\n");
00868 #endif
00869   thread = data->thread;
00870 #if DEBUG_OPTIMIZE
00871   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00872            thread,
00873            optimize->thread_direction[thread],
00874            optimize->thread_direction[thread + 1]);
00875 #endif
00876   for (i = optimize->thread_direction[thread];
00877        i < optimize->thread_direction[thread + 1]; ++i)
00878     {
00879       e = optimize_norm (i);
00880       g_mutex_lock (mutex);
00881       optimize_best_direction (i, e);
00882       optimize_save_variables (i, e);
00883       if (e < optimize->threshold)
00884         optimize->stop = 1;
00885       g_mutex_unlock (mutex);
00886       if (optimize->stop)
00887         break;
00888 #if DEBUG_OPTIMIZE
00889       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00890 #endif
00891     }
00892 #if DEBUG_OPTIMIZE
00893   fprintf (stderr, "optimize_direction_thread: end\n");
00894 #endif
00895   g_thread_exit (NULL);
00896   return NULL;
00897 }
00898
00908 double
00909 optimize_estimate_direction_random (unsigned int variable,
00910                                     unsigned int estimate)
00911 {
00912   double x;
00913 #if DEBUG_OPTIMIZE
00914   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00915 #endif
00916   x = optimize->direction[variable]
00917     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00918 #if DEBUG_OPTIMIZE
00919   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00920            variable, x);
00921   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00922 #endif
00923   return x;
00924 }
00925
00935 double
00936 optimize_estimate_direction_coordinates (unsigned int variable,
00937                                          unsigned int estimate)
00938 {
00939   double x;
00940 #if DEBUG_OPTIMIZE
00941   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00942 #endif
00943   x = optimize->direction[variable];
00944   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00945     {
00946       if (estimate & 1)
00947         x += optimize->step[variable];
00948       else
00949         x -= optimize->step[variable];
00950     }
00951 #if DEBUG_OPTIMIZE
00952   fprintf (stderr,
00953            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00954            variable, x);
```

```
00955   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00956 #endif
00957   return x;
00958 }
00959
00966 void
00967 optimize_step_direction (unsigned int simulation)
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->nvariables;
00979 #if DEBUG_OPTIMIZE
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j, i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011             / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
01022             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
01031
01036 void
01037 optimize_direction ()
01038 {
01039   unsigned int i, j, k, b, s, adjust;
01040 #if DEBUG_OPTIMIZE
01041   fprintf (stderr, "optimize_direction: start\n");
01042 #endif
01043   for (i = 0; i < optimize->nvariables; ++i)
01044     optimize->direction[i] = 0.;
01045   b = optimize->simulation_best[0] * optimize->nvariables;
01046   s = optimize->nsimulations;
01047   adjust = 1;
01048   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01049     {
01050 #if DEBUG_OPTIMIZE
```

```
01051       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01052                i, optimize->simulation_best[0]);
01053 #endif
01054       optimize_step_direction (s);
01055       k = optimize->simulation_best[0] * optimize->nvariables;
01056 #if DEBUG_OPTIMIZE
01057       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01058                i, optimize->simulation_best[0]);
01059 #endif
01060       if (k == b)
01061         {
01062           if (adjust)
01063             for (j = 0; j < optimize->nvariables; ++j)
01064               optimize->step[j] *= 0.5;
01065           for (j = 0; j < optimize->nvariables; ++j)
01066             optimize->direction[j] = 0.;
01067           adjust = 1;
01068         }
01069       else
01070         {
01071           for (j = 0; j < optimize->nvariables; ++j)
01072             {
01073 #if DEBUG_OPTIMIZE
01074               fprintf (stderr,
01075                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01076                        j, optimize->value[k + j], j, optimize->value[b + j]);
01077 #endif
01078               optimize->direction[j]
01079                 = (1. - optimize->relaxation) * optimize->direction[j]
01080                 + optimize->relaxation
01081                 * (optimize->value[k + j] - optimize->value[b + j]);
01082 #if DEBUG_OPTIMIZE
01083               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01084                        j, optimize->direction[j]);
01085 #endif
01086             }
01087           adjust = 0;
01088         }
01089     }
01090 #if DEBUG_OPTIMIZE
01091   fprintf (stderr, "optimize_direction: end\n");
01092 #endif
01093 }
01094
01102 double
01103 optimize_genetic_objective (Entity * entity)
01104 {
01105   unsigned int j;
01106   double objective;
01107   char buffer[64];
01108 #if DEBUG_OPTIMIZE
01109   fprintf (stderr, "optimize_genetic_objective: start\n");
01110 #endif
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       optimize->value[entity->id * optimize->nvariables + j]
01114         = genetic_get_variable (entity, optimize->genetic_variable + j);
01115     }
01116   objective = optimize_norm (entity->id);
01117   g_mutex_lock (mutex);
01118   for (j = 0; j < optimize->nvariables; ++j)
01119     {
01120       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01121       fprintf (optimize->file_variables, buffer,
01122                genetic_get_variable (entity, optimize->genetic_variable + j));
01123     }
01124   fprintf (optimize->file_variables, "%.14le\n", objective);
01125   g_mutex_unlock (mutex);
01126 #if DEBUG_OPTIMIZE
01127   fprintf (stderr, "optimize_genetic_objective: end\n");
01128 #endif
01129   return objective;
01130 }
01131
01136 void
01137 optimize_genetic ()
01138 {
01139   char *best_genome;
01140   double best_objective, *best_variable;
01141 #if DEBUG_OPTIMIZE
01142   fprintf (stderr, "optimize_genetic: start\n");
01143   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01144            nthreads);
01145   fprintf (stderr,
01146            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01147            optimize->nvariables, optimize->nsimulations, optimize->
      niterations);
```

```
01148   fprintf (stderr,
01149            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01150            optimize->mutation_ratio, optimize->reproduction_ratio,
01151            optimize->adaptation_ratio);
01152 #endif
01153   genetic_algorithm_default (optimize->nvariables,
01154                              optimize->genetic_variable,
01155                              optimize->nsimulations,
01156                              optimize->niterations,
01157                              optimize->mutation_ratio,
01158                              optimize->reproduction_ratio,
01159                              optimize->adaptation_ratio,
01160                              optimize->seed,
01161                              optimize->threshold,
01162                              &optimize_genetic_objective,
01163                              &best_genome, &best_variable, &best_objective);
01164 #if DEBUG_OPTIMIZE
01165   fprintf (stderr, "optimize_genetic: the best\n");
01166 #endif
01167   optimize->error_old = (double *) g_malloc (sizeof (double));
01168   optimize->value_old
01169     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01170   optimize->error_old[0] = best_objective;
01171   memcpy (optimize->value_old, best_variable,
01172           optimize->nvariables * sizeof (double));
01173   g_free (best_genome);
01174   g_free (best_variable);
01175   optimize_print ();
01176 #if DEBUG_OPTIMIZE
01177   fprintf (stderr, "optimize_genetic: end\n");
01178 #endif
01179 }
01180
01185 void
01186 optimize_save_old ()
01187 {
01188   unsigned int i, j;
01189 #if DEBUG_OPTIMIZE
01190   fprintf (stderr, "optimize_save_old: start\n");
01191   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01192 #endif
01193   memcpy (optimize->error_old, optimize->error_best,
01194           optimize->nbest * sizeof (double));
01195   for (i = 0; i < optimize->nbest; ++i)
01196     {
01197       j = optimize->simulation_best[i];
01198 #if DEBUG_OPTIMIZE
01199       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01200 #endif
01201       memcpy (optimize->value_old + i * optimize->nvariables,
01202               optimize->value + j * optimize->nvariables,
01203               optimize->nvariables * sizeof (double));
01204     }
01205 #if DEBUG_OPTIMIZE
01206   for (i = 0; i < optimize->nvariables; ++i)
01207     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01208              i, optimize->value_old[i]);
01209   fprintf (stderr, "optimize_save_old: end\n");
01210 #endif
01211 }
01212
01218 void
01219 optimize_merge_old ()
01220 {
01221   unsigned int i, j, k;
01222   double v[optimize->nbest * optimize->nvariables], e[optimize->
      nbest],
01223     *enew, *eold;
01224 #if DEBUG_OPTIMIZE
01225   fprintf (stderr, "optimize_merge_old: start\n");
01226 #endif
01227   enew = optimize->error_best;
01228   eold = optimize->error_old;
01229   i = j = k = 0;
01230   do
01231     {
01232       if (*enew < *eold)
01233         {
01234           memcpy (v + k * optimize->nvariables,
01235                   optimize->value
01236                   + optimize->simulation_best[i] * optimize->
      nvariables,
01237                   optimize->nvariables * sizeof (double));
01238           e[k] = *enew;
01239           ++k;
01240           ++enew;
01241           ++i;
```

```
01242         }
01243       else
01244         {
01245           memcpy (v + k * optimize->nvariables,
01246                   optimize->value_old + j * optimize->nvariables,
01247                   optimize->nvariables * sizeof (double));
01248           e[k] = *eold;
01249           ++k;
01250           ++eold;
01251           ++j;
01252         }
01253     }
01254   while (k < optimize->nbest);
01255   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01256   memcpy (optimize->error_old, e, k * sizeof (double));
01257 #if DEBUG_OPTIMIZE
01258   fprintf (stderr, "optimize_merge_old: end\n");
01259 #endif
01260 }
01261
01267 void
01268 optimize_refine ()
01269 {
01270   unsigned int i, j;
01271   double d;
01272 #if HAVE_MPI
01273   MPI_Status mpi_stat;
01274 #endif
01275 #if DEBUG_OPTIMIZE
01276   fprintf (stderr, "optimize_refine: start\n");
01277 #endif
01278 #if HAVE_MPI
01279   if (!optimize->mpi_rank)
01280     {
01281 #endif
01282       for (j = 0; j < optimize->nvariables; ++j)
01283         {
01284           optimize->rangemin[j] = optimize->rangemax[j]
01285             = optimize->value_old[j];
01286         }
01287       for (i = 0; ++i < optimize->nbest;)
01288         {
01289           for (j = 0; j < optimize->nvariables; ++j)
01290             {
01291               optimize->rangemin[j]
01292                 = fmin (optimize->rangemin[j],
01293                         optimize->value_old[i * optimize->nvariables + j]);
01294               optimize->rangemax[j]
01295                 = fmax (optimize->rangemax[j],
01296                         optimize->value_old[i * optimize->nvariables + j]);
01297             }
01298         }
01299       for (j = 0; j < optimize->nvariables; ++j)
01300         {
01301           d = optimize->tolerance
01302             * (optimize->rangemax[j] - optimize->rangemin[j]);
01303           switch (optimize->algorithm)
01304             {
01305             case ALGORITHM_MONTE_CARLO:
01306               d *= 0.5;
01307               break;
01308             default:
01309               if (optimize->nsweeps[j] > 1)
01310                 d /= optimize->nsweeps[j] - 1;
01311               else
01312                 d = 0.;
01313             }
01314           optimize->rangemin[j] -= d;
01315           optimize->rangemin[j]
01316             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01317           optimize->rangemax[j] += d;
01318           optimize->rangemax[j]
01319             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01320           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01321                   optimize->rangemin[j], optimize->rangemax[j]);
01322           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01323                    optimize->label[j], optimize->rangemin[j],
01324                    optimize->rangemax[j]);
01325         }
01326 #if HAVE_MPI
01327       for (i = 1; i < ntasks; ++i)
01328         {
01329           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01330                     1, MPI_COMM_WORLD);
01331           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01332                     1, MPI_COMM_WORLD);
01333         }
```

```
01334     }
01335   else
01336     {
01337       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01338                 MPI_COMM_WORLD, &mpi_stat);
01339       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01340                 MPI_COMM_WORLD, &mpi_stat);
01341     }
01342 #endif
01343 #if DEBUG_OPTIMIZE
01344   fprintf (stderr, "optimize_refine: end\n");
01345 #endif
01346 }
01347
01352 void
01353 optimize_step ()
01354 {
01355 #if DEBUG_OPTIMIZE
01356   fprintf (stderr, "optimize_step: start\n");
01357 #endif
01358   optimize_algorithm ();
01359   if (optimize->nsteps)
01360     optimize_direction ();
01361 #if DEBUG_OPTIMIZE
01362   fprintf (stderr, "optimize_step: end\n");
01363 #endif
01364 }
01365
01370 void
01371 optimize_iterate ()
01372 {
01373   unsigned int i;
01374 #if DEBUG_OPTIMIZE
01375   fprintf (stderr, "optimize_iterate: start\n");
01376 #endif
01377   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01378   optimize->value_old =
01379     (double *) g_malloc (optimize->nbest * optimize->nvariables *
01380                          sizeof (double));
01381   optimize_step ();
01382   optimize_save_old ();
01383   optimize_refine ();
01384   optimize_print ();
01385   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01386     {
01387       optimize_step ();
01388       optimize_merge_old ();
01389       optimize_refine ();
01390       optimize_print ();
01391     }
01392 #if DEBUG_OPTIMIZE
01393   fprintf (stderr, "optimize_iterate: end\n");
01394 #endif
01395 }
01396
01401 void
01402 optimize_free ()
01403 {
01404   unsigned int i, j;
01405 #if DEBUG_OPTIMIZE
01406   fprintf (stderr, "optimize_free: start\n");
01407 #endif
01408   for (j = 0; j < optimize->ninputs; ++j)
01409     {
01410       for (i = 0; i < optimize->nexperiments; ++i)
01411         g_mapped_file_unref (optimize->file[j][i]);
01412       g_free (optimize->file[j]);
01413     }
01414   g_free (optimize->error_old);
01415   g_free (optimize->value_old);
01416   g_free (optimize->value);
01417   g_free (optimize->genetic_variable);
01418 #if DEBUG_OPTIMIZE
01419   fprintf (stderr, "optimize_free: end\n");
01420 #endif
01421 }
01422
01427 void
01428 optimize_open ()
01429 {
01430   GTimeZone *tz;
01431   GDateTime *t0, *t;
01432   unsigned int i, j;
01433
01434 #if DEBUG_OPTIMIZE
01435   char *buffer;
01436   fprintf (stderr, "optimize_open: start\n");
```

```
01437 #endif
01438
01439   // Getting initial time
01440 #if DEBUG_OPTIMIZE
01441   fprintf (stderr, "optimize_open: getting initial time\n");
01442 #endif
01443   tz = g_time_zone_new_utc ();
01444   t0 = g_date_time_new_now (tz);
01445
01446   // Obtaining and initing the pseudo-random numbers generator seed
01447 #if DEBUG_OPTIMIZE
01448   fprintf (stderr, "optimize_open: getting initial seed\n");
01449 #endif
01450   if (optimize->seed == DEFAULT_RANDOM_SEED)
01451     optimize->seed = input->seed;
01452   gsl_rng_set (optimize->rng, optimize->seed);
01453
01454   // Replacing the working directory
01455 #if DEBUG_OPTIMIZE
01456   fprintf (stderr, "optimize_open: replacing the working directory\n");
01457 #endif
01458   g_chdir (input->directory);
01459
01460   // Getting results file names
01461   optimize->result = input->result;
01462   optimize->variables = input->variables;
01463
01464   // Obtaining the simulator file
01465   optimize->simulator = input->simulator;
01466
01467   // Obtaining the evaluator file
01468   optimize->evaluator = input->evaluator;
01469
01470   // Reading the algorithm
01471   optimize->algorithm = input->algorithm;
01472   switch (optimize->algorithm)
01473     {
01474     case ALGORITHM_MONTE_CARLO:
01475       optimize_algorithm = optimize_MonteCarlo;
01476       break;
01477     case ALGORITHM_SWEEP:
01478       optimize_algorithm = optimize_sweep;
01479       break;
01480     default:
01481       optimize_algorithm = optimize_genetic;
01482       optimize->mutation_ratio = input->mutation_ratio;
01483       optimize->reproduction_ratio = input->
      reproduction_ratio;
01484       optimize->adaptation_ratio = input->adaptation_ratio;
01485     }
01486   optimize->nvariables = input->nvariables;
01487   optimize->nsimulations = input->nsimulations;
01488   optimize->niterations = input->niterations;
01489   optimize->nbest = input->nbest;
01490   optimize->tolerance = input->tolerance;
01491   optimize->nsteps = input->nsteps;
01492   optimize->nestimates = 0;
01493   optimize->threshold = input->threshold;
01494   optimize->stop = 0;
01495   if (input->nsteps)
01496     {
01497       optimize->relaxation = input->relaxation;
01498       switch (input->direction)
01499         {
01500         case DIRECTION_METHOD_COORDINATES:
01501           optimize->nestimates = 2 * optimize->nvariables;
01502           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
01503           break;
01504         default:
01505           optimize->nestimates = input->nestimates;
01506           optimize_estimate_direction =
      optimize_estimate_direction_random;
01507         }
01508     }
01509
01510 #if DEBUG_OPTIMIZE
01511   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01512 #endif
01513   optimize->simulation_best
01514     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01515   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01516
01517   // Reading the experimental data
01518 #if DEBUG_OPTIMIZE
01519   buffer = g_get_current_dir ();
01520   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
```

```
01521   g_free (buffer);
01522 #endif
01523   optimize->nexperiments = input->nexperiments;
01524   optimize->ninputs = input->experiment->ninputs;
01525   optimize->experiment
01526     = (char **) alloca (input->nexperiments * sizeof (char *));
01527   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01528   for (i = 0; i < input->experiment->ninputs; ++i)
01529     optimize->file[i] = (GMappedFile **)
01530       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01531   for (i = 0; i < input->nexperiments; ++i)
01532     {
01533 #if DEBUG_OPTIMIZE
01534       fprintf (stderr, "optimize_open: i=%u\n", i);
01535 #endif
01536       optimize->experiment[i] = input->experiment[i].
      name;
01537       optimize->weight[i] = input->experiment[i].weight;
01538 #if DEBUG_OPTIMIZE
01539       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01540                optimize->experiment[i], optimize->weight[i]);
01541 #endif
01542       for (j = 0; j < input->experiment->ninputs; ++j)
01543         {
01544 #if DEBUG_OPTIMIZE
01545           fprintf (stderr, "optimize_open: template%u\n", j + 1);
01546 #endif
01547           optimize->file[j][i]
01548             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01549         }
01550     }
01551
01552   // Reading the variables data
01553 #if DEBUG_OPTIMIZE
01554   fprintf (stderr, "optimize_open: reading variables\n");
01555 #endif
01556   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01557   j = input->nvariables * sizeof (double);
01558   optimize->rangemin = (double *) alloca (j);
01559   optimize->rangeminabs = (double *) alloca (j);
01560   optimize->rangemax = (double *) alloca (j);
01561   optimize->rangemaxabs = (double *) alloca (j);
01562   optimize->step = (double *) alloca (j);
01563   j = input->nvariables * sizeof (unsigned int);
01564   optimize->precision = (unsigned int *) alloca (j);
01565   optimize->nsweeps = (unsigned int *) alloca (j);
01566   optimize->nbits = (unsigned int *) alloca (j);
01567   for (i = 0; i < input->nvariables; ++i)
01568     {
01569       optimize->label[i] = input->variable[i].name;
01570       optimize->rangemin[i] = input->variable[i].rangemin;
01571       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01572       optimize->rangemax[i] = input->variable[i].rangemax;
01573       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01574       optimize->precision[i] = input->variable[i].
      precision;
01575       optimize->step[i] = input->variable[i].step;
01576       optimize->nsweeps[i] = input->variable[i].nsweeps;
01577       optimize->nbits[i] = input->variable[i].nbits;
01578     }
01579   if (input->algorithm == ALGORITHM_SWEEP)
01580     {
01581       optimize->nsimulations = 1;
01582       for (i = 0; i < input->nvariables; ++i)
01583         {
01584           if (input->algorithm == ALGORITHM_SWEEP)
01585             {
01586               optimize->nsimulations *= optimize->nsweeps[i];
01587 #if DEBUG_OPTIMIZE
01588               fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01589                        optimize->nsweeps[i], optimize->nsimulations);
01590 #endif
01591             }
01592         }
01593     }
01594   if (optimize->nsteps)
01595     optimize->direction
01596       = (double *) alloca (optimize->nvariables * sizeof (double));
01597
01598   // Setting error norm
01599   switch (input->norm)
01600     {
01601     case ERROR_NORM_EUCLIDIAN:
01602       optimize_norm = optimize_norm_euclidian;
01603       break;
```

```
01604     case ERROR_NORM_MAXIMUM:
01605       optimize_norm = optimize_norm_maximum;
01606       break;
01607     case ERROR_NORM_P:
01608       optimize_norm = optimize_norm_p;
01609       optimize->p = input->p;
01610       break;
01611     default:
01612       optimize_norm = optimize_norm_taxicab;
01613     }
01614
01615   // Allocating values
01616 #if DEBUG_OPTIMIZE
01617   fprintf (stderr, "optimize_open: allocating variables\n");
01618   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01619            optimize->nvariables, optimize->algorithm);
01620 #endif
01621   optimize->genetic_variable = NULL;
01622   if (optimize->algorithm == ALGORITHM_GENETIC)
01623     {
01624       optimize->genetic_variable = (GeneticVariable *)
01625         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01626       for (i = 0; i < optimize->nvariables; ++i)
01627         {
01628 #if DEBUG_OPTIMIZE
01629           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01630                    i, optimize->rangemin[i], optimize->rangemax[i],
01631                    optimize->nbits[i]);
01632 #endif
01633           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
01634           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
01635           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01636         }
01637     }
01638 #if DEBUG_OPTIMIZE
01639   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01640            optimize->nvariables, optimize->nsimulations);
01641 #endif
01642   optimize->value = (double *)
01643     g_malloc ((optimize->nsimulations
01644                + optimize->nestimates * optimize->nsteps)
01645               * optimize->nvariables * sizeof (double));
01646
01647   // Calculating simulations to perform for each task
01648 #if HAVE_MPI
01649 #if DEBUG_OPTIMIZE
01650   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01651            optimize->mpi_rank, ntasks);
01652 #endif
01653   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
01654   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
01655   if (optimize->nsteps)
01656     {
01657       optimize->nstart_direction
01658         = optimize->mpi_rank * optimize->nestimates / ntasks;
01659       optimize->nend_direction
01660         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
01661     }
01662 #else
01663   optimize->nstart = 0;
01664   optimize->nend = optimize->nsimulations;
01665   if (optimize->nsteps)
01666     {
01667       optimize->nstart_direction = 0;
01668       optimize->nend_direction = optimize->nestimates;
01669     }
01670 #endif
01671 #if DEBUG_OPTIMIZE
01672   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01673            optimize->nend);
01674 #endif
01675
01676   // Calculating simulations to perform for each thread
01677   optimize->thread
01678     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01679   for (i = 0; i <= nthreads; ++i)
01680     {
01681       optimize->thread[i] = optimize->nstart
01682         + i * (optimize->nend - optimize->nstart) / nthreads;
01683 #if DEBUG_OPTIMIZE
01684       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01685                optimize->thread[i]);
```

```
01686 #endif
01687     }
01688   if (optimize->nsteps)
01689     optimize->thread_direction = (unsigned int *)
01690       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01691
01692   // Opening result files
01693   optimize->file_result = g_fopen (optimize->result, "w");
01694   optimize->file_variables = g_fopen (optimize->variables, "w");
01695
01696   // Performing the algorithm
01697   switch (optimize->algorithm)
01698     {
01699       // Genetic algorithm
01700     case ALGORITHM_GENETIC:
01701       optimize_genetic ();
01702       break;
01703
01704       // Iterative algorithm
01705     default:
01706       optimize_iterate ();
01707     }
01708
01709   // Getting calculation time
01710   t = g_date_time_new_now (tz);
01711   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01712   g_date_time_unref (t);
01713   g_date_time_unref (t0);
01714   g_time_zone_unref (tz);
01715   printf ("%s = %.6lg s\n", _("Calculation time"), optimize->calculation_time);
01716   fprintf (optimize->file_result, "%s = %.6lg s\n",
01717            _("Calculation time"), optimize->calculation_time);
01718
01719   // Closing result files
01720   fclose (optimize->file_variables);
01721   fclose (optimize->file_result);
01722
01723 #if DEBUG_OPTIMIZE
01724   fprintf (stderr, "optimize_open: end\n");
01725 #endif
01726 }
```

## 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**



Definition in file optimize.h.

## 4.19.2 Function Documentation

### 4.19.2.1 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 469 of file optimize.c.

```
00470 {
```

```
00471   unsigned int i, j;
00472   double e;
00473 #if DEBUG_OPTIMIZE
00474   fprintf (stderr, "optimize_best: start\n");
00475   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00476            optimize->nsaveds, optimize->nbest);
00477 #endif
00478   if (optimize->nsaveds < optimize->nbest
00479       || value < optimize->error_best[optimize->nsaveds - 1])
00480     {
00481       if (optimize->nsaveds < optimize->nbest)
00482         ++optimize->nsaveds;
00483       optimize->error_best[optimize->nsaveds - 1] = value;
00484       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00485       for (i = optimize->nsaveds; --i;)
00486         {
00487           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00488             {
00489               j = optimize->simulation_best[i];
00490               e = optimize->error_best[i];
00491               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00492               optimize->error_best[i] = optimize->
      error_best[i - 1];
00493               optimize->simulation_best[i - 1] = j;
00494               optimize->error_best[i - 1] = e;
00495             }
00496           else
00497             break;
00498         }
00499     }
00500 #if DEBUG_OPTIMIZE
00501   fprintf (stderr, "optimize_best: end\n");
00502 #endif
00503 }
```

#### 4.19.2.2 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 794 of file optimize.c.

```
00795 {
00796 #if DEBUG_OPTIMIZE
00797   fprintf (stderr, "optimize_best_direction: start\n");
00798   fprintf (stderr,
00799            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00800            simulation, value, optimize->error_best[0]);
00801 #endif
00802   if (value < optimize->error_best[0])
00803     {
00804       optimize->error_best[0] = value;
00805       optimize->simulation_best[0] = simulation;
00806 #if DEBUG_OPTIMIZE
00807       fprintf (stderr,
00808                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00809                simulation, value);
00810 #endif
00811     }
00812 #if DEBUG_OPTIMIZE
00813   fprintf (stderr, "optimize_best_direction: end\n");
00814 #endif
00815 }
```

#### 4.19.2.3 void optimize_direction_sequential ( unsigned int *simulation* )

Function to estimate the direction search sequentially.

**Parameters**

| | |
|---|---|
| *simulation* | Simulation number. |

Definition at line 824 of file optimize.c.

```
00825 {
00826   unsigned int i, j;
00827   double e;
00828 #if DEBUG_OPTIMIZE
00829   fprintf (stderr, "optimize_direction_sequential: start\n");
00830   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00831            "nend_direction=%u\n",
00832            optimize->nstart_direction, optimize->
      nend_direction);
00833 #endif
00834   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00835     {
00836       j = simulation + i;
00837       e = optimize_norm (j);
00838       optimize_best_direction (j, e);
00839       optimize_save_variables (j, e);
00840       if (e < optimize->threshold)
00841         {
00842           optimize->stop = 1;
00843           break;
00844         }
00845 #if DEBUG_OPTIMIZE
00846       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00847 #endif
00848     }
00849 #if DEBUG_OPTIMIZE
00850   fprintf (stderr, "optimize_direction_sequential: end\n");
00851 #endif
00852 }
```

Here is the call graph for this function:

#### 4.19.2.4 void* optimize_direction_thread ( ParallelData * *data* )

Function to estimate the direction search on a thread.

**Parameters**

| | |
|---|---|
| *data* | Function data. |

**Returns**

NULL

Definition at line 862 of file optimize.c.

```
00863 {
00864   unsigned int i, thread;
00865   double e;
00866 #if DEBUG_OPTIMIZE
00867   fprintf (stderr, "optimize_direction_thread: start\n");
00868 #endif
00869   thread = data->thread;
00870 #if DEBUG_OPTIMIZE
00871   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00872            thread,
00873            optimize->thread_direction[thread],
00874            optimize->thread_direction[thread + 1]);
00875 #endif
```

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

Here is the call graph for this function:

#### 4.19.2.5 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 936 of file optimize.c.

```
00938 {
00939   double x;
00940 #if DEBUG_OPTIMIZE
00941   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00942 #endif
00943   x = optimize->direction[variable];
00944   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00945     {
00946       if (estimate & 1)
00947         x += optimize->step[variable];
00948       else
00949         x -= optimize->step[variable];
00950     }
00951 #if DEBUG_OPTIMIZE
00952   fprintf (stderr,
00953            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00954            variable, x);
00955   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00956 #endif
00957   return x;
00958 }
```

#### 4.19.2.6 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 909 of file optimize.c.

```
00911 {
00912   double x;
00913 #if DEBUG_OPTIMIZE
00914   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00915 #endif
00916   x = optimize->direction[variable]
00917     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00918 #if DEBUG_OPTIMIZE
00919   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00920            variable, x);
00921   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00922 #endif
00923   return x;
00924 }
```

#### 4.19.2.7 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 1103 of file optimize.c.

```
01104 {
01105   unsigned int j;
01106   double objective;
01107   char buffer[64];
01108 #if DEBUG_OPTIMIZE
01109   fprintf (stderr, "optimize_genetic_objective: start\n");
01110 #endif
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       optimize->value[entity->id * optimize->nvariables + j]
01114         = genetic_get_variable (entity, optimize->genetic_variable + j);
01115     }
01116   objective = optimize_norm (entity->id);
01117   g_mutex_lock (mutex);
01118   for (j = 0; j < optimize->nvariables; ++j)
01119     {
01120       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01121       fprintf (optimize->file_variables, buffer,
01122                genetic_get_variable (entity, optimize->genetic_variable + j));
01123     }
01124   fprintf (optimize->file_variables, "%.14le\n", objective);
01125   g_mutex_unlock (mutex);
01126 #if DEBUG_OPTIMIZE
01127   fprintf (stderr, "optimize_genetic_objective: end\n");
01128 #endif
01129   return objective;
01130 }
```

Here is the call graph for this function:

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

#### 4.19.2.9 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 592 of file optimize.c.

```
00594 {
00595   unsigned int i, j, k, s[optimize->nbest];
00596   double e[optimize->nbest];
00597 #if DEBUG_OPTIMIZE
00598   fprintf (stderr, "optimize_merge: start\n");
00599 #endif
00600   i = j = k = 0;
00601   do
00602     {
00603       if (i == optimize->nsaveds)
00604         {
00605           s[k] = simulation_best[j];
00606           e[k] = error_best[j];
00607           ++j;
00608           ++k;
00609           if (j == nsaveds)
00610             break;
00611         }
00612       else if (j == nsaveds)
00613         {
00614           s[k] = optimize->simulation_best[i];
00615           e[k] = optimize->error_best[i];
00616           ++i;
00617           ++k;
00618           if (i == optimize->nsaveds)
00619             break;
00620         }
00621       else if (optimize->error_best[i] > error_best[j])
00622         {
00623           s[k] = simulation_best[j];
00624           e[k] = error_best[j];
00625           ++j;
00626           ++k;
00627         }
00628       else
00629         {
00630           s[k] = optimize->simulation_best[i];
00631           e[k] = optimize->error_best[i];
00632           ++i;
00633           ++k;
00634         }
00635     }
00636   while (k < optimize->nbest);
00637   optimize->nsaveds = k;
00638   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00639   memcpy (optimize->error_best, e, k * sizeof (double));
00640 #if DEBUG_OPTIMIZE
00641   fprintf (stderr, "optimize_merge: end\n");
00642 #endif
00643 }
```

#### 4.19.2.10 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 301 of file optimize.c.

```
00302 {
00303   double e, ei;
00304   unsigned int i;
00305 #if DEBUG_OPTIMIZE
00306   fprintf (stderr, "optimize_norm_euclidian: start\n");
00307 #endif
00308   e = 0.;
00309   for (i = 0; i < optimize->nexperiments; ++i)
00310     {
00311       ei = optimize_parse (simulation, i);
00312       e += ei * ei;
00313     }
00314   e = sqrt (e);
00315 #if DEBUG_OPTIMIZE
00316   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00317   fprintf (stderr, "optimize_norm_euclidian: end\n");
00318 #endif
00319   return e;
00320 }
```

Here is the call graph for this function:

#### 4.19.2.11 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 330 of file optimize.c.

```
00331 {
00332   double e, ei;
00333   unsigned int i;
00334 #if DEBUG_OPTIMIZE
00335   fprintf (stderr, "optimize_norm_maximum: start\n");
00336 #endif
00337   e = 0.;
00338   for (i = 0; i < optimize->nexperiments; ++i)
00339     {
00340       ei = fabs (optimize_parse (simulation, i));
00341       e = fmax (e, ei);
00342     }
00343 #if DEBUG_OPTIMIZE
00344   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00345   fprintf (stderr, "optimize_norm_maximum: end\n");
00346 #endif
00347   return e;
00348 }
```

Here is the call graph for this function:

#### 4.19.2.12 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 358 of file optimize.c.

```
00359 {
00360   double e, ei;
00361   unsigned int i;
00362 #if DEBUG_OPTIMIZE
00363   fprintf (stderr, "optimize_norm_p: start\n");
00364 #endif
00365   e = 0.;
00366   for (i = 0; i < optimize->nexperiments; ++i)
00367     {
00368       ei = fabs (optimize_parse (simulation, i));
00369       e += pow (ei, optimize->p);
00370     }
00371   e = pow (e, 1. / optimize->p);
00372 #if DEBUG_OPTIMIZE
00373   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00374   fprintf (stderr, "optimize_norm_p: end\n");
00375 #endif
00376   return e;
00377 }
```

Here is the call graph for this function:

#### 4.19.2.13 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 387 of file optimize.c.

```
00388 {
00389   double e;
00390   unsigned int i;
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: start\n");
00393 #endif
00394   e = 0.;
00395   for (i = 0; i < optimize->nexperiments; ++i)
00396     e += fabs (optimize_parse (simulation, i));
00397 #if DEBUG_OPTIMIZE
00398   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00399   fprintf (stderr, "optimize_norm_taxicab: end\n");
00400 #endif
00401   return e;
00402 }
```

Here is the call graph for this function:

#### 4.19.2.14 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 190 of file optimize.c.

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

```
00250 #endif
00251       system (buffer);
00252       file_result = g_fopen (result, "r");
00253       e = atof (fgets (buffer, 512, file_result));
00254       fclose (file_result);
00255     }
00256   else
00257     {
00258 #if DEBUG_OPTIMIZE
00259       fprintf (stderr, "optimize_parse: output=%s\n", output);
00260 #endif
00261       strcpy (result, "");
00262       file_result = g_fopen (output, "r");
00263       e = atof (fgets (buffer, 512, file_result));
00264       fclose (file_result);
00265     }
00266
00267   // Removing files
00268 #if !DEBUG_OPTIMIZE
00269   for (i = 0; i < optimize->ninputs; ++i)
00270     {
00271       if (optimize->file[i][0])
00272         {
00273           snprintf (buffer, 512, RM " %s", &input[i][0]);
00274           system (buffer);
00275         }
00276     }
00277   snprintf (buffer, 512, RM " %s %s", output, result);
00278   system (buffer);
00279 #endif
00280
00281   // Processing pending events
00282   if (show_pending)
00283     show_pending ();
00284
00285 #if DEBUG_OPTIMIZE
00286   fprintf (stderr, "optimize_parse: end\n");
00287 #endif
00288
00289   // Returning the objective function
00290   return e * optimize->weight[experiment];
00291 }
```

Here is the call graph for this function:

#### 4.19.2.15 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 440 of file optimize.c.

```
00441 {
00442   unsigned int i;
00443   char buffer[64];
00444 #if DEBUG_OPTIMIZE
00445   fprintf (stderr, "optimize_save_variables: start\n");
00446 #endif
00447   for (i = 0; i < optimize->nvariables; ++i)
00448     {
00449       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00450       fprintf (optimize->file_variables, buffer,
00451                optimize->value[simulation * optimize->
      nvariables + i]);
00452     }
00453   fprintf (optimize->file_variables, "%.14le\n", error);
00454   fflush (optimize->file_variables);
00455 #if DEBUG_OPTIMIZE
00456   fprintf (stderr, "optimize_save_variables: end\n");
00457 #endif
00458 }
```

#### 4.19.2.16 void optimize_step_direction ( unsigned int *simulation* )

Function to do a step of the direction search method.

**Parameters**

| | |
|---|---|
| *simulation* | Simulation number. |

Definition at line 967 of file optimize.c.

```
00968 {
00969   GThread *thread[nthreads_direction];
00970   ParallelData data[nthreads_direction];
00971   unsigned int i, j, k, b;
00972 #if DEBUG_OPTIMIZE
00973   fprintf (stderr, "optimize_step_direction: start\n");
00974 #endif
00975   for (i = 0; i < optimize->nestimates; ++i)
00976     {
00977       k = (simulation + i) * optimize->nvariables;
00978       b = optimize->simulation_best[0] * optimize->
      nvariables;
00979 #if DEBUG_OPTIMIZE
00980       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00981                simulation + i, optimize->simulation_best[0]);
00982 #endif
00983       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00984         {
00985 #if DEBUG_OPTIMIZE
00986           fprintf (stderr,
00987                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00988                    i, j, optimize->value[b]);
00989 #endif
00990           optimize->value[k]
00991             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00992           optimize->value[k] = fmin (fmax (optimize->value[k],
00993                                            optimize->rangeminabs[j]),
00994                                      optimize->rangemaxabs[j]);
00995 #if DEBUG_OPTIMIZE
00996           fprintf (stderr,
00997                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00998                    i, j, optimize->value[k]);
00999 #endif
01000         }
01001     }
01002   if (nthreads_direction == 1)
01003     optimize_direction_sequential (simulation);
01004   else
01005     {
01006       for (i = 0; i <= nthreads_direction; ++i)
01007         {
01008           optimize->thread_direction[i]
01009             = simulation + optimize->nstart_direction
01010             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01011               / nthreads_direction;
01012 #if DEBUG_OPTIMIZE
01013           fprintf (stderr,
01014                    "optimize_step_direction: i=%u thread_direction=%u\n",
01015                    i, optimize->thread_direction[i]);
01016 #endif
01017         }
01018       for (i = 0; i < nthreads_direction; ++i)
01019         {
01020           data[i].thread = i;
01021           thread[i] = g_thread_new
01022             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01023         }
01024       for (i = 0; i < nthreads_direction; ++i)
01025         g_thread_join (thread[i]);
01026     }
01027 #if DEBUG_OPTIMIZE
01028   fprintf (stderr, "optimize_step_direction: end\n");
01029 #endif
01030 }
```

Here is the call graph for this function:

#### 4.19.2.17 void* optimize_thread ( ParallelData * *data* )

Function to optimize on a thread.

**Parameters**

| | |
|---|---|
| *data* | Function data. |

**Returns**

NULL

Definition at line 546 of file optimize.c.

```
00547 {
00548   unsigned int i, thread;
00549   double e;
00550 #if DEBUG_OPTIMIZE
00551   fprintf (stderr, "optimize_thread: start\n");
00552 #endif
00553   thread = data->thread;
00554 #if DEBUG_OPTIMIZE
00555   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00556            optimize->thread[thread], optimize->thread[thread + 1]);
00557 #endif
00558   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00559     {
00560       e = optimize_norm (i);
00561       g_mutex_lock (mutex);
00562       optimize_best (i, e);
00563       optimize_save_variables (i, e);
00564       if (e < optimize->threshold)
00565         optimize->stop = 1;
00566       g_mutex_unlock (mutex);
00567       if (optimize->stop)
00568         break;
00569 #if DEBUG_OPTIMIZE
00570       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00571 #endif
00572     }
00573 #if DEBUG_OPTIMIZE
00574   fprintf (stderr, "optimize_thread: end\n");
00575 #endif
00576   g_thread_exit (NULL);
00577   return NULL;
00578 }
```

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 2012-2017, 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



Definition in file utils.c.

### 4.21.2 Function Documentation

#### 4.21.2.1 int cores_number ( )

Function to obtain the cores number.

#### Returns

Cores number.

Definition at line 530 of file utils.c.

```
00531 {
00532 #ifdef G_OS_WIN32
00533   SYSTEM_INFO sysinfo;
00534   GetSystemInfo (&sysinfo);
00535   return sysinfo.dwNumberOfProcessors;
00536 #else
00537   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00538 #endif
00539 }
```

#### 4.21.2.2 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 565 of file utils.c.

```
00566 {
00567   unsigned int i;
00568   for (i = 0; i < n; ++i)
00569     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570       break;
00571   return i;
00572 }
```

#### 4.21.2.3 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 420 of file utils.c.

```
00421 {
00422   const char *buffer;
00423   double x = 0.;
00424   buffer = json_object_get_string_member (object, prop);
00425   if (!buffer)
00426     *error_code = 1;
00427   else
00428     {
00429       if (sscanf (buffer, "%lf", &x) != 1)
00430         *error_code = 2;
00431       else
00432         *error_code = 0;
00433     }
00434   return x;
00435 }
```

#### 4.21.2.4 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 453 of file utils.c.

```
00455 {
00456   double x;
00457   if (json_object_get_member (object, prop))
00458     x = json_object_get_float (object, prop, error_code);
00459   else
00460     {
00461       x = default_value;
00462       *error_code = 0;
00463     }
00464   return x;
00465 }
```

Here is the call graph for this function:

#### 4.21.2.5 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 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 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.

```
00395 {
00396   unsigned int i;
00397   if (json_object_get_member (object, prop))
00398     i = json_object_get_uint (object, prop, error_code);
00399   else
00400     {
00401       i = default_value;
00402       *error_code = 0;
00403     }
00404   return i;
00405 }
```

Here is the call graph for this function:

#### 4.21.2.8 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 517 of file utils.c.

```
00518 {
00519   char buffer[64];
00520   snprintf (buffer, 64, "%.14lg", value);
00521   json_object_set_string_member (object, prop, buffer);
00522 }
```

#### 4.21.2.9 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 479 of file utils.c.

```
00480 {
00481   char buffer[64];
00482   snprintf (buffer, 64, "%d", value);
00483   json_object_set_string_member (object, prop, buffer);
00484 }
```

#### 4.21.2.10 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 498 of file utils.c.

```
00499 {
```

```
00500   char buffer[64];
00501   snprintf (buffer, 64, "%u", value);
00502   json_object_set_string_member (object, prop, buffer);
00503 }
```

#### 4.21.2.11 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 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 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 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 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 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 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 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 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 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 2012-2017, 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, int *error_code)
00395 {
00396   unsigned int i;
00397   if (json_object_get_member (object, prop))
00398     i = json_object_get_uint (object, prop, error_code);
00399   else
00400     {
00401       i = default_value;
00402       *error_code = 0;
00403     }
00404   return i;
00405 }
00406
00419 double
00420 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00421 {
00422   const char *buffer;
00423   double x = 0.;
00424   buffer = json_object_get_string_member (object, prop);
00425   if (!buffer)
00426     *error_code = 1;
00427   else
00428     {
00429       if (sscanf (buffer, "%lf", &x) != 1)
00430         *error_code = 2;
00431       else
00432         *error_code = 0;
00433     }
00434   return x;
00435 }
00436
00452 double
00453 json_object_get_float_with_default (JsonObject * object, const char *prop
    ,
00454                                     double default_value, int *error_code)
00455 {
00456   double x;
```

```
00457   if (json_object_get_member (object, prop))
00458     x = json_object_get_float (object, prop, error_code);
00459   else
00460     {
00461       x = default_value;
00462       *error_code = 0;
00463     }
00464   return x;
00465 }
00466
00478 void
00479 json_object_set_int (JsonObject * object, const char *prop, int value)
00480 {
00481   char buffer[64];
00482   snprintf (buffer, 64, "%d", value);
00483   json_object_set_string_member (object, prop, buffer);
00484 }
00485
00497 void
00498 json_object_set_uint (JsonObject * object, const char *prop, unsigned int value)
00499 {
00500   char buffer[64];
00501   snprintf (buffer, 64, "%u", value);
00502   json_object_set_string_member (object, prop, buffer);
00503 }
00504
00516 void
00517 json_object_set_float (JsonObject * object, const char *prop, double value)
00518 {
00519   char buffer[64];
00520   snprintf (buffer, 64, "%.14lg", value);
00521   json_object_set_string_member (object, prop, buffer);
00522 }
00523
00529 int
00530 cores_number ()
00531 {
00532 #ifdef G_OS_WIN32
00533   SYSTEM_INFO sysinfo;
00534   GetSystemInfo (&sysinfo);
00535   return sysinfo.dwNumberOfProcessors;
00536 #else
00537   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00538 #endif
00539 }
00540
00541 #if HAVE_GTK
00542
00547 void
00548 process_pending ()
00549 {
00550   while (gtk_events_pending ())
00551     gtk_main_iteration ();
00552 }
00553
00564 unsigned int
00565 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00566 {
00567   unsigned int i;
00568   for (i = 0; i < n; ++i)
00569     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570       break;
00571   return i;
00572 }
00573
00574 #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 2012-2017, all rights reserved.

Definition in file utils.h.

## 4.23.2 Function Documentation

### 4.23.2.1 int cores_number ( )

Function to obtain the cores number.

#### Returns

Cores number.

Definition at line 530 of file utils.c.

```
00531 {
00532 #ifdef G_OS_WIN32
00533   SYSTEM_INFO sysinfo;
00534   GetSystemInfo (&sysinfo);
00535   return sysinfo.dwNumberOfProcessors;
00536 #else
00537   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00538 #endif
00539 }
```

### 4.23.2.2 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 565 of file utils.c.

```
00566 {
00567   unsigned int i;
00568   for (i = 0; i < n; ++i)
00569     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570       break;
00571   return i;
00572 }
```

#### 4.23.2.3 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 420 of file utils.c.

```
00421 {
00422   const char *buffer;
00423   double x = 0.;
00424   buffer = json_object_get_string_member (object, prop);
00425   if (!buffer)
00426     *error_code = 1;
00427   else
00428     {
00429       if (sscanf (buffer, "%lf", &x) != 1)
00430         *error_code = 2;
00431       else
00432         *error_code = 0;
00433     }
00434   return x;
00435 }
```

#### 4.23.2.4 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 453 of file utils.c.

```
00455 {
00456   double x;
00457   if (json_object_get_member (object, prop))
00458     x = json_object_get_float (object, prop, error_code);
00459   else
00460     {
00461       x = default_value;
00462       *error_code = 0;
00463     }
00464   return x;
00465 }
```

Here is the call graph for this function:

#### 4.23.2.5 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 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 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.

```
00395 {
00396   unsigned int i;
00397   if (json_object_get_member (object, prop))
00398     i = json_object_get_uint (object, prop, error_code);
00399   else
```

```
00400     {
00401       i = default_value;
00402       *error_code = 0;
00403     }
00404   return i;
00405 }
```

Here is the call graph for this function:

#### 4.23.2.8 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 517 of file utils.c.

```
00518 {
00519   char buffer[64];
00520   snprintf (buffer, 64, "%.14lg", value);
00521   json_object_set_string_member (object, prop, buffer);
00522 }
```

#### 4.23.2.9 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 479 of file utils.c.

```
00480 {
00481   char buffer[64];
00482   snprintf (buffer, 64, "%d", value);
00483   json_object_set_string_member (object, prop, buffer);
00484 }
```

#### 4.23.2.10 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 498 of file utils.c.

```
00499 {
00500   char buffer[64];
00501   snprintf (buffer, 64, "%u", value);
00502   json_object_set_string_member (object, prop, buffer);
00503 }
```

#### 4.23.2.11 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 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 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 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 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 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 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 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 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 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 2012-2017, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013   1. Redistributions of source code must retain the above copyright notice,
00014     this list of conditions and the following disclaimer.
00015
00016   2. Redistributions in binary form must reproduce the above copyright notice,
00017     this list of conditions and the following disclaimer in the
00018     documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef UTILS__H
00039 #define UTILS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 extern char *error_message;
00057 extern void (*show_pending) ();
00058
00059 // Public functions
00060 void show_message (char *title, char *msg, int type);
00061 void show_error (char *msg);
00062 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00063 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00064                                 int *error_code);
00065 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00066                                              const xmlChar * prop,
00067                                              unsigned int default_value,
00068                                              int *error_code);
00069 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00070                            int *error_code);
00071 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop
      ,
00072                                         double default_value, int *error_code);
00073 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00074 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00075                         unsigned int value);
00076 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00077 int json_object_get_int (JsonObject * object, const char *prop,
00078                          int *error_code);
00079 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00080                                    int *error_code);
00081 unsigned int json_object_get_uint_with_default (JsonObject * object,
00082                                                 const char *prop,
00083                                                 unsigned int default_value,
00084                                                 int *error_code);
00085 double json_object_get_float (JsonObject * object, const char *prop,
00086                               int *error_code);
```

```
00087 double json_object_get_float_with_default (JsonObject * object,
00088                                             const char *prop,
00089                                             double default_value,
00090                                             int *error_code);
00091 void json_object_set_int (JsonObject * object, const char *prop, int value);
00092 void json_object_set_uint (JsonObject * object, const char *prop,
00093                            unsigned int value);
00094 void json_object_set_float (JsonObject * object, const char *prop,
00095                             double value);
00096 int cores_number ();
00097 #if HAVE_GTK
00098 void process_pending ();
00099 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00100 #endif
00101
00102 #endif
```

## 4.25 variable.c File Reference

Source file to define the variable data.

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

Include dependency graph for variable.c:

### Macros

- #define DEBUG_VARIABLE 0

  *Macro to debug variable functions.*

### Functions

- void variable_new (Variable *variable)

  *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 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, message);
00117   error_message = g_strdup (buffer);
00118 }
```

#### 4.25.2.2 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 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 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 301 of file variable.c.

```
00303 {
00304   JsonObject *object;
00305   const char *label;
00306   int error_code;
00307 #if DEBUG_VARIABLE
00308   fprintf (stderr, "variable_open_json: start\n");
00309 #endif
```

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

```
00394             {
00395               variable_error (variable, _("bad sweeps"));
00396               goto exit_on_error;
00397             }
00398         }
00399       else
00400         {
00401           variable_error (variable, _("no sweeps number"));
00402           goto exit_on_error;
00403         }
00404 #if DEBUG_VARIABLE
00405       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407     }
00408   if (algorithm == ALGORITHM_GENETIC)
00409     {
00410       // Obtaining bits representing each variable
00411       if (json_object_get_member (object, LABEL_NBITS))
00412         {
00413           variable->nbits
00414             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415           if (error_code || !variable->nbits)
00416             {
00417               variable_error (variable, _("invalid bits number"));
00418               goto exit_on_error;
00419             }
00420         }
00421       else
00422         {
00423           variable_error (variable, _("no bits number"));
00424           goto exit_on_error;
00425         }
00426     }
00427   else if (nsteps)
00428     {
00429       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00430       if (error_code || variable->step < 0.)
00431         {
00432           variable_error (variable, _("bad step size"));
00433           goto exit_on_error;
00434         }
00435     }
00436
00437 #if DEBUG_VARIABLE
00438   fprintf (stderr, "variable_open_json: end\n");
00439 #endif
00440   return 1;
00441 exit_on_error:
00442   variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444   fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446   return 0;
00447 }
```

Here is the call graph for this function:

#### 4.25.2.5 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 135 of file variable.c.

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

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

Here is the call graph for this function:

### 4.25.3 Variable Documentation

#### 4.25.3.1 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 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 2012-2017, 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, message);
00117   error_message = g_strdup (buffer);
00118 }
00119
00134 int
00135 variable_open_xml (Variable * variable, xmlNode * node,
00136                    unsigned int algorithm, unsigned int nsteps)
00137 {
00138   int error_code;
00139
00140 #if DEBUG_VARIABLE
00141   fprintf (stderr, "variable_open_xml: start\n");
00142 #endif
00143
00144   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145   if (!variable->name)
00146     {
00147       variable_error (variable, _("no name"));
00148       goto exit_on_error;
00149     }
00150   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00151     {
00152       variable->rangemin
00153         = xml_node_get_float (node, (const xmlChar *)
      LABEL_MINIMUM,
00154                               &error_code);
00155       if (error_code)
00156         {
00157           variable_error (variable, _("bad minimum"));
00158           goto exit_on_error;
00159         }
00160       variable->rangeminabs = xml_node_get_float_with_default
00161         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00162          &error_code);
00163       if (error_code)
00164         {
00165           variable_error (variable, _("bad absolute minimum"));
00166           goto exit_on_error;
00167         }
00168       if (variable->rangemin < variable->rangeminabs)
00169         {
00170           variable_error (variable, _("minimum range not allowed"));
00171           goto exit_on_error;
00172         }
00173     }
```

```
00174   else
00175     {
00176       variable_error (variable, _("no minimum range"));
00177       goto exit_on_error;
00178     }
00179   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00180     {
00181       variable->rangemax
00182         = xml_node_get_float (node, (const xmlChar *)
    LABEL_MAXIMUM,
00183                               &error_code);
00184       if (error_code)
00185         {
00186           variable_error (variable, _("bad maximum"));
00187           goto exit_on_error;
00188         }
00189       variable->rangemaxabs = xml_node_get_float_with_default
00190         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00191          &error_code);
00192       if (error_code)
00193         {
00194           variable_error (variable, _("bad absolute maximum"));
00195           goto exit_on_error;
00196         }
00197       if (variable->rangemax > variable->rangemaxabs)
00198         {
00199           variable_error (variable, _("maximum range not allowed"));
00200           goto exit_on_error;
00201         }
00202       if (variable->rangemax < variable->rangemin)
00203         {
00204           variable_error (variable, _("bad range"));
00205           goto exit_on_error;
00206         }
00207     }
00208   else
00209     {
00210       variable_error (variable, _("no maximum range"));
00211       goto exit_on_error;
00212     }
00213   variable->precision
00214     = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_PRECISION,
00215                                       DEFAULT_PRECISION, &error_code);
00216   if (error_code || variable->precision >= NPRECISIONS)
00217     {
00218       variable_error (variable, _("bad precision"));
00219       goto exit_on_error;
00220     }
00221   if (algorithm == ALGORITHM_SWEEP)
00222     {
00223       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00224         {
00225           variable->nsweeps
00226             = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NSWEEPS,
00227                                  &error_code);
00228           if (error_code || !variable->nsweeps)
00229             {
00230               variable_error (variable, _("bad sweeps"));
00231               goto exit_on_error;
00232             }
00233         }
00234       else
00235         {
00236           variable_error (variable, _("no sweeps number"));
00237           goto exit_on_error;
00238         }
00239 #if DEBUG_VARIABLE
00240       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00241 #endif
00242     }
00243   if (algorithm == ALGORITHM_GENETIC)
00244     {
00245       // Obtaining bits representing each variable
00246       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00247         {
00248           variable->nbits
00249             = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NBITS,
00250                                  &error_code);
00251           if (error_code || !variable->nbits)
00252             {
00253               variable_error (variable, _("invalid bits number"));
00254               goto exit_on_error;
00255             }
00256         }
```

```
00257       else
00258         {
00259           variable_error (variable, _("no bits number"));
00260           goto exit_on_error;
00261         }
00262     }
00263   else if (nsteps)
00264     {
00265       variable->step
00266         = xml_node_get_float (node, (const xmlChar *)
      LABEL_STEP, &error_code);
00267       if (error_code || variable->step < 0.)
00268         {
00269           variable_error (variable, _("bad step size"));
00270           goto exit_on_error;
00271         }
00272     }
00273
00274 #if DEBUG_VARIABLE
00275   fprintf (stderr, "variable_open_xml: end\n");
00276 #endif
00277   return 1;
00278 exit_on_error:
00279   variable_free (variable, INPUT_TYPE_XML);
00280 #if DEBUG_VARIABLE
00281   fprintf (stderr, "variable_open_xml: end\n");
00282 #endif
00283   return 0;
00284 }
00285
00300 int
00301 variable_open_json (Variable * variable, JsonNode * node,
00302                     unsigned int algorithm, unsigned int nsteps)
00303 {
00304   JsonObject *object;
00305   const char *label;
00306   int error_code;
00307 #if DEBUG_VARIABLE
00308   fprintf (stderr, "variable_open_json: start\n");
00309 #endif
00310   object = json_node_get_object (node);
00311   label = json_object_get_string_member (object, LABEL_NAME);
00312   if (!label)
00313     {
00314       variable_error (variable, _("no name"));
00315       goto exit_on_error;
00316     }
00317   variable->name = g_strdup (label);
00318   if (json_object_get_member (object, LABEL_MINIMUM))
00319     {
00320       variable->rangemin
00321         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00322       if (error_code)
00323         {
00324           variable_error (variable, _("bad minimum"));
00325           goto exit_on_error;
00326         }
00327       variable->rangeminabs
00328         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00329                                               -G_MAXDOUBLE, &error_code);
00330       if (error_code)
00331         {
00332           variable_error (variable, _("bad absolute minimum"));
00333           goto exit_on_error;
00334         }
00335       if (variable->rangemin < variable->rangeminabs)
00336         {
00337           variable_error (variable, _("minimum range not allowed"));
00338           goto exit_on_error;
00339         }
00340     }
00341   else
00342     {
00343       variable_error (variable, _("no minimum range"));
00344       goto exit_on_error;
00345     }
00346   if (json_object_get_member (object, LABEL_MAXIMUM))
00347     {
00348       variable->rangemax
00349         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00350       if (error_code)
00351         {
00352           variable_error (variable, _("bad maximum"));
00353           goto exit_on_error;
00354         }
00355       variable->rangemaxabs
```

```
00356         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00357                                               G_MAXDOUBLE, &error_code);
00358       if (error_code)
00359         {
00360           variable_error (variable, _("bad absolute maximum"));
00361           goto exit_on_error;
00362         }
00363       if (variable->rangemax > variable->rangemaxabs)
00364         {
00365           variable_error (variable, _("maximum range not allowed"));
00366           goto exit_on_error;
00367         }
00368       if (variable->rangemax < variable->rangemin)
00369         {
00370           variable_error (variable, _("bad range"));
00371           goto exit_on_error;
00372         }
00373     }
00374   else
00375     {
00376       variable_error (variable, _("no maximum range"));
00377       goto exit_on_error;
00378     }
00379   variable->precision
00380     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00381                                          DEFAULT_PRECISION, &error_code);
00382   if (error_code || variable->precision >= NPRECISIONS)
00383     {
00384       variable_error (variable, _("bad precision"));
00385       goto exit_on_error;
00386     }
00387   if (algorithm == ALGORITHM_SWEEP)
00388     {
00389       if (json_object_get_member (object, LABEL_NSWEEPS))
00390         {
00391           variable->nsweeps
00392             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00393           if (error_code || !variable->nsweeps)
00394             {
00395               variable_error (variable, _("bad sweeps"));
00396               goto exit_on_error;
00397             }
00398         }
00399       else
00400         {
00401           variable_error (variable, _("no sweeps number"));
00402           goto exit_on_error;
00403         }
00404 #if DEBUG_VARIABLE
00405       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407     }
00408   if (algorithm == ALGORITHM_GENETIC)
00409     {
00410       // Obtaining bits representing each variable
00411       if (json_object_get_member (object, LABEL_NBITS))
00412         {
00413           variable->nbits
00414             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415           if (error_code || !variable->nbits)
00416             {
00417               variable_error (variable, _("invalid bits number"));
00418               goto exit_on_error;
00419             }
00420         }
00421       else
00422         {
00423           variable_error (variable, _("no bits number"));
00424           goto exit_on_error;
00425         }
00426     }
00427   else if (nsteps)
00428     {
00429       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00430       if (error_code || variable->step < 0.)
00431         {
00432           variable_error (variable, _("bad step size"));
00433           goto exit_on_error;
00434         }
00435     }
00436
00437 #if DEBUG_VARIABLE
00438   fprintf (stderr, "variable_open_json: end\n");
00439 #endif
```

```
00440   return 1;
00441 exit_on_error:
00442   variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444   fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446   return 0;
00447 }
```

## 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 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 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, message);
00117   error_message = g_strdup (buffer);
00118 }
```

#### 4.27.3.2 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 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 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 301 of file variable.c.

```
00303 {
00304   JsonObject *object;
00305   const char *label;
00306   int error_code;
00307 #if DEBUG_VARIABLE
00308   fprintf (stderr, "variable_open_json: start\n");
00309 #endif
00310   object = json_node_get_object (node);
00311   label = json_object_get_string_member (object, LABEL_NAME);
00312   if (!label)
00313     {
00314       variable_error (variable, _("no name"));
00315       goto exit_on_error;
00316     }
00317   variable->name = g_strdup (label);
00318   if (json_object_get_member (object, LABEL_MINIMUM))
00319     {
00320       variable->rangemin
00321         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00322       if (error_code)
00323         {
00324           variable_error (variable, _("bad minimum"));
00325           goto exit_on_error;
00326         }
00327       variable->rangeminabs
00328         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MINIMUM,
00329                                               -G_MAXDOUBLE, &error_code);
00330       if (error_code)
00331         {
00332           variable_error (variable, _("bad absolute minimum"));
00333           goto exit_on_error;
00334         }
00335       if (variable->rangemin < variable->rangeminabs)
00336         {
00337           variable_error (variable, _("minimum range not allowed"));
00338           goto exit_on_error;
00339         }
00340     }
00341   else
00342     {
00343       variable_error (variable, _("no minimum range"));
00344       goto exit_on_error;
00345     }
00346   if (json_object_get_member (object, LABEL_MAXIMUM))
00347     {
00348       variable->rangemax
00349         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00350       if (error_code)
00351         {
00352           variable_error (variable, _("bad maximum"));
00353           goto exit_on_error;
00354         }
00355       variable->rangemaxabs
00356         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MAXIMUM,
00357                                               G_MAXDOUBLE, &error_code);
00358       if (error_code)
00359         {
00360           variable_error (variable, _("bad absolute maximum"));
00361           goto exit_on_error;
00362         }
00363       if (variable->rangemax > variable->rangemaxabs)
00364         {
```

```
00365           variable_error (variable, _("maximum range not allowed"));
00366           goto exit_on_error;
00367         }
00368       if (variable->rangemax < variable->rangemin)
00369         {
00370           variable_error (variable, _("bad range"));
00371           goto exit_on_error;
00372         }
00373     }
00374   else
00375     {
00376       variable_error (variable, _("no maximum range"));
00377       goto exit_on_error;
00378     }
00379   variable->precision
00380     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00381                                          DEFAULT_PRECISION, &error_code);
00382   if (error_code || variable->precision >= NPRECISIONS)
00383     {
00384       variable_error (variable, _("bad precision"));
00385       goto exit_on_error;
00386     }
00387   if (algorithm == ALGORITHM_SWEEP)
00388     {
00389       if (json_object_get_member (object, LABEL_NSWEEPS))
00390         {
00391           variable->nsweeps
00392             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00393           if (error_code || !variable->nsweeps)
00394             {
00395               variable_error (variable, _("bad sweeps"));
00396               goto exit_on_error;
00397             }
00398         }
00399       else
00400         {
00401           variable_error (variable, _("no sweeps number"));
00402           goto exit_on_error;
00403         }
00404 #if DEBUG_VARIABLE
00405       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00406 #endif
00407     }
00408   if (algorithm == ALGORITHM_GENETIC)
00409     {
00410       // Obtaining bits representing each variable
00411       if (json_object_get_member (object, LABEL_NBITS))
00412         {
00413           variable->nbits
00414             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00415           if (error_code || !variable->nbits)
00416             {
00417               variable_error (variable, _("invalid bits number"));
00418               goto exit_on_error;
00419             }
00420         }
00421       else
00422         {
00423           variable_error (variable, _("no bits number"));
00424           goto exit_on_error;
00425         }
00426     }
00427   else if (nsteps)
00428     {
00429       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00430       if (error_code || variable->step < 0.)
00431         {
00432           variable_error (variable, _("bad step size"));
00433           goto exit_on_error;
00434         }
00435     }
00436
00437 #if DEBUG_VARIABLE
00438   fprintf (stderr, "variable_open_json: end\n");
00439 #endif
00440   return 1;
00441 exit_on_error:
00442   variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444   fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446   return 0;
00447 }
```

Here is the call graph for this function:

#### 4.27.3.5 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 135 of file variable.c.

```
00137 {
00138   int error_code;
00139
00140 #if DEBUG_VARIABLE
00141   fprintf (stderr, "variable_open_xml: start\n");
00142 #endif
00143
00144   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00145   if (!variable->name)
00146     {
00147       variable_error (variable, _("no name"));
00148       goto exit_on_error;
00149     }
00150   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00151     {
00152       variable->rangemin
00153         = xml_node_get_float (node, (const xmlChar *)
    LABEL_MINIMUM,
00154                               &error_code);
00155       if (error_code)
00156         {
00157           variable_error (variable, _("bad minimum"));
00158           goto exit_on_error;
00159         }
00160       variable->rangeminabs = xml_node_get_float_with_default
00161         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00162          &error_code);
00163       if (error_code)
00164         {
00165           variable_error (variable, _("bad absolute minimum"));
00166           goto exit_on_error;
00167         }
00168       if (variable->rangemin < variable->rangeminabs)
00169         {
00170           variable_error (variable, _("minimum range not allowed"));
00171           goto exit_on_error;
00172         }
00173     }
00174   else
00175     {
00176       variable_error (variable, _("no minimum range"));
00177       goto exit_on_error;
00178     }
00179   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00180     {
00181       variable->rangemax
00182         = xml_node_get_float (node, (const xmlChar *)
    LABEL_MAXIMUM,
00183                               &error_code);
00184       if (error_code)
00185         {
00186           variable_error (variable, _("bad maximum"));
```

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

```
00270           goto exit_on_error;
00271         }
00272     }
00273
00274 #if DEBUG_VARIABLE
00275   fprintf (stderr, "variable_open_xml: end\n");
00276 #endif
00277   return 1;
00278 exit_on_error:
00279   variable_free (variable, INPUT_TYPE_XML);
00280 #if DEBUG_VARIABLE
00281   fprintf (stderr, "variable_open_xml: end\n");
00282 #endif
00283   return 0;
00284 }
```

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 2012-2017, 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