MPCOTool

3.0.0

Generated by Doxygen 1.8.9.1

Sat Feb 27 2016 08:08:30

Contents

Chapter 1

MPCOTool

The Multi-Purposes Calibration and Optimization Tool. A software to perform calibrations or optimizations of empirical parameters.

VERSIONS

- 3.0.0: Stable and recommended version.
- 3.1.0: Developing version to do new features.

AUTHORS

- Javier Burguete Tolosa (jburguete@eead.csic.es)
- Borja Latorre Garcés (borja.latorre@csic.es)

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

- gcc or clang (to compile the source code)
- make (to build the executable file)
- autoconf (to generate the Makefile in different operative systems)
- automake (to check the operative system)
- pkg-config (to find the libraries to compile)
- gsl (to generate random numbers)
- libxml (to deal with XML files)
- glib (extended utilities of C to work with data, lists, mapped files, regular expressions, using multicores in shared memory machines, ...)
- json-glib (to deal with JSON files)
- genetic (genetic algorithm)

OPTIONAL TOOLS AND LIBRARIES

- `gettext` (to work with different locales)
- `gtk+3` (to create the interactive GUI tool)
- `openmpi` or `mpich` (to run in parallelized tasks on multiple computers)
- `doxygen` (standard comments format to generate documentation)
- `latex` (to build the PDF manuals)

FILES

The source code has to have the following files:

- 3.0.0/configure.ac: configure generator.
- 3.0.0/Makefile.in: Makefile generator.
- 3.0.0/config.h.in: config header generator.
- 3.0.0/mpcotool.c: main source code.
- 3.0.0/mpcotool.h: main header code.
- 3.0.0/interface.h: interface header code.
- 3.0.0/build: script to build all.
- 3.0.0/logo.png: logo figure.
- 3.0.0/Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- README.md: this file.
- license.md: license file.
- tests/testX/∗: several tests to check the program working.
- locales/∗/LC_MESSAGES/mpcotool.po: translation files.
- manuals/∗.eps: manual figures in EPS format.
- manuals/∗.png: manual figures in PNG format.
- manuals/∗.tex: documentation source files.
- applications/∗/∗: several practical application cases.
- check_errors/∗.xml: several mistaken files to check error handling.

BUILDING INSTRUCTIONS

This software has been built and tested in the following operative systems. Probably, it can be built in other systems, distributions, or versions but it has not been tested.

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.2

Dyson Illumos

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 12, 14, and 15

1. Download the latest genetic doing on a terminal:

   $ git clone https://github.com/jburguete/genetic.git

2. Download this repository:

   $ git clone https://github.com/jburguete/mpcotool.git

3. Link the latest genetic version to genetic:

   $ cd mpcotool/3.0.0
   $ ln -s ../../genetic/2.0.1 genetic

4. Build doing on a terminal:

   $ ./build

OpenBSD 5.8

1. Select adequate versions:

   $ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15

2. Then, in a terminal, follow steps 1 to 4 of the previous Debian 8 section.

Microsoft Windows 7 (with MSYS2)

Microsoft Windows 8.1 (with MSYS2)

1. Install MSYS2 and the required libraries and utilities. You can follow detailed instructions in install-unix
2. Then, in a MSYS2 terminal, follow steps 1 to 4 of the previous Debian 8 section.
3. Optional Windows binary package can be built doing in the terminal:

   $ make windist

Fedora Linux 23

1. In order to use OpenMPI compilation do in a terminal (in 64 bits version):

   $ export PATH=$PATH:/usr/lib64/openmpi/bin

2. Then, follow steps 1 to 4 of the previous Debian 8 section.

FreeBSD 10.2

1. In order to build in FreeBSD, due to a wrong error in default gcc version, do in a terminal:

   $ export CC=gcc5 (or CC=clang)

2. Then, follow steps 1 to 4 of the previous Debian 8 section.

Building no-GUI version on servers

On servers or clusters, where no-GUI with MPI parallelization is desirable, replace the 4th step of the previous Debian 8 section by:

$ ./build_without_gui

MAKING MANUALS INSTRUCTIONS

On UNIX type systems you need `texlive` installed. On Windows systems you need `MiKTeX`. In order to compile the manuals you can type on a terminal:

```
$ make manuals
```

MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory mpcotool/3.0.0):

```
$ cd ../tests/test2
$ ln -s ../../../genetic/2.0.1 genetic
$ cd ../test3
$ ln -s ../../../genetic/2.0.1 genetic
$ cd ../test4
$ ln -s ../../../genetic/2.0.1 genetic
```

2. Build all tests doing in the same terminal:

```
$ cd ../../3.0.0
$ make tests
```

USER INSTRUCTIONS

Optional arguments are typed in square brackets.

- Command line in sequential mode (where X is the number of threads to execute and S is a seed for the pseudo-random numbers generator):

```
$ ./mpcotoolbin [-nthreads X] [-seed S] input_file.xml [result_file] [variables_file]
```

- Command line in parallelized mode (where X is the number of threads to open for every node and S is a seed for the pseudo-random numbers generator):

```
$ mpirun [MPI options] ./mpcotoolbin [-nthreads X] [-seed S] input_file.xml [result_file] [variables↩
_file]
```

- The syntax of the simulator has to be:

```
$ ./simulator_name input_file_1 [input_file_2] [input_file_3] [input_file_4] output_file
```

- The syntax of the program to evaluate the objetive function has to be (where the first data in the results file has to be the objective function value):

```
$ ./evaluator_name simulated_file data_file results_file
```

- On UNIX type systems the GUI application can be open doing on a terminal:

```
$ ./mpcotool
```

INPUT FILE FORMAT

The format of the main input file is as:

“‘xml <?xml version="1.0"?> <optimize simulator="simulator_name" evaluator="evaluator_name" algorithm="algorithm↩
_type" nsimulations="simulations_number" niterations="iterations_number" tolerance="tolerance_value" nbest="best↩
_number" npopulation="population_number" ngenerations="generations_number" mutation="mutation_ratio"
reproduction="reproduction_ratio" adaptation="adaptation_ratio" direction="direction_search_type" nsteps="steps↩
_number" relaxation="relaxation_parameter" nestimates="estimates_number" threshold="threshold_parameter"
norm="norm_type" p="p_parameter" seed="random_seed" result_file="result_file" variables_file="variables_file">
<experiment name="data_file_1" template1="template_1_1" template2="template_1_2" ... weight="weight_1"/> ...
<experiment name="data_file_N" template1="template_N_1" template2="template_N_2" ... weight="weight↩
_N"/> <variable name="variable_1" minimum="min_value" maximum="max_value" precision="precision_↩
digits" sweeps="sweeps_number" nbits="bits_number" step="step_size"> ... <variable name="variable↩
_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps="sweeps_number"
nbits="bits_number" step="step_size"> </optimize> “‘

with:

- **simulator**: simulator executable file name.
- **evaluator**: optional. When needed is the evaluator executable file name.
- **seed**: optional. Seed of the pseudo-random numbers generator (default value is 7007).
- **result_file**: optional. It is the name of the optime result file (default name is "result").
- **variables_file**: optional. It is the name of all simulated variables file (default name is "variables").
- **precision**: optional, defined for each variable. Number of precision digits to evaluate the variable. 0 apply for integer numbers (default value is 14).
- **weight**: optional, defined for each experiment. Multiplies the objective value obtained for each experiment in the final objective function value (default value is 1).
- **threshold**: optional, to stop the simulations if objective function value less than the threshold is obtained (default value is 0).
- **algorithm**: optimization algorithm type.
- **norm**: error norm type.

Implemented algorithms are:

- **sweep**: Sweep brute force algorithm. It requires for each variable:
  - *sweeps*: number of sweeps to generate for each variable in every experiment.
    The total number of simulations to run is:

    (number of experiments) x (variable 1 number of sweeps) x ... x (variable n number of sweeps) x (number of iterations)
- **Monte-Carlo**: Monte-Carlo brute force algorithm. It requires on calibrate:
  - *nsimulations*: number of simulations to run in every experiment.
    The total number of simulations to run is:

    (number of experiments) x (number of simulations) x (number of iterations)
- Both brute force algorithms can be iterated to improve convergence by using the following parameters:
  - *nbest*: number of best simulations to calculate convergence interval on next iteration (default 1).
  - *tolerance*: tolerance parameter to increase convergence interval (default 0).

  - *niterations*: number of iterations (default 1).
    It multiplies the total number of simulations:
    x (number of iterations)

- Moreover, both brute force algorithms can be coupled with a direction search method by using:
  - *direction*: method to estimate the optimal direction. Two options are currently available:
    * coordinates: coordinates descent method.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x 2 x (number of variables)
    * random: random method. It requires:
    * nestimates: number of random checks to estimate the optimal direction.
      It increases the total number of simulations by:
      (number of experiments) x (number of iterations) x (number of steps) x (number of estimates)

  Both methods require also:

  - nsteps: number of steps to perform the direction search method,
  - relaxation: relaxation parameter,

  and for each variable:

  - step: initial step size for the direction search method.

- **genetic**: Genetic algorithm. It requires the following parameters:
  - *npopulation*: number of population.
  - *ngenerations*: number of generations.
  - *mutation*: mutation ratio.
  - *reproduction*: reproduction ratio.
  - *adaptation*: adaptation ratio.

  and for each variable:

  - *nbits*: number of bits to encode each variable.

  The total number of simulations to run is:

  (number of experiments) x (npopulation) x [1 + (ngenerations - 1) x (mutation + reproduction + adaptation)]

Implemented error noms are:

- **euclidian**: Euclidian norm.
- **maximum**: maximum norm.
- **p**: p-norm. It requires the parameter:
  - *p*: p exponent.
- **taxicab**: Taxicab norm.

Alternatively, the input file can be also written in JSON format as:

```
"'json { "simulator": "simulator_name", "evaluator": "evaluator_name", "algorithm": "algorithm_type", "nsimulations"↩
: "simulations_number", "niterations": "iterations_number", "tolerance": "tolerance_value", "nbest": "best_↩
number", "npopulation": "population_number", "ngenerations": "generations_number", "mutation": "mutation_↩
ratio", "reproduction": "reproduction_ratio", "adaptation": "adaptation_ratio", "direction": "direction_search_type",
"nsteps": "steps_number", "relaxation": "relaxation_parameter", "nestimates": "estimates_number", "threshold":
"threshold_parameter", "norm": "norm_type", "p": "p_parameter", "seed": "random_seed", "result_file": "result↩
_file", "variables_file": "variables_file", "experiments": [ { "name": "data_file_1", "template1": "template_1_1",
"template2": "template_1_2", ... "weight": "weight_1", }, ... { "name": "data_file_N", "template1": "template_N_1",
"template2": "template_N_2", ... "weight": "weight_N", } ], "variables": [ {
```

```
"name": "variable_1",
"minimum": "min_value",
"maximum": "max_value",
"precision": "precision_digits",
"sweeps": "sweeps_number",
"nbits": "bits_number",
"step": "step_size",
```

}, ... { "name": "variable_M", "minimum": "min_value", "maximum": "max_value", "precision": "precision_digits", "sweeps": "sweeps_number", "nbits": "bits_number", "step": "step_size", }] } “‘

SOME EXAMPLES OF INPUT FILES

Example 1

- The simulator program name is: *pivot*
- The syntax is:

  $ ./pivot input_file output_file

- The program to evaluate the objective function is: *compare*
- The syntax is:

  $ ./compare simulated_file data_file result_file

- The calibration is performed with a *sweep brute force algorithm*.
- The experimental data files are:

  27-48.txt
  42.txt
  52.txt
  100.txt

- Templates to get input files to simulator for each experiment are:

  template1.js
  template2.js
  template3.js
  template4.js

- The variables to calibrate, ranges, precision and sweeps number to perform are:

  alpha1, [179.70, 180.20], 2, 5
  alpha2, [179.30, 179.60], 2, 5
  random, [0.00, 0.20], 2, 5
  boot-time, [0.0, 3.0], 1, 5

- Then, the number of simulations to run is: 4x5x5x5x5=2500.
- The input file is:

“‘xml <?xml version="1.0"?> <optimize simulator="pivot" evaluator="compare" algorithm="sweep"> <experiment name="27-48.txt" template1="template1.js"> <experiment name="42.txt" template1="template2.js"> <experiment name="52.txt" template1="template3.js"> <experiment name="100.txt" template1="template4.js"> <variable name="alpha1" minimum="179.70" maximum="180.20" precision="2" nsweeps="5"> <variable name="alpha2" minimum="179.30" maximum="179.60" precision="2" nsweeps="5"> <variable name="random" minimum="0.↩
00" maximum="0.20" precision="2" nsweeps="5"> <variable name="boot-time" minimum="0.0" maximum="3.0" precision="1" nsweeps="5"> </optimize> “‘

- A template file as *template1.js*:

“‘json { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.02824, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03008, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }, { "length" : 50.11, "velocity" : 0.03753, "@variable1@" : @, "@variable2@" : @, "@variable3@" : @, "@variable4@" : @ }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

- produces simulator input files to reproduce the experimental data file *27-48.txt* as:

“‘json { "towers" : [{ "length" : 50.11, "velocity" : 0.02738, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.02824, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03008, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }, { "length" : 50.11, "velocity" : 0.03753, "alpha1" : 179.95, "alpha2" : 179.45, "random" : 0.10, "boot-time" : 1.5 }], "cycle-time" : 71.0, "plot-time" : 1.0, "comp-time-step": 0.1, "active-percent" : 27.48 } “‘

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 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.

4.1.1 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

4.2 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.

4.2.1 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

4.3 Optimize Struct Reference

Struct to define the optimization ation data.

`#include <optimize.h>`

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.

4.3.1 Detailed Description

Struct to define the optimization ation data.

Definition at line 45 of file optimize.h.

4.3.2 Field Documentation

4.3.2.1 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

4.4 Options Struct Reference

Struct to define the options dialog.

`#include <interface.h>`

Data Fields

- GtkDialog * dialog

  Main GtkDialog.
- GtkGrid * grid

  Main GtkGrid.
- GtkLabel * label_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.

4.4.1 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

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

`#include <optimize.h>`

Data Fields

- unsigned int thread

  Thread number.

4.5.1 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

4.6 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.

4.6.1 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

4.7 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.

4.7.1 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

4.8 Window Struct Reference

Struct to define the main window.

`#include <interface.h>`

Collaboration diagram for Window:

Data Fields

- GtkWindow * window
  Main GtkWindow.
- GtkGrid * grid
  Main GtkGrid.
- GtkToolbar * bar_buttons
  GtkToolbar to store the main buttons.
- GtkToolButton * button_open
  Open GtkToolButton.
- GtkToolButton * button_save
  Save GtkToolButton.
- GtkToolButton * button_run
  Run GtkToolButton.
- GtkToolButton * button_options
  Options GtkToolButton.
- GtkToolButton * button_help
  Help GtkToolButton.
- GtkToolButton * button_about
  Help GtkToolButton.
- GtkToolButton * button_exit
  Exit GtkToolButton.
- GtkGrid * grid_files
  Files GtkGrid.
- GtkLabel * label_simulator
  Simulator program GtkLabel.
- GtkFileChooserButton * button_simulator
  Simulator program GtkFileChooserButton.
- GtkCheckButton * check_evaluator
  Evaluator program GtkCheckButton.
- GtkFileChooserButton * button_evaluator
  Evaluator program GtkFileChooserButton.
- GtkLabel * label_result
  Result file GtkLabel.
- GtkEntry * entry_result

Result file GtkEntry.

- GtkLabel * label_variables

  Variables file GtkLabel.
- GtkEntry * entry_variables

  Variables file GtkEntry.
- GtkFrame * frame_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.

4.8.1 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 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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

5.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2016, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00039 #ifndef CONFIG__H
00040 #define CONFIG__H 1
00041
00042 // Array sizes
00043
00044 #define MAX_NINPUTS 8
00045 #define NALGORITHMS 3
00047 #define NDIRECTIONS 2
00048 #define NNORMS 4
00049 #define NPRECISIONS 15
00050
00051 // Default choices
00052
00053 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00054 #define DEFAULT_RANDOM_SEED 7007
00055 #define DEFAULT_RELAXATION 1.
```

```
00056
00057 // Interface labels
00058
00059 #define LOCALE_DIR "locales"
00060 #define PROGRAM_INTERFACE "mpcotool"
00061
00062 // Labels
00063
00064 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00065 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00067 #define LABEL_ADAPTATION "adaptation"
00069 #define LABEL_ALGORITHM "algorithm"
00070 #define LABEL_OPTIMIZE "optimize"
00071 #define LABEL_COORDINATES "coordinates"
00072 #define LABEL_DIRECTION "direction"
00073 #define LABEL_EUCLIDIAN "euclidian"
00074 #define LABEL_EVALUATOR "evaluator"
00075 #define LABEL_EXPERIMENT "experiment"
00076 #define LABEL_EXPERIMENTS "experiments"
00077 #define LABEL_GENETIC "genetic"
00078 #define LABEL_MINIMUM "minimum"
00079 #define LABEL_MAXIMUM "maximum"
00080 #define LABEL_MONTE_CARLO "Monte-Carlo"
00081 #define LABEL_MUTATION "mutation"
00082 #define LABEL_NAME "name"
00083 #define LABEL_NBEST "nbest"
00084 #define LABEL_NBITS "nbits"
00085 #define LABEL_NESTIMATES "nestimates"
00086 #define LABEL_NGENERATIONS "ngenerations"
00087 #define LABEL_NITERATIONS "niterations"
00088 #define LABEL_NORM "norm"
00089 #define LABEL_NPOPULATION "npopulation"
00090 #define LABEL_NSIMULATIONS "nsimulations"
00091 #define LABEL_NSTEPS "nsteps"
00092 #define LABEL_NSWEEPS "nsweeps"
00093 #define LABEL_P "p"
00094 #define LABEL_PRECISION "precision"
00095 #define LABEL_RANDOM "random"
00096 #define LABEL_RELAXATION "relaxation"
00097 #define LABEL_REPRODUCTION "reproduction"
00098 #define LABEL_RESULT_FILE "result_file"
00099 #define LABEL_SIMULATOR "simulator"
00100 #define LABEL_SEED "seed"
00101 #define LABEL_STEP "step"
00102 #define LABEL_SWEEP "sweep"
00103 #define LABEL_TAXICAB "taxicab"
00104 #define LABEL_TEMPLATE1 "template1"
00105 #define LABEL_TEMPLATE2 "template2"
00106 #define LABEL_TEMPLATE3 "template3"
00107 #define LABEL_TEMPLATE4 "template4"
00108 #define LABEL_TEMPLATE5 "template5"
00109 #define LABEL_TEMPLATE6 "template6"
00110 #define LABEL_TEMPLATE7 "template7"
00111 #define LABEL_TEMPLATE8 "template8"
00112 #define LABEL_THRESHOLD "threshold"
00113 #define LABEL_TOLERANCE "tolerance"
00114 #define LABEL_VARIABLE "variable"
00115 #define LABEL_VARIABLES "variables"
00116 #define LABEL_VARIABLES_FILE "variables_file"
00117 #define LABEL_WEIGHT "weight"
00118
00119 // Enumerations
00120
00125 enum INPUT_TYPE
00126 {
00127   INPUT_TYPE_XML = 0,
00128   INPUT_TYPE_JSON = 1
00129 };
00130
00131 #endif
```

5.3 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 **_GNU_SOURCE**
- #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.

5.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file experiment.c.

5.3.2 Function Documentation

5.3.2.1 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", gettext ("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00128               message);
00129   error_message = g_strdup (buffer);
00130 }
```

5.3.2.2 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 }
```

5.3.2.3 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 }
```

5.3.2.4 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 252 of file experiment.c.

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

```
00317               experiment_error (experiment, gettext ("bad templates number"));
00318               goto exit_on_error;
00319             }
00320           name = json_object_get_string_member (object, template[i]);
00321 #if DEBUG_EXPERIMENT
00322           fprintf (stderr,
00323                    "experiment_open_json: experiment=%s template%u=%s\n",
00324                    experiment->nexperiments, name, template[i]);
00325 #endif
00326           experiment->template[i] = g_strdup (name);
00327           ++experiment->ninputs;
00328         }
00329       else if (ninputs && ninputs > i)
00330         {
00331           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00332           experiment_error (experiment, buffer);
00333           goto exit_on_error;
00334         }
00335       else
00336         break;
00337     }
00338
00339 #if DEBUG_EXPERIMENT
00340   fprintf (stderr, "experiment_open_json: end\n");
00341 #endif
00342   return 1;
00343
00344 exit_on_error:
00345   experiment_free (experiment, INPUT_TYPE_JSON);
00346 #if DEBUG_EXPERIMENT
00347   fprintf (stderr, "experiment_open_json: end\n");
00348 #endif
00349   return 0;
00350 }
```

Here is the call graph for this function:

5.3.2.5 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, gettext ("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     = xml_node_get_float_with_default (node, (const xmlChar *)
```

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

Here is the call graph for this function:

5.3.3 Variable Documentation

5.3.3.1 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.

5.4 experiment.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "utils.h"
00046 #include "experiment.h"
00047
00048 #define DEBUG_EXPERIMENT 0
00049
00050 const char *template[MAX_NINPUTS] = {
00051   LABEL_TEMPLATE1, LABEL_TEMPLATE2,
      LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00052   LABEL_TEMPLATE5, LABEL_TEMPLATE6,
      LABEL_TEMPLATE7, LABEL_TEMPLATE8
00053 };
00054
00056
00063 void
00064 experiment_new (Experiment * experiment)
00065 {
00066   unsigned int i;
00067 #if DEBUG_EXPERIMENT
00068   fprintf (stderr, "experiment_new: start\n");
00069 #endif
00070   experiment->name = NULL;
00071   experiment->ninputs = 0;
00072   for (i = 0; i < MAX_NINPUTS; ++i)
00073     experiment->template[i] = NULL;
00074 #if DEBUG_EXPERIMENT
00075   fprintf (stderr, "input_new: end\n");
00076 #endif
00077 }
00078
00087 void
00088 experiment_free (Experiment * experiment, unsigned int type)
00089 {
00090   unsigned int i;
00091 #if DEBUG_EXPERIMENT
00092   fprintf (stderr, "experiment_free: start\n");
```

```
00093 #endif
00094   if (type == INPUT_TYPE_XML)
00095     {
00096       for (i = 0; i < experiment->ninputs; ++i)
00097         xmlFree (experiment->template[i]);
00098       xmlFree (experiment->name);
00099     }
00100   else
00101     {
00102       for (i = 0; i < experiment->ninputs; ++i)
00103         g_free (experiment->template[i]);
00104       g_free (experiment->name);
00105     }
00106   experiment->ninputs = 0;
00107 #if DEBUG_EXPERIMENT
00108   fprintf (stderr, "experiment_free: end\n");
00109 #endif
00110 }
00111
00120 void
00121 experiment_error (Experiment * experiment, char *message)
00122 {
00123   char buffer[64];
00124   if (!experiment->name)
00125     snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00128               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, gettext ("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     = xml_node_get_float_with_default (node, (const xmlChar *)
    LABEL_WEIGHT, 1.,
00171                                        &error_code);
00172   if (error_code)
00173     {
00174       experiment_error (experiment, gettext ("bad weight"));
00175       goto exit_on_error;
00176     }
00177 #if DEBUG_EXPERIMENT
00178   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00179 #endif
00180   experiment->template[0]
00181     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00182   if (experiment->template[0])
00183     {
00184 #if DEBUG_EXPERIMENT
00185       fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00186                experiment->name, template[0]);
00187 #endif
00188       ++experiment->ninputs;
00189     }
00190   else
00191     {
00192       experiment_error (experiment, gettext ("no template"));
00193       goto exit_on_error;
00194     }
00195   for (i = 1; i < MAX_NINPUTS; ++i)
00196     {
00197 #if DEBUG_EXPERIMENT
00198       fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
```

```
00199 #endif
00200       if (xmlHasProp (node, (const xmlChar *) template[i]))
00201         {
00202           if (ninputs && ninputs <= i)
00203             {
00204               experiment_error (experiment, gettext ("bad templates number"));
00205               goto exit_on_error;
00206             }
00207           experiment->template[i]
00208             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00209 #if DEBUG_EXPERIMENT
00210           fprintf (stderr, "experiment_open_xml: experiment=%s template%u=%s\n",
00211                    experiment->nexperiments, experiment->name,
00212                    experiment->template[i]);
00213 #endif
00214           ++experiment->ninputs;
00215         }
00216       else if (ninputs && ninputs > i)
00217         {
00218           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00219           experiment_error (experiment, buffer);
00220           goto exit_on_error;
00221         }
00222       else
00223         break;
00224     }
00225
00226 #if DEBUG_EXPERIMENT
00227   fprintf (stderr, "experiment_open_xml: end\n");
00228 #endif
00229   return 1;
00230
00231 exit_on_error:
00232   experiment_free (experiment, INPUT_TYPE_XML);
00233 #if DEBUG_EXPERIMENT
00234   fprintf (stderr, "experiment_open_xml: end\n");
00235 #endif
00236   return 0;
00237 }
00238
00251 int
00252 experiment_open_json (Experiment * experiment, JsonNode * node,
00253                       unsigned int ninputs)
00254 {
00255   char buffer[64];
00256   JsonObject *object;
00257   const char *name;
00258   int error_code;
00259   unsigned int i;
00260
00261 #if DEBUG_EXPERIMENT
00262   fprintf (stderr, "experiment_open_json: start\n");
00263 #endif
00264
00265   // Resetting experiment data
00266   experiment_new (experiment);
00267
00268   // Getting JSON object
00269   object = json_node_get_object (node);
00270
00271   // Reading the experimental data
00272   name = json_object_get_string_member (object, LABEL_NAME);
00273   if (!name)
00274     {
00275       experiment_error (experiment, gettext ("no data file name"));
00276       goto exit_on_error;
00277     }
00278   experiment->name = g_strdup (name);
00279 #if DEBUG_EXPERIMENT
00280   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00281 #endif
00282   experiment->weight
00283     = json_object_get_float_with_default (object,
    LABEL_WEIGHT, 1.,
00284                                           &error_code);
00285   if (error_code)
00286     {
00287       experiment_error (experiment, gettext ("bad weight"));
00288       goto exit_on_error;
00289     }
00290 #if DEBUG_EXPERIMENT
00291   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00292 #endif
00293   name = json_object_get_string_member (object, template[0]);
00294   if (name)
00295     {
00296 #if DEBUG_EXPERIMENT
```

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

5.5 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.

5.5.1 Detailed Description

Header file to define the experiment data.

Authors

Javier Burguete.

Copyright

Definition in file experiment.h.

5.5.2 Function Documentation

5.5.2.1 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", gettext ("Experiment"), message);
00126   else
00127     snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00128               message);
00129   error_message = g_strdup (buffer);
00130 }
```

5.5.2.2 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 }
```

5.5.2.3 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 }
```

5.5.2.4 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 252 of file experiment.c.

```
00254 {
00255   char buffer[64];
00256   JsonObject *object;
00257   const char *name;
00258   int error_code;
00259   unsigned int i;
00260
00261 #if DEBUG_EXPERIMENT
00262   fprintf (stderr, "experiment_open_json: start\n");
00263 #endif
00264
00265   // Resetting experiment data
00266   experiment_new (experiment);
00267
00268   // Getting JSON object
00269   object = json_node_get_object (node);
00270
00271   // Reading the experimental data
00272   name = json_object_get_string_member (object, LABEL_NAME);
00273   if (!name)
00274     {
00275       experiment_error (experiment, gettext ("no data file name"));
00276       goto exit_on_error;
00277     }
00278   experiment->name = g_strdup (name);
00279 #if DEBUG_EXPERIMENT
00280   fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00281 #endif
00282   experiment->weight
00283     = json_object_get_float_with_default (object,
      LABEL_WEIGHT, 1.,
00284                                           &error_code);
00285   if (error_code)
00286     {
00287       experiment_error (experiment, gettext ("bad weight"));
00288       goto exit_on_error;
00289     }
00290 #if DEBUG_EXPERIMENT
00291   fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00292 #endif
00293   name = json_object_get_string_member (object, template[0]);
00294   if (name)
00295     {
00296 #if DEBUG_EXPERIMENT
00297       fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00298                name, template[0]);
00299 #endif
00300       ++experiment->ninputs;
00301     }
00302   else
00303     {
00304       experiment_error (experiment, gettext ("no template"));
00305       goto exit_on_error;
00306     }
00307   experiment->template[0] = g_strdup (name);
00308   for (i = 1; i < MAX_NINPUTS; ++i)
00309     {
00310 #if DEBUG_EXPERIMENT
00311       fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00312 #endif
00313       if (json_object_get_member (object, template[i]))
00314         {
00315           if (ninputs && ninputs <= i)
00316             {
00317               experiment_error (experiment, gettext ("bad templates number"));
00318               goto exit_on_error;
00319             }
00320           name = json_object_get_string_member (object, template[i]);
00321 #if DEBUG_EXPERIMENT
00322           fprintf (stderr,
00323                    "experiment_open_json: experiment=%s template%u=%s\n",
00324                    experiment->nexperiments, name, template[i]);
00325 #endif
00326           experiment->template[i] = g_strdup (name);
00327           ++experiment->ninputs;
00328         }
00329       else if (ninputs && ninputs > i)
00330         {
00331           snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00332           experiment_error (experiment, buffer);
00333           goto exit_on_error;
00334         }
00335       else
00336         break;
00337     }
00338
00339 #if DEBUG_EXPERIMENT
```

```
00340   fprintf (stderr, "experiment_open_json: end\n");
00341 #endif
00342   return 1;
00343
00344 exit_on_error:
00345   experiment_free (experiment, INPUT_TYPE_JSON);
00346 #if DEBUG_EXPERIMENT
00347   fprintf (stderr, "experiment_open_json: end\n");
00348 #endif
00349   return 0;
00350 }
```

Here is the call graph for this function:

5.5.2.5 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, gettext ("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     = xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_WEIGHT, 1.,
00171                                        &error_code);
00172   if (error_code)
00173     {
00174       experiment_error (experiment, gettext ("bad weight"));
00175       goto exit_on_error;
00176     }
00177 #if DEBUG_EXPERIMENT
00178   fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00179 #endif
00180   experiment->template[0]
00181     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00182   if (experiment->template[0])
00183     {
00184 #if DEBUG_EXPERIMENT
00185       fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00186                experiment->name, template[0]);
00187 #endif
00188       ++experiment->ninputs;
00189     }
00190   else
00191     {
00192       experiment_error (experiment, gettext ("no template"));
```

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

Here is the call graph for this function:

5.6 experiment.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef EXPERIMENT__H
```

```
00039 #define EXPERIMENT__H 1
00040
00045 typedef struct
00046 {
00047   char *name;
00048   char *template[MAX_NINPUTS];
00049   double weight;
00050   unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const char *template[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_new (Experiment * experiment);
00057 void experiment_free (Experiment * experiment, unsigned int type);
00058 void experiment_error (Experiment * experiment, char *message);
00059 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00060                          unsigned int ninputs);
00061 int experiment_open_json (Experiment * experiment, JsonNode * node,
00062                           unsigned int ninputs);
00063
00064 #endif
```

5.7 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 **_GNU_SOURCE**
- #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.

5.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file input.c.

5.7.2 Function Documentation

5.7.2.1 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", gettext ("Input"), message);
00128   error_message = g_strdup (buffer);
00129 }
```

5.7.2.2 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 947 of file input.c.

```
00948 {
00949   xmlDoc *doc;
00950   JsonParser *parser;
00951
00952 #if DEBUG_INPUT
00953   fprintf (stderr, "input_open: start\n");
00954 #endif
00955
00956   // Resetting input data
00957   input_new ();
00958
00959   // Opening input file
00960 #if DEBUG_INPUT
00961   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00962   fprintf (stderr, "input_open: trying XML format\n");
00963 #endif
00964   doc = xmlParseFile (filename);
00965   if (!doc)
00966     {
00967 #if DEBUG_INPUT
00968       fprintf (stderr, "input_open: trying JSON format\n");
00969 #endif
00970       parser = json_parser_new ();
00971       if (!json_parser_load_from_file (parser, filename, NULL))
00972         {
00973           input_error (gettext ("Unable to parse the input file"));
00974           goto exit_on_error;
00975         }
00976       if (!input_open_json (parser))
00977         goto exit_on_error;
00978     }
00979   else if (!input_open_xml (doc))
00980     goto exit_on_error;
00981
00982   // Getting the working directory
00983   input->directory = g_path_get_dirname (filename);
00984   input->name = g_path_get_basename (filename);
00985
00986 #if DEBUG_INPUT
00987   fprintf (stderr, "input_open: end\n");
00988 #endif
00989   return 1;
00990
00991 exit_on_error:
00992   show_error (error_message);
00993   g_free (error_message);
00994   input_free ();
00995 #if DEBUG_INPUT
00996   fprintf (stderr, "input_open: end\n");
00997 #endif
00998   return 0;
00999 }
```

Here is the call graph for this function:

5.7.2.3 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 557 of file input.c.

```
00558 {
00559   JsonNode *node, *child;
00560   JsonObject *object;
00561   JsonArray *array;
00562   const char *buffer;
00563   int error_code;
00564   unsigned int i, n;
00565
00566 #if DEBUG_INPUT
```

```
00567   fprintf (stderr, "input_open_json: start\n");
00568 #endif
00569
00570   // Resetting input data
00571   input->type = INPUT_TYPE_JSON;
00572
00573   // Getting the root node
00574 #if DEBUG_INPUT
00575   fprintf (stderr, "input_open_json: getting the root node\n");
00576 #endif
00577   node = json_parser_get_root (parser);
00578   object = json_node_get_object (node);
00579
00580   // Getting result and variables file names
00581   if (!input->result)
00582     {
00583       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00584       if (!buffer)
00585         buffer = result_name;
00586       input->result = g_strdup (buffer);
00587     }
00588   else
00589     input->result = g_strdup (result_name);
00590   if (!input->variables)
00591     {
00592       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00593       if (!buffer)
00594         buffer = variables_name;
00595       input->variables = g_strdup (buffer);
00596     }
00597   else
00598     input->variables = g_strdup (variables_name);
00599
00600   // Opening simulator program name
00601   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00602   if (!buffer)
00603     {
00604       input_error (gettext ("Bad simulator program"));
00605       goto exit_on_error;
00606     }
00607   input->simulator = g_strdup (buffer);
00608
00609   // Opening evaluator program name
00610   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00611   if (buffer)
00612     input->evaluator = g_strdup (buffer);
00613
00614   // Obtaining pseudo-random numbers generator seed
00615   input->seed
00616     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00617                                          DEFAULT_RANDOM_SEED, &error_code);
00618   if (error_code)
00619     {
00620       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00621       goto exit_on_error;
00622     }
00623
00624   // Opening algorithm
00625   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00626   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627     {
00628       input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630       // Obtaining simulations number
00631       input->nsimulations
00632         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00633       if (error_code)
00634         {
00635           input_error (gettext ("Bad simulations number"));
00636           goto exit_on_error;
00637         }
00638     }
00639   else if (!strcmp (buffer, LABEL_SWEEP))
00640     input->algorithm = ALGORITHM_SWEEP;
00641   else if (!strcmp (buffer, LABEL_GENETIC))
00642     {
00643       input->algorithm = ALGORITHM_GENETIC;
00644
00645       // Obtaining population
00646       if (json_object_get_member (object, LABEL_NPOPULATION))
00647         {
00648           input->nsimulations
00649             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00650           if (error_code || input->nsimulations < 3)
```

```
00651             {
00652               input_error (gettext ("Invalid population number"));
00653               goto exit_on_error;
00654             }
00655         }
00656       else
00657         {
00658           input_error (gettext ("No population number"));
00659           goto exit_on_error;
00660         }
00661
00662       // Obtaining generations
00663       if (json_object_get_member (object, LABEL_NGENERATIONS))
00664         {
00665           input->niterations
00666             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00667           if (error_code || !input->niterations)
00668             {
00669               input_error (gettext ("Invalid generations number"));
00670               goto exit_on_error;
00671             }
00672         }
00673       else
00674         {
00675           input_error (gettext ("No generations number"));
00676           goto exit_on_error;
00677         }
00678
00679       // Obtaining mutation probability
00680       if (json_object_get_member (object, LABEL_MUTATION))
00681         {
00682           input->mutation_ratio
00683             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00684           if (error_code || input->mutation_ratio < 0.
00685               || input->mutation_ratio >= 1.)
00686             {
00687               input_error (gettext ("Invalid mutation probability"));
00688               goto exit_on_error;
00689             }
00690         }
00691       else
00692         {
00693           input_error (gettext ("No mutation probability"));
00694           goto exit_on_error;
00695         }
00696
00697       // Obtaining reproduction probability
00698       if (json_object_get_member (object, LABEL_REPRODUCTION))
00699         {
00700           input->reproduction_ratio
00701             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00702           if (error_code || input->reproduction_ratio < 0.
00703               || input->reproduction_ratio >= 1.0)
00704             {
00705               input_error (gettext ("Invalid reproduction probability"));
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         {
00711           input_error (gettext ("No reproduction probability"));
00712           goto exit_on_error;
00713         }
00714
00715       // Obtaining adaptation probability
00716       if (json_object_get_member (object, LABEL_ADAPTATION))
00717         {
00718           input->adaptation_ratio
00719             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00720           if (error_code || input->adaptation_ratio < 0.
00721               || input->adaptation_ratio >= 1.)
00722             {
00723               input_error (gettext ("Invalid adaptation probability"));
00724               goto exit_on_error;
00725             }
00726         }
00727       else
00728         {
00729           input_error (gettext ("No adaptation probability"));
00730           goto exit_on_error;
00731         }
00732
00733       // Checking survivals
```

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

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

```
00892     }
00893
00894   // Obtaining the error norm
00895   if (json_object_get_member (object, LABEL_NORM))
00896     {
00897       buffer = json_object_get_string_member (object, LABEL_NORM);
00898       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00899         input->norm = ERROR_NORM_EUCLIDIAN;
00900       else if (!strcmp (buffer, LABEL_MAXIMUM))
00901         input->norm = ERROR_NORM_MAXIMUM;
00902       else if (!strcmp (buffer, LABEL_P))
00903         {
00904           input->norm = ERROR_NORM_P;
00905           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00906           if (!error_code)
00907             {
00908               input_error (gettext ("Bad P parameter"));
00909               goto exit_on_error;
00910             }
00911         }
00912       else if (!strcmp (buffer, LABEL_TAXICAB))
00913         input->norm = ERROR_NORM_TAXICAB;
00914       else
00915         {
00916           input_error (gettext ("Unknown error norm"));
00917           goto exit_on_error;
00918         }
00919     }
00920   else
00921     input->norm = ERROR_NORM_EUCLIDIAN;
00922
00923   // Closing the JSON document
00924   g_object_unref (parser);
00925
00926 #if DEBUG_INPUT
00927   fprintf (stderr, "input_open_json: end\n");
00928 #endif
00929   return 1;
00930
00931 exit_on_error:
00932   g_object_unref (parser);
00933 #if DEBUG_INPUT
00934   fprintf (stderr, "input_open_json: end\n");
00935 #endif
00936   return 0;
00937 }
```

Here is the call graph for this function:

5.7.2.4 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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("Invalid population number"));
```

```
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (gettext ("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 (gettext ("Invalid generations number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           input_error (gettext ("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 (gettext ("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
00277       else
00278         {
00279           input_error (gettext ("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 (gettext ("Invalid reproduction probability"));
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (gettext ("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 (gettext ("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (gettext ("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             (gettext
00329              ("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (gettext ("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP)
00343     {
00344
00345       // Obtaining iterations number
00346       input->niterations
00347         = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NITERATIONS,
00348                              &error_code);
00349       if (error_code == 1)
00350         input->niterations = 1;
00351       else if (error_code)
00352         {
00353           input_error (gettext ("Bad iterations number"));
00354           goto exit_on_error;
00355         }
00356
00357       // Obtaining best number
00358       input->nbest
00359         = xml_node_get_uint_with_default (node, (const xmlChar *)
    LABEL_NBEST,
00360                                           1, &error_code);
00361       if (error_code || !input->nbest)
00362         {
00363           input_error (gettext ("Invalid best number"));
00364           goto exit_on_error;
00365         }
00366
00367       // Obtaining tolerance
00368       input->tolerance
00369         = xml_node_get_float_with_default (node,
00370                                            (const xmlChar *) LABEL_TOLERANCE,
00371                                            0., &error_code);
00372       if (error_code || input->tolerance < 0.)
00373         {
00374           input_error (gettext ("Invalid tolerance"));
00375           goto exit_on_error;
00376         }
00377
00378       // Getting direction search method parameters
00379       if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00380         {
00381           input->nsteps =
00382             xml_node_get_uint (node, (const xmlChar *)
    LABEL_NSTEPS,
00383                                &error_code);
00384           if (error_code || !input->nsteps)
00385             {
00386               input_error (gettext ("Invalid steps number"));
00387               goto exit_on_error;
00388             }
00389           buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00390           if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00391             input->direction = DIRECTION_METHOD_COORDINATES;
00392           else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00393             {
00394               input->direction = DIRECTION_METHOD_RANDOM;
00395               input->nestimates
00396                 = xml_node_get_uint (node, (const xmlChar *)
    LABEL_NESTIMATES,
00397                                      &error_code);
00398               if (error_code || !input->nestimates)
00399                 {
00400                   input_error (gettext ("Invalid estimates number"));
```

```
00401                   goto exit_on_error;
00402                 }
00403             }
00404           else
00405             {
00406               input_error
00407                 (gettext ("Unknown method to estimate the direction search"));
00408               goto exit_on_error;
00409             }
00410           xmlFree (buffer);
00411           buffer = NULL;
00412           input->relaxation
00413             = xml_node_get_float_with_default (node,
00414                                                (const xmlChar *)
00415                                                LABEL_RELAXATION,
00416                                                DEFAULT_RELAXATION, &error_code);
00417           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00418             {
00419               input_error (gettext ("Invalid relaxation parameter"));
00420               goto exit_on_error;
00421             }
00422         }
00423       else
00424         input->nsteps = 0;
00425     }
00426   // Obtaining the threshold
00427   input->threshold =
00428     xml_node_get_float_with_default (node, (const xmlChar *)
      LABEL_THRESHOLD,
00429                                      0., &error_code);
00430   if (error_code)
00431     {
00432       input_error (gettext ("Invalid threshold"));
00433       goto exit_on_error;
00434     }
00435
00436   // Reading the experimental data
00437   for (child = node->children; child; child = child->next)
00438     {
00439       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00440         break;
00441 #if DEBUG_INPUT
00442       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00443                input->nexperiments);
00444 #endif
00445       input->experiment = (Experiment *)
00446         g_realloc (input->experiment,
00447                    (1 + input->nexperiments) * sizeof (
      Experiment));
00448       if (!input->nexperiments)
00449         {
00450           if (!experiment_open_xml (input->experiment, child, 0))
00451             goto exit_on_error;
00452         }
00453       else
00454         {
00455           if (!experiment_open_xml (input->experiment +
      input->nexperiments,
00456                                     child, input->experiment->
      ninputs))
00457             goto exit_on_error;
00458         }
00459       ++input->nexperiments;
00460 #if DEBUG_INPUT
00461       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00462                input->nexperiments);
00463 #endif
00464     }
00465   if (!input->nexperiments)
00466     {
00467       input_error (gettext ("No optimization experiments"));
00468       goto exit_on_error;
00469     }
00470   buffer = NULL;
00471
00472   // Reading the variables data
00473   for (; child; child = child->next)
00474     {
00475 #if DEBUG_INPUT
00476       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00477 #endif
00478       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00479         {
00480           snprintf (buffer2, 64, "%s %u: %s",
00481                     gettext ("Variable"),
00482                     input->nvariables + 1, gettext ("bad XML node"));
```

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

Here is the call graph for this function:

5.8 input.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
```

```
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <string.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <glib/gstdio.h>
00046 #include <json-glib/json-glib.h>
00047 #include "utils.h"
00048 #include "experiment.h"
00049 #include "variable.h"
00050 #include "input.h"
00051
00052 #define DEBUG_INPUT 0
00053
00054 Input input[1];
00055
00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
00063 void
00064 input_new ()
00065 {
00066 #if DEBUG_INPUT
00067   fprintf (stderr, "input_new: start\n");
00068 #endif
00069   input->nvariables = input->nexperiments = input->nsteps = 0;
00070   input->simulator = input->evaluator = input->directory = input->
      name = NULL;
00071   input->experiment = NULL;
00072   input->variable = NULL;
00073 #if DEBUG_INPUT
00074   fprintf (stderr, "input_new: end\n");
00075 #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085   unsigned int i;
00086 #if DEBUG_INPUT
00087   fprintf (stderr, "input_free: start\n");
00088 #endif
00089   g_free (input->name);
00090   g_free (input->directory);
00091   for (i = 0; i < input->nexperiments; ++i)
00092     experiment_free (input->experiment + i, input->type);
00093   for (i = 0; i < input->nvariables; ++i)
00094     variable_free (input->variable + i, input->type);
00095   g_free (input->experiment);
00096   g_free (input->variable);
00097   if (input->type == INPUT_TYPE_XML)
00098     {
00099       xmlFree (input->evaluator);
00100       xmlFree (input->simulator);
00101       xmlFree (input->result);
00102       xmlFree (input->variables);
00103     }
00104   else
00105     {
00106       g_free (input->evaluator);
00107       g_free (input->simulator);
00108       g_free (input->result);
00109       g_free (input->variables);
00110     }
```

```
00111   input->nexperiments = input->nvariables = input->nsteps = 0;
00112 #if DEBUG_INPUT
00113   fprintf (stderr, "input_free: end\n");
00114 #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126   char buffer[64];
00127   snprintf (buffer, 64, "%s: %s\n", gettext ("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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("Invalid population number"));
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (gettext ("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 (gettext ("Invalid generations number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           input_error (gettext ("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 (gettext ("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
00277       else
00278         {
00279           input_error (gettext ("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 (gettext ("Invalid reproduction probability"));
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (gettext ("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 (gettext ("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (gettext ("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             (gettext
00329              ("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (gettext ("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP)
00343     {
00344
00345       // Obtaining iterations number
00346       input->niterations
00347         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00348                              &error_code);
00349       if (error_code == 1)
00350         input->niterations = 1;
00351       else if (error_code)
00352         {
00353           input_error (gettext ("Bad iterations number"));
00354           goto exit_on_error;
00355         }
00356
00357       // Obtaining best number
00358       input->nbest
00359         = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_NBEST,
00360                                           1, &error_code);
00361       if (error_code || !input->nbest)
00362         {
00363           input_error (gettext ("Invalid best number"));
00364           goto exit_on_error;
00365         }
00366
00367       // Obtaining tolerance
00368       input->tolerance
00369         = xml_node_get_float_with_default (node,
00370                                            (const xmlChar *) LABEL_TOLERANCE,
00371                                            0., &error_code);
00372       if (error_code || input->tolerance < 0.)
00373         {
00374           input_error (gettext ("Invalid tolerance"));
00375           goto exit_on_error;
```

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

```
00458         }
00459       ++input->nexperiments;
00460 #if DEBUG_INPUT
00461       fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00462                input->nexperiments);
00463 #endif
00464     }
00465   if (!input->nexperiments)
00466     {
00467       input_error (gettext ("No optimization experiments"));
00468       goto exit_on_error;
00469     }
00470   buffer = NULL;
00471
00472   // Reading the variables data
00473   for (; child; child = child->next)
00474     {
00475 #if DEBUG_INPUT
00476       fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00477 #endif
00478       if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00479         {
00480           snprintf (buffer2, 64, "%s %u: %s",
00481                     gettext ("Variable"),
00482                     input->nvariables + 1, gettext ("bad XML node"));
00483           input_error (buffer2);
00484           goto exit_on_error;
00485         }
00486       input->variable = (Variable *)
00487         g_realloc (input->variable,
00488                    (1 + input->nvariables) * sizeof (Variable));
00489       if (!variable_open_xml (input->variable + input->
      nvariables, child,
00490                               input->algorithm, input->nsteps))
00491         goto exit_on_error;
00492       ++input->nvariables;
00493     }
00494   if (!input->nvariables)
00495     {
00496       input_error (gettext ("No optimization variables"));
00497       goto exit_on_error;
00498     }
00499   buffer = NULL;
00500
00501   // Obtaining the error norm
00502   if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00503     {
00504       buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00505       if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00506         input->norm = ERROR_NORM_EUCLIDIAN;
00507       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00508         input->norm = ERROR_NORM_MAXIMUM;
00509       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00510         {
00511           input->norm = ERROR_NORM_P;
00512           input->p
00513             = xml_node_get_float (node, (const xmlChar *)
      LABEL_P, &error_code);
00514           if (!error_code)
00515             {
00516               input_error (gettext ("Bad P parameter"));
00517               goto exit_on_error;
00518             }
00519         }
00520       else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00521         input->norm = ERROR_NORM_TAXICAB;
00522       else
00523         {
00524           input_error (gettext ("Unknown error norm"));
00525           goto exit_on_error;
00526         }
00527       xmlFree (buffer);
00528     }
00529   else
00530     input->norm = ERROR_NORM_EUCLIDIAN;
00531
00532   // Closing the XML document
00533   xmlFreeDoc (doc);
00534
00535 #if DEBUG_INPUT
00536   fprintf (stderr, "input_open_xml: end\n");
00537 #endif
00538   return 1;
00539
00540 exit_on_error:
00541   xmlFree (buffer);
00542   xmlFreeDoc (doc);
```

```
00543 #if DEBUG_INPUT
00544   fprintf (stderr, "input_open_xml: end\n");
00545 #endif
00546   return 0;
00547 }
00548
00556 int
00557 input_open_json (JsonParser * parser)
00558 {
00559   JsonNode *node, *child;
00560   JsonObject *object;
00561   JsonArray *array;
00562   const char *buffer;
00563   int error_code;
00564   unsigned int i, n;
00565
00566 #if DEBUG_INPUT
00567   fprintf (stderr, "input_open_json: start\n");
00568 #endif
00569
00570   // Resetting input data
00571   input->type = INPUT_TYPE_JSON;
00572
00573   // Getting the root node
00574 #if DEBUG_INPUT
00575   fprintf (stderr, "input_open_json: getting the root node\n");
00576 #endif
00577   node = json_parser_get_root (parser);
00578   object = json_node_get_object (node);
00579
00580   // Getting result and variables file names
00581   if (!input->result)
00582     {
00583       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00584       if (!buffer)
00585         buffer = result_name;
00586       input->result = g_strdup (buffer);
00587     }
00588   else
00589     input->result = g_strdup (result_name);
00590   if (!input->variables)
00591     {
00592       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00593       if (!buffer)
00594         buffer = variables_name;
00595       input->variables = g_strdup (buffer);
00596     }
00597   else
00598     input->variables = g_strdup (variables_name);
00599
00600   // Opening simulator program name
00601   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00602   if (!buffer)
00603     {
00604       input_error (gettext ("Bad simulator program"));
00605       goto exit_on_error;
00606     }
00607   input->simulator = g_strdup (buffer);
00608
00609   // Opening evaluator program name
00610   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00611   if (buffer)
00612     input->evaluator = g_strdup (buffer);
00613
00614   // Obtaining pseudo-random numbers generator seed
00615   input->seed
00616     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00617                                          DEFAULT_RANDOM_SEED, &error_code);
00618   if (error_code)
00619     {
00620       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00621       goto exit_on_error;
00622     }
00623
00624   // Opening algorithm
00625   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00626   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627     {
00628       input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630       // Obtaining simulations number
00631       input->nsimulations
00632         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00633       if (error_code)
00634         {
```

```
00635           input_error (gettext ("Bad simulations number"));
00636           goto exit_on_error;
00637         }
00638     }
00639   else if (!strcmp (buffer, LABEL_SWEEP))
00640     input->algorithm = ALGORITHM_SWEEP;
00641   else if (!strcmp (buffer, LABEL_GENETIC))
00642     {
00643       input->algorithm = ALGORITHM_GENETIC;
00644
00645       // Obtaining population
00646       if (json_object_get_member (object, LABEL_NPOPULATION))
00647         {
00648           input->nsimulations
00649             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00650           if (error_code || input->nsimulations < 3)
00651             {
00652               input_error (gettext ("Invalid population number"));
00653               goto exit_on_error;
00654             }
00655         }
00656       else
00657         {
00658           input_error (gettext ("No population number"));
00659           goto exit_on_error;
00660         }
00661
00662       // Obtaining generations
00663       if (json_object_get_member (object, LABEL_NGENERATIONS))
00664         {
00665           input->niterations
00666             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00667           if (error_code || !input->niterations)
00668             {
00669               input_error (gettext ("Invalid generations number"));
00670               goto exit_on_error;
00671             }
00672         }
00673       else
00674         {
00675           input_error (gettext ("No generations number"));
00676           goto exit_on_error;
00677         }
00678
00679       // Obtaining mutation probability
00680       if (json_object_get_member (object, LABEL_MUTATION))
00681         {
00682           input->mutation_ratio
00683             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00684           if (error_code || input->mutation_ratio < 0.
00685               || input->mutation_ratio >= 1.)
00686             {
00687               input_error (gettext ("Invalid mutation probability"));
00688               goto exit_on_error;
00689             }
00690         }
00691       else
00692         {
00693           input_error (gettext ("No mutation probability"));
00694           goto exit_on_error;
00695         }
00696
00697       // Obtaining reproduction probability
00698       if (json_object_get_member (object, LABEL_REPRODUCTION))
00699         {
00700           input->reproduction_ratio
00701             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00702           if (error_code || input->reproduction_ratio < 0.
00703               || input->reproduction_ratio >= 1.0)
00704             {
00705               input_error (gettext ("Invalid reproduction probability"));
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         {
00711           input_error (gettext ("No reproduction probability"));
00712           goto exit_on_error;
00713         }
00714
00715       // Obtaining adaptation probability
00716       if (json_object_get_member (object, LABEL_ADAPTATION))
00717         {
```

```
00718           input->adaptation_ratio
00719             = json_object_get_float (object,
    LABEL_ADAPTATION, &error_code);
00720           if (error_code || input->adaptation_ratio < 0.
00721               || input->adaptation_ratio >= 1.)
00722             {
00723               input_error (gettext ("Invalid adaptation probability"));
00724               goto exit_on_error;
00725             }
00726         }
00727       else
00728         {
00729           input_error (gettext ("No adaptation probability"));
00730           goto exit_on_error;
00731         }
00732
00733       // Checking survivals
00734       i = input->mutation_ratio * input->nsimulations;
00735       i += input->reproduction_ratio * input->nsimulations;
00736       i += input->adaptation_ratio * input->nsimulations;
00737       if (i > input->nsimulations - 2)
00738         {
00739           input_error
00740             (gettext
00741              ("No enough survival entities to reproduce the population"));
00742           goto exit_on_error;
00743         }
00744     }
00745   else
00746     {
00747       input_error (gettext ("Unknown algorithm"));
00748       goto exit_on_error;
00749     }
00750
00751   if (input->algorithm == ALGORITHM_MONTE_CARLO
00752       || input->algorithm == ALGORITHM_SWEEP)
00753     {
00754
00755       // Obtaining iterations number
00756       input->niterations
00757         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
    );
00758       if (error_code == 1)
00759         input->niterations = 1;
00760       else if (error_code)
00761         {
00762           input_error (gettext ("Bad iterations number"));
00763           goto exit_on_error;
00764         }
00765
00766       // Obtaining best number
00767       input->nbest
00768         = json_object_get_uint_with_default (object,
    LABEL_NBEST, 1,
00769                                              &error_code);
00770       if (error_code || !input->nbest)
00771         {
00772           input_error (gettext ("Invalid best number"));
00773           goto exit_on_error;
00774         }
00775
00776       // Obtaining tolerance
00777       input->tolerance
00778         = json_object_get_float_with_default (object,
    LABEL_TOLERANCE, 0.,
00779                                               &error_code);
00780       if (error_code || input->tolerance < 0.)
00781         {
00782           input_error (gettext ("Invalid tolerance"));
00783           goto exit_on_error;
00784         }
00785
00786       // Getting direction search method parameters
00787       if (json_object_get_member (object, LABEL_NSTEPS))
00788         {
00789           input->nsteps
00790             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00791           if (error_code || !input->nsteps)
00792             {
00793               input_error (gettext ("Invalid steps number"));
00794               goto exit_on_error;
00795             }
00796           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00797           if (!strcmp (buffer, LABEL_COORDINATES))
00798             input->direction = DIRECTION_METHOD_COORDINATES;
00799           else if (!strcmp (buffer, LABEL_RANDOM))
00800             {
```

```
00801               input->direction = DIRECTION_METHOD_RANDOM;
00802               input->nestimates
00803                 = json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00804               if (error_code || !input->nestimates)
00805                 {
00806                   input_error (gettext ("Invalid estimates number"));
00807                   goto exit_on_error;
00808                 }
00809             }
00810           else
00811             {
00812               input_error
00813                 (gettext ("Unknown method to estimate the direction search"));
00814               goto exit_on_error;
00815             }
00816           input->relaxation
00817             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00818                                                   DEFAULT_RELAXATION,
00819                                                   &error_code);
00820           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00821             {
00822               input_error (gettext ("Invalid relaxation parameter"));
00823               goto exit_on_error;
00824             }
00825         }
00826       else
00827         input->nsteps = 0;
00828     }
00829   // Obtaining the threshold
00830   input->threshold
00831     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00832                                           &error_code);
00833   if (error_code)
00834     {
00835       input_error (gettext ("Invalid threshold"));
00836       goto exit_on_error;
00837     }
00838
00839   // Reading the experimental data
00840   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00841   n = json_array_get_length (array);
00842   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00843   for (i = 0; i < n; ++i)
00844     {
00845 #if DEBUG_INPUT
00846       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00847                input->nexperiments);
00848 #endif
00849       child = json_array_get_element (array, i);
00850       if (!input->nexperiments)
00851         {
00852           if (!experiment_open_json (input->experiment, child, 0))
00853             goto exit_on_error;
00854         }
00855       else
00856         {
00857           if (!experiment_open_json (input->experiment + input->
      nexperiments,
00858                                      child, input->experiment->ninputs))
00859             goto exit_on_error;
00860         }
00861       ++input->nexperiments;
00862 #if DEBUG_INPUT
00863       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00864                input->nexperiments);
00865 #endif
00866     }
00867   if (!input->nexperiments)
00868     {
00869       input_error (gettext ("No optimization experiments"));
00870       goto exit_on_error;
00871     }
00872
00873   // Reading the variables data
00874   array = json_object_get_array_member (object, LABEL_VARIABLES);
00875   n = json_array_get_length (array);
00876   input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00877   for (i = 0; i < n; ++i)
00878     {
00879 #if DEBUG_INPUT
00880       fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00881 #endif
```

```
00882       child = json_array_get_element (array, i);
00883       if (!variable_open_json (input->variable + input->
    nvariables, child,
00884                                input->algorithm, input->nsteps))
00885         goto exit_on_error;
00886       ++input->nvariables;
00887     }
00888   if (!input->nvariables)
00889     {
00890       input_error (gettext ("No optimization variables"));
00891       goto exit_on_error;
00892     }
00893
00894   // Obtaining the error norm
00895   if (json_object_get_member (object, LABEL_NORM))
00896     {
00897       buffer = json_object_get_string_member (object, LABEL_NORM);
00898       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00899         input->norm = ERROR_NORM_EUCLIDIAN;
00900       else if (!strcmp (buffer, LABEL_MAXIMUM))
00901         input->norm = ERROR_NORM_MAXIMUM;
00902       else if (!strcmp (buffer, LABEL_P))
00903         {
00904           input->norm = ERROR_NORM_P;
00905           input->p = json_object_get_float (object,
    LABEL_P, &error_code);
00906           if (!error_code)
00907             {
00908               input_error (gettext ("Bad P parameter"));
00909               goto exit_on_error;
00910             }
00911         }
00912       else if (!strcmp (buffer, LABEL_TAXICAB))
00913         input->norm = ERROR_NORM_TAXICAB;
00914       else
00915         {
00916           input_error (gettext ("Unknown error norm"));
00917           goto exit_on_error;
00918         }
00919     }
00920   else
00921     input->norm = ERROR_NORM_EUCLIDIAN;
00922
00923   // Closing the JSON document
00924   g_object_unref (parser);
00925
00926 #if DEBUG_INPUT
00927   fprintf (stderr, "input_open_json: end\n");
00928 #endif
00929   return 1;
00930
00931 exit_on_error:
00932   g_object_unref (parser);
00933 #if DEBUG_INPUT
00934   fprintf (stderr, "input_open_json: end\n");
00935 #endif
00936   return 0;
00937 }
00938
00946 int
00947 input_open (char *filename)
00948 {
00949   xmlDoc *doc;
00950   JsonParser *parser;
00951
00952 #if DEBUG_INPUT
00953   fprintf (stderr, "input_open: start\n");
00954 #endif
00955
00956   // Resetting input data
00957   input_new ();
00958
00959   // Opening input file
00960 #if DEBUG_INPUT
00961   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00962   fprintf (stderr, "input_open: trying XML format\n");
00963 #endif
00964   doc = xmlParseFile (filename);
00965   if (!doc)
00966     {
00967 #if DEBUG_INPUT
00968       fprintf (stderr, "input_open: trying JSON format\n");
00969 #endif
00970       parser = json_parser_new ();
00971       if (!json_parser_load_from_file (parser, filename, NULL))
00972         {
00973           input_error (gettext ("Unable to parse the input file"));
```

```
00974           goto exit_on_error;
00975         }
00976       if (!input_open_json (parser))
00977         goto exit_on_error;
00978     }
00979   else if (!input_open_xml (doc))
00980     goto exit_on_error;
00981
00982   // Getting the working directory
00983   input->directory = g_path_get_dirname (filename);
00984   input->name = g_path_get_basename (filename);
00985
00986 #if DEBUG_INPUT
00987   fprintf (stderr, "input_open: end\n");
00988 #endif
00989   return 1;
00990
00991 exit_on_error:
00992   show_error (error_message);
00993   g_free (error_message);
00994   input_free ();
00995 #if DEBUG_INPUT
00996   fprintf (stderr, "input_open: end\n");
00997 #endif
00998   return 0;
00999 }
```

5.9 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.

5.9.1 Detailed Description

Header file to define the input functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file input.h.

5.9.2 Enumeration Type Documentation

5.9.2.1 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 };
```

5.9.2.2 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 };
```

5.9.3 Function Documentation

5.9.3.1 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", gettext ("Input"), message);
00128   error_message = g_strdup (buffer);
00129 }
```

5.9.3.2 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 947 of file input.c.

```
00948 {
00949   xmlDoc *doc;
00950   JsonParser *parser;
00951
00952 #if DEBUG_INPUT
00953   fprintf (stderr, "input_open: start\n");
00954 #endif
00955
00956   // Resetting input data
00957   input_new ();
00958
00959   // Opening input file
00960 #if DEBUG_INPUT
00961   fprintf (stderr, "input_open: opening the input file %s\n", filename);
00962   fprintf (stderr, "input_open: trying XML format\n");
00963 #endif
00964   doc = xmlParseFile (filename);
00965   if (!doc)
00966     {
00967 #if DEBUG_INPUT
00968       fprintf (stderr, "input_open: trying JSON format\n");
00969 #endif
00970       parser = json_parser_new ();
00971       if (!json_parser_load_from_file (parser, filename, NULL))
00972         {
00973           input_error (gettext ("Unable to parse the input file"));
00974           goto exit_on_error;
00975         }
00976       if (!input_open_json (parser))
00977         goto exit_on_error;
00978     }
00979   else if (!input_open_xml (doc))
00980     goto exit_on_error;
00981
00982   // Getting the working directory
00983   input->directory = g_path_get_dirname (filename);
00984   input->name = g_path_get_basename (filename);
00985
00986 #if DEBUG_INPUT
00987   fprintf (stderr, "input_open: end\n");
00988 #endif
00989   return 1;
```

```
00990
00991 exit_on_error:
00992   show_error (error_message);
00993   g_free (error_message);
00994   input_free ();
00995 #if DEBUG_INPUT
00996   fprintf (stderr, "input_open: end\n");
00997 #endif
00998   return 0;
00999 }
```

Here is the call graph for this function:

5.9.3.3 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 557 of file input.c.

```
00558 {
00559   JsonNode *node, *child;
00560   JsonObject *object;
00561   JsonArray *array;
00562   const char *buffer;
00563   int error_code;
00564   unsigned int i, n;
00565
00566 #if DEBUG_INPUT
00567   fprintf (stderr, "input_open_json: start\n");
00568 #endif
00569
00570   // Resetting input data
00571   input->type = INPUT_TYPE_JSON;
00572
00573   // Getting the root node
00574 #if DEBUG_INPUT
00575   fprintf (stderr, "input_open_json: getting the root node\n");
00576 #endif
00577   node = json_parser_get_root (parser);
00578   object = json_node_get_object (node);
00579
00580   // Getting result and variables file names
00581   if (!input->result)
00582     {
00583       buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00584       if (!buffer)
00585         buffer = result_name;
00586       input->result = g_strdup (buffer);
00587     }
00588   else
00589     input->result = g_strdup (result_name);
00590   if (!input->variables)
00591     {
00592       buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00593       if (!buffer)
00594         buffer = variables_name;
00595       input->variables = g_strdup (buffer);
00596     }
00597   else
00598     input->variables = g_strdup (variables_name);
00599
00600   // Opening simulator program name
00601   buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00602   if (!buffer)
00603     {
00604       input_error (gettext ("Bad simulator program"));
00605       goto exit_on_error;
00606     }
00607   input->simulator = g_strdup (buffer);
00608
```

```
00609   // Opening evaluator program name
00610   buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00611   if (buffer)
00612     input->evaluator = g_strdup (buffer);
00613
00614   // Obtaining pseudo-random numbers generator seed
00615   input->seed
00616     = json_object_get_uint_with_default (object,
      LABEL_SEED,
00617                                          DEFAULT_RANDOM_SEED, &error_code);
00618   if (error_code)
00619     {
00620       input_error (gettext ("Bad pseudo-random numbers generator seed"));
00621       goto exit_on_error;
00622     }
00623
00624   // Opening algorithm
00625   buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00626   if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627     {
00628       input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630       // Obtaining simulations number
00631       input->nsimulations
00632         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
      );
00633       if (error_code)
00634         {
00635           input_error (gettext ("Bad simulations number"));
00636           goto exit_on_error;
00637         }
00638     }
00639   else if (!strcmp (buffer, LABEL_SWEEP))
00640     input->algorithm = ALGORITHM_SWEEP;
00641   else if (!strcmp (buffer, LABEL_GENETIC))
00642     {
00643       input->algorithm = ALGORITHM_GENETIC;
00644
00645       // Obtaining population
00646       if (json_object_get_member (object, LABEL_NPOPULATION))
00647         {
00648           input->nsimulations
00649             = json_object_get_uint (object,
      LABEL_NPOPULATION, &error_code);
00650           if (error_code || input->nsimulations < 3)
00651             {
00652               input_error (gettext ("Invalid population number"));
00653               goto exit_on_error;
00654             }
00655         }
00656       else
00657         {
00658           input_error (gettext ("No population number"));
00659           goto exit_on_error;
00660         }
00661
00662       // Obtaining generations
00663       if (json_object_get_member (object, LABEL_NGENERATIONS))
00664         {
00665           input->niterations
00666             = json_object_get_uint (object,
      LABEL_NGENERATIONS, &error_code);
00667           if (error_code || !input->niterations)
00668             {
00669               input_error (gettext ("Invalid generations number"));
00670               goto exit_on_error;
00671             }
00672         }
00673       else
00674         {
00675           input_error (gettext ("No generations number"));
00676           goto exit_on_error;
00677         }
00678
00679       // Obtaining mutation probability
00680       if (json_object_get_member (object, LABEL_MUTATION))
00681         {
00682           input->mutation_ratio
00683             = json_object_get_float (object, LABEL_MUTATION, &error_code
      );
00684           if (error_code || input->mutation_ratio < 0.
00685               || input->mutation_ratio >= 1.)
00686             {
00687               input_error (gettext ("Invalid mutation probability"));
00688               goto exit_on_error;
00689             }
00690         }
```

```
00691       else
00692         {
00693           input_error (gettext ("No mutation probability"));
00694           goto exit_on_error;
00695         }
00696
00697       // Obtaining reproduction probability
00698       if (json_object_get_member (object, LABEL_REPRODUCTION))
00699         {
00700           input->reproduction_ratio
00701             = json_object_get_float (object,
      LABEL_REPRODUCTION, &error_code);
00702           if (error_code || input->reproduction_ratio < 0.
00703               || input->reproduction_ratio >= 1.0)
00704             {
00705               input_error (gettext ("Invalid reproduction probability"));
00706               goto exit_on_error;
00707             }
00708         }
00709       else
00710         {
00711           input_error (gettext ("No reproduction probability"));
00712           goto exit_on_error;
00713         }
00714
00715       // Obtaining adaptation probability
00716       if (json_object_get_member (object, LABEL_ADAPTATION))
00717         {
00718           input->adaptation_ratio
00719             = json_object_get_float (object,
      LABEL_ADAPTATION, &error_code);
00720           if (error_code || input->adaptation_ratio < 0.
00721               || input->adaptation_ratio >= 1.)
00722             {
00723               input_error (gettext ("Invalid adaptation probability"));
00724               goto exit_on_error;
00725             }
00726         }
00727       else
00728         {
00729           input_error (gettext ("No adaptation probability"));
00730           goto exit_on_error;
00731         }
00732
00733       // Checking survivals
00734       i = input->mutation_ratio * input->nsimulations;
00735       i += input->reproduction_ratio * input->
      nsimulations;
00736       i += input->adaptation_ratio * input->
      nsimulations;
00737       if (i > input->nsimulations - 2)
00738         {
00739           input_error
00740             (gettext
00741              ("No enough survival entities to reproduce the population"));
00742           goto exit_on_error;
00743         }
00744     }
00745   else
00746     {
00747       input_error (gettext ("Unknown algorithm"));
00748       goto exit_on_error;
00749     }
00750
00751   if (input->algorithm == ALGORITHM_MONTE_CARLO
00752       || input->algorithm == ALGORITHM_SWEEP)
00753     {
00754
00755       // Obtaining iterations number
00756       input->niterations
00757         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
      );
00758       if (error_code == 1)
00759         input->niterations = 1;
00760       else if (error_code)
00761         {
00762           input_error (gettext ("Bad iterations number"));
00763           goto exit_on_error;
00764         }
00765
00766       // Obtaining best number
00767       input->nbest
00768         = json_object_get_uint_with_default (object,
      LABEL_NBEST, 1,
00769                                              &error_code);
00770       if (error_code || !input->nbest)
00771         {
```

```
00772           input_error (gettext ("Invalid best number"));
00773           goto exit_on_error;
00774         }
00775
00776       // Obtaining tolerance
00777       input->tolerance
00778         = json_object_get_float_with_default (object,
      LABEL_TOLERANCE, 0.,
00779                                               &error_code);
00780       if (error_code || input->tolerance < 0.)
00781         {
00782           input_error (gettext ("Invalid tolerance"));
00783           goto exit_on_error;
00784         }
00785
00786       // Getting direction search method parameters
00787       if (json_object_get_member (object, LABEL_NSTEPS))
00788         {
00789           input->nsteps
00790             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00791           if (error_code || !input->nsteps)
00792             {
00793               input_error (gettext ("Invalid steps number"));
00794               goto exit_on_error;
00795             }
00796           buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00797           if (!strcmp (buffer, LABEL_COORDINATES))
00798             input->direction = DIRECTION_METHOD_COORDINATES;
00799           else if (!strcmp (buffer, LABEL_RANDOM))
00800             {
00801               input->direction = DIRECTION_METHOD_RANDOM;
00802               input->nestimates
00803                 = json_object_get_uint (object,
      LABEL_NESTIMATES, &error_code);
00804               if (error_code || !input->nestimates)
00805                 {
00806                   input_error (gettext ("Invalid estimates number"));
00807                   goto exit_on_error;
00808                 }
00809             }
00810           else
00811             {
00812               input_error
00813                 (gettext ("Unknown method to estimate the direction search"));
00814               goto exit_on_error;
00815             }
00816           input->relaxation
00817             = json_object_get_float_with_default (object,
      LABEL_RELAXATION,
00818                                                   DEFAULT_RELAXATION,
00819                                                   &error_code);
00820           if (error_code || input->relaxation < 0. || input->
      relaxation > 2.)
00821             {
00822               input_error (gettext ("Invalid relaxation parameter"));
00823               goto exit_on_error;
00824             }
00825         }
00826       else
00827         input->nsteps = 0;
00828     }
00829   // Obtaining the threshold
00830   input->threshold
00831     = json_object_get_float_with_default (object,
      LABEL_THRESHOLD, 0.,
00832                                           &error_code);
00833   if (error_code)
00834     {
00835       input_error (gettext ("Invalid threshold"));
00836       goto exit_on_error;
00837     }
00838
00839   // Reading the experimental data
00840   array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00841   n = json_array_get_length (array);
00842   input->experiment = (Experiment *) g_malloc (n * sizeof (
      Experiment));
00843   for (i = 0; i < n; ++i)
00844     {
00845 #if DEBUG_INPUT
00846       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00847                input->nexperiments);
00848 #endif
00849       child = json_array_get_element (array, i);
00850       if (!input->nexperiments)
00851         {
00852           if (!experiment_open_json (input->experiment, child, 0))
```

```
00853             goto exit_on_error;
00854         }
00855       else
00856         {
00857           if (!experiment_open_json (input->experiment +
      input->nexperiments,
00858                                      child, input->experiment->
      ninputs))
00859             goto exit_on_error;
00860         }
00861       ++input->nexperiments;
00862 #if DEBUG_INPUT
00863       fprintf (stderr, "input_open_json: nexperiments=%u\n",
00864                input->nexperiments);
00865 #endif
00866     }
00867   if (!input->nexperiments)
00868     {
00869       input_error (gettext ("No optimization experiments"));
00870       goto exit_on_error;
00871     }
00872
00873   // Reading the variables data
00874   array = json_object_get_array_member (object, LABEL_VARIABLES);
00875   n = json_array_get_length (array);
00876   input->variable = (Variable *) g_malloc (n * sizeof (
      Variable));
00877   for (i = 0; i < n; ++i)
00878     {
00879 #if DEBUG_INPUT
00880       fprintf (stderr, "input_open_json: nvariables=%u\n", input->
      nvariables);
00881 #endif
00882       child = json_array_get_element (array, i);
00883       if (!variable_open_json (input->variable +
      input->nvariables, child,
00884                                input->algorithm, input->
      nsteps))
00885         goto exit_on_error;
00886       ++input->nvariables;
00887     }
00888   if (!input->nvariables)
00889     {
00890       input_error (gettext ("No optimization variables"));
00891       goto exit_on_error;
00892     }
00893
00894   // Obtaining the error norm
00895   if (json_object_get_member (object, LABEL_NORM))
00896     {
00897       buffer = json_object_get_string_member (object, LABEL_NORM);
00898       if (!strcmp (buffer, LABEL_EUCLIDIAN))
00899         input->norm = ERROR_NORM_EUCLIDIAN;
00900       else if (!strcmp (buffer, LABEL_MAXIMUM))
00901         input->norm = ERROR_NORM_MAXIMUM;
00902       else if (!strcmp (buffer, LABEL_P))
00903         {
00904           input->norm = ERROR_NORM_P;
00905           input->p = json_object_get_float (object,
      LABEL_P, &error_code);
00906           if (!error_code)
00907             {
00908               input_error (gettext ("Bad P parameter"));
00909               goto exit_on_error;
00910             }
00911         }
00912       else if (!strcmp (buffer, LABEL_TAXICAB))
00913         input->norm = ERROR_NORM_TAXICAB;
00914       else
00915         {
00916           input_error (gettext ("Unknown error norm"));
00917           goto exit_on_error;
00918         }
00919     }
00920   else
00921     input->norm = ERROR_NORM_EUCLIDIAN;
00922
00923   // Closing the JSON document
00924   g_object_unref (parser);
00925
00926 #if DEBUG_INPUT
00927   fprintf (stderr, "input_open_json: end\n");
00928 #endif
00929   return 1;
00930
00931 exit_on_error:
00932   g_object_unref (parser);
```

```
00933 #if DEBUG_INPUT
00934   fprintf (stderr, "input_open_json: end\n");
00935 #endif
00936   return 0;
00937 }
```

Here is the call graph for this function:

5.9.3.4 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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("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 (gettext ("Invalid population number"));
00237               goto exit_on_error;
00238             }
00239         }
00240       else
00241         {
00242           input_error (gettext ("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 (gettext ("Invalid generations number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           input_error (gettext ("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 (gettext ("Invalid mutation probability"));
00274               goto exit_on_error;
00275             }
00276         }
```

```
00277       else
00278         {
00279           input_error (gettext ("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 (gettext ("Invalid reproduction probability"));
00293               goto exit_on_error;
00294             }
00295         }
00296       else
00297         {
00298           input_error (gettext ("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 (gettext ("Invalid adaptation probability"));
00312               goto exit_on_error;
00313             }
00314         }
00315       else
00316         {
00317           input_error (gettext ("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             (gettext
00329              ("No enough survival entities to reproduce the population"));
00330           goto exit_on_error;
00331         }
00332     }
00333   else
00334     {
00335       input_error (gettext ("Unknown algorithm"));
00336       goto exit_on_error;
00337     }
00338   xmlFree (buffer);
00339   buffer = NULL;
00340
00341   if (input->algorithm == ALGORITHM_MONTE_CARLO
00342       || input->algorithm == ALGORITHM_SWEEP)
00343     {
00344
00345       // Obtaining iterations number
00346       input->niterations
00347         = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NITERATIONS,
00348                              &error_code);
00349       if (error_code == 1)
00350         input->niterations = 1;
00351       else if (error_code)
00352         {
00353           input_error (gettext ("Bad iterations number"));
00354           goto exit_on_error;
00355         }
00356
00357       // Obtaining best number
00358       input->nbest
```

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

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

```
00523         {
00524           input_error (gettext ("Unknown error norm"));
00525           goto exit_on_error;
00526         }
00527       xmlFree (buffer);
00528     }
00529   else
00530     input->norm = ERROR_NORM_EUCLIDIAN;
00531
00532   // Closing the XML document
00533   xmlFreeDoc (doc);
00534
00535 #if DEBUG_INPUT
00536   fprintf (stderr, "input_open_xml: end\n");
00537 #endif
00538   return 1;
00539
00540 exit_on_error:
00541   xmlFree (buffer);
00542   xmlFreeDoc (doc);
00543 #if DEBUG_INPUT
00544   fprintf (stderr, "input_open_xml: end\n");
00545 #endif
00546   return 0;
00547 }
```

Here is the call graph for this function:

5.10 input.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INPUT__H
00039 #define INPUT__H 1
00040
00045 enum DirectionMethod
00046 {
00047   DIRECTION_METHOD_COORDINATES = 0,
00048   DIRECTION_METHOD_RANDOM = 1,
00049 };
00050
00055 enum ErrorNorm
00056 {
00057   ERROR_NORM_EUCLIDIAN = 0,
00059   ERROR_NORM_MAXIMUM = 1,
00061   ERROR_NORM_P = 2,
00063   ERROR_NORM_TAXICAB = 3
00065 };
00066
00071 typedef struct
00072 {
00073   Experiment *experiment;
00074   Variable *variable;
```

```
00075   char *result;
00076   char *variables;
00077   char *simulator;
00078   char *evaluator;
00080   char *directory;
00081   char *name;
00082   double tolerance;
00083   double mutation_ratio;
00084   double reproduction_ratio;
00085   double adaptation_ratio;
00086   double relaxation;
00087   double p;
00088   double threshold;
00089   unsigned long int seed;
00091   unsigned int nvariables;
00092   unsigned int nexperiments;
00093   unsigned int nsimulations;
00094   unsigned int algorithm;
00095   unsigned int nsteps;
00097   unsigned int direction;
00098   unsigned int nestimates;
00100   unsigned int niterations;
00101   unsigned int nbest;
00102   unsigned int norm;
00103   unsigned int type;
00104 } Input;
00105
00106 extern Input input[1];
00107 extern const char *result_name;
00108 extern const char *variables_name;
00109
00110 // Public functions
00111 void input_new ();
00112 void input_free ();
00113 void input_error (char *message);
00114 int input_open_xml (xmlDoc * doc);
00115 int input_open_json (JsonParser * parser);
00116 int input_open (char *filename);
00117
00118 #endif
```

5.11 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 <alloca.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 **_GNU_SOURCE**
- #define DEBUG_INTERFACE 1

  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 ()

  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.

5.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file interface.c.

5.11.2 Function Documentation

5.11.2.1 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 580 of file interface.c.

```
00581 {
00582   xmlDoc *doc;
00583   JsonGenerator *generator;
00584
00585 #if DEBUG_INTERFACE
00586   fprintf (stderr, "input_save: start\n");
00587 #endif
00588
00589   // Getting the input file directory
00590   input->name = g_path_get_basename (filename);
00591   input->directory = g_path_get_dirname (filename);
00592
00593   if (input->type == INPUT_TYPE_XML)
00594     {
00595       // Opening the input file
00596       doc = xmlNewDoc ((const xmlChar *) "1.0");
00597       input_save_xml (doc);
00598
00599       // Saving the XML file
00600       xmlSaveFormatFile (filename, doc, 1);
00601
00602       // Freeing memory
00603       xmlFreeDoc (doc);
00604     }
00605   else
00606     {
00607       // Opening the input file
00608       generator = json_generator_new ();
00609       json_generator_set_pretty (generator, TRUE);
00610       input_save_json (generator);
00611
00612       // Saving the JSON file
00613       json_generator_to_file (generator, filename, NULL);
00614
00615       // Freeing memory
00616       g_object_unref (generator);
00617     }
00618
00619 #if DEBUG_INTERFACE
00620   fprintf (stderr, "input_save: end\n");
00621 #endif
00622 }
```

Here is the call graph for this function:

5.11.2.2 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 209 of file interface.c.

```
00210 {
00211   JsonObject *object;
00212 #if DEBUG_INTERFACE
00213   fprintf (stderr, "input_save_direction_json: start\n");
00214 #endif
00215   object = json_node_get_object (node);
00216   if (input->nsteps)
00217     {
00218       json_object_set_uint (object, LABEL_NSTEPS,
      input->nsteps);
00219       if (input->relaxation != DEFAULT_RELAXATION)
00220         json_object_set_float (object, LABEL_RELAXATION,
      input->relaxation);
00221       switch (input->direction)
00222         {
00223         case DIRECTION_METHOD_COORDINATES:
00224           json_object_set_string_member (object, LABEL_DIRECTION,
00225                                          LABEL_COORDINATES);
00226           break;
00227         default:
00228           json_object_set_string_member (object, LABEL_DIRECTION,
      LABEL_RANDOM);
00229           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00230         }
00231     }
00232 #if DEBUG_INTERFACE
00233   fprintf (stderr, "input_save_direction_json: end\n");
00234 #endif
00235 }
```

Here is the call graph for this function:

5.11.2.3 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 173 of file interface.c.

```
00174 {
00175 #if DEBUG_INTERFACE
00176   fprintf (stderr, "input_save_direction_xml: start\n");
00177 #endif
00178   if (input->nsteps)
00179     {
00180       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00181       if (input->relaxation != DEFAULT_RELAXATION)
00182         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00183                             input->relaxation);
00184       switch (input->direction)
00185         {
00186         case DIRECTION_METHOD_COORDINATES:
00187           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00188                       (const xmlChar *) LABEL_COORDINATES);
00189           break;
00190         default:
00191           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00192                       (const xmlChar *) LABEL_RANDOM);
00193           xml_node_set_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00194                              input->nestimates);
00195         }
00196     }
00197 #if DEBUG_INTERFACE
00198   fprintf (stderr, "input_save_direction_xml: end\n");
00199 #endif
00200 }
```

Here is the call graph for this function:

5.11.2.4 void input_save_json (JsonGenerator * *generator*)

Function to save the input file in JSON format.

Parameters

generator	JsonGenerator struct.

Definition at line 414 of file interface.c.

```
00415 {
00416   unsigned int i, j;
00417   char *buffer;
00418   JsonNode *node, *child;
00419   JsonObject *object, *object2;
00420   JsonArray *array;
00421   GFile *file, *file2;
00422
00423 #if DEBUG_INTERFACE
00424   fprintf (stderr, "input_save_json: start\n");
00425 #endif
00426
00427   // Setting root JSON node
00428   node = json_node_alloc ();
00429   object = json_object_new ();
00430   json_node_init_object (node, object);
00431   json_generator_set_root (generator, node);
00432
00433   // Adding properties to the root JSON node
00434   if (strcmp (input->result, result_name))
00435     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00436   if (strcmp (input->variables, variables_name))
00437     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00438                                    input->variables);
00439   file = g_file_new_for_path (input->directory);
00440   file2 = g_file_new_for_path (input->simulator);
00441   buffer = g_file_get_relative_path (file, file2);
00442   g_object_unref (file2);
00443   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00444   g_free (buffer);
00445   if (input->evaluator)
00446     {
00447       file2 = g_file_new_for_path (input->evaluator);
00448       buffer = g_file_get_relative_path (file, file2);
00449       g_object_unref (file2);
00450       if (strlen (buffer))
00451         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00452       g_free (buffer);
00453     }
00454   if (input->seed != DEFAULT_RANDOM_SEED)
00455     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00456
00457   // Setting the algorithm
00458   buffer = (char *) g_slice_alloc (64);
00459   switch (input->algorithm)
00460     {
00461     case ALGORITHM_MONTE_CARLO:
00462       json_object_set_string_member (object, LABEL_ALGORITHM,
00463                                      LABEL_MONTE_CARLO);
00464       snprintf (buffer, 64, "%u", input->nsimulations);
00465       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00466       snprintf (buffer, 64, "%u", input->niterations);
00467       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00468       snprintf (buffer, 64, "%.3lg", input->tolerance);
00469       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00470       snprintf (buffer, 64, "%u", input->nbest);
00471       json_object_set_string_member (object, LABEL_NBEST, buffer);
00472       input_save_direction_json (node);
00473       break;
00474     case ALGORITHM_SWEEP:
00475       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00476       snprintf (buffer, 64, "%u", input->niterations);
00477       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00478       snprintf (buffer, 64, "%.3lg", input->tolerance);
00479       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00480       snprintf (buffer, 64, "%u", input->nbest);
00481       json_object_set_string_member (object, LABEL_NBEST, buffer);
```

```
00482       input_save_direction_json (node);
00483       break;
00484     default:
00485       json_object_set_string_member (object, LABEL_ALGORITHM,
    LABEL_GENETIC);
00486       snprintf (buffer, 64, "%u", input->nsimulations);
00487       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00488       snprintf (buffer, 64, "%u", input->niterations);
00489       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00490       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00491       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00492       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00493       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00494       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00495       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00496       break;
00497     }
00498   g_slice_free1 (64, buffer);
00499   if (input->threshold != 0.)
00500     json_object_set_float (object, LABEL_THRESHOLD,
    input->threshold);
00501
00502   // Setting the experimental data
00503   array = json_array_new ();
00504   for (i = 0; i < input->nexperiments; ++i)
00505     {
00506       child = json_node_alloc ();
00507       object2 = json_object_new ();
00508       json_object_set_string_member (object2, LABEL_NAME,
00509                                      input->experiment[i].name);
00510       if (input->experiment[i].weight != 1.)
00511         json_object_set_float (object2, LABEL_WEIGHT,
00512                                input->experiment[i].weight);
00513       for (j = 0; j < input->experiment->ninputs; ++j)
00514         json_object_set_string_member (object2, template[j],
00515                                        input->experiment[i].
    template[j]);
00516       json_node_set_object (child, object2);
00517       json_array_add_element (array, child);
00518     }
00519   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00520
00521   // Setting the variables data
00522   array = json_array_new ();
00523   for (i = 0; i < input->nvariables; ++i)
00524     {
00525       child = json_node_alloc ();
00526       object2 = json_object_new ();
00527       json_object_set_string_member (object2, LABEL_NAME,
00528                                      input->variable[i].name);
00529       json_object_set_float (object2, LABEL_MINIMUM,
00530                              input->variable[i].rangemin);
00531       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532         json_object_set_float (object2,
    LABEL_ABSOLUTE_MINIMUM,
00533                                input->variable[i].rangeminabs);
00534       json_object_set_float (object2, LABEL_MAXIMUM,
00535                              input->variable[i].rangemax);
00536       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
    LABEL_ABSOLUTE_MAXIMUM,
00538                                input->variable[i].rangemaxabs);
00539       if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00540         json_object_set_uint (object2, LABEL_PRECISION,
00541                               input->variable[i].precision);
00542       if (input->algorithm == ALGORITHM_SWEEP)
00543         json_object_set_uint (object2, LABEL_NSWEEPS,
00544                               input->variable[i].nsweeps);
00545       else if (input->algorithm == ALGORITHM_GENETIC)
00546         json_object_set_uint (object2, LABEL_NBITS,
    input->variable[i].nbits);
00547       if (input->nsteps)
00548         json_object_set_float (object, LABEL_STEP,
    input->variable[i].step);
00549       json_node_set_object (child, object2);
00550       json_array_add_element (array, child);
00551     }
00552   json_object_set_array_member (object, LABEL_VARIABLES, array);
00553
00554   // Saving the error norm
00555   switch (input->norm)
00556     {
00557     case ERROR_NORM_MAXIMUM:
00558       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00559       break;
00560     case ERROR_NORM_P:
```

```
00561       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00562       json_object_set_float (object, LABEL_P, input->
    p);
00563       break;
00564     case ERROR_NORM_TAXICAB:
00565       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00566     }
00567
00568 #if DEBUG_INTERFACE
00569   fprintf (stderr, "input_save_json: end\n");
00570 #endif
00571 }
```

Here is the call graph for this function:

5.11.2.5 void input_save_xml (xmlDoc * *doc*)

Function to save the input file in XML format.

Parameters

doc	xmlDoc struct.

Definition at line 244 of file interface.c.

```
00245 {
00246   unsigned int i, j;
00247   char *buffer;
00248   xmlNode *node, *child;
00249   GFile *file, *file2;
00250
00251 #if DEBUG_INTERFACE
00252   fprintf (stderr, "input_save_xml: start\n");
00253 #endif
00254
00255   // Setting root XML node
00256   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00257   xmlDocSetRootElement (doc, node);
00258
00259   // Adding properties to the root XML node
00260   if (xmlStrcmp
00261       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00262     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00263                 (xmlChar *) input->result);
00264   if (xmlStrcmp
00265       ((const xmlChar *) input->variables, (const xmlChar *)
    variables_name))
00266     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00267                 (xmlChar *) input->variables);
00268   file = g_file_new_for_path (input->directory);
00269   file2 = g_file_new_for_path (input->simulator);
00270   buffer = g_file_get_relative_path (file, file2);
00271   g_object_unref (file2);
00272   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273   g_free (buffer);
00274   if (input->evaluator)
00275     {
00276       file2 = g_file_new_for_path (input->evaluator);
00277       buffer = g_file_get_relative_path (file, file2);
00278       g_object_unref (file2);
00279       if (xmlStrlen ((xmlChar *) buffer))
00280         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00281                     (xmlChar *) buffer);
00282       g_free (buffer);
00283     }
00284   if (input->seed != DEFAULT_RANDOM_SEED)
00285     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
    input->seed);
00286
00287   // Setting the algorithm
00288   buffer = (char *) g_slice_alloc (64);
00289   switch (input->algorithm)
00290     {
00291     case ALGORITHM_MONTE_CARLO:
00292       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                   (const xmlChar *) LABEL_MONTE_CARLO);
00294       snprintf (buffer, 64, "%u", input->nsimulations);
00295       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                   (xmlChar *) buffer);
00297       snprintf (buffer, 64, "%u", input->niterations);
00298       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
```

```
00299                   (xmlChar *) buffer);
00300       snprintf (buffer, 64, "%.3lg", input->tolerance);
00301       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302       snprintf (buffer, 64, "%u", input->nbest);
00303       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304       input_save_direction_xml (node);
00305       break;
00306     case ALGORITHM_SWEEP:
00307       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                   (const xmlChar *) LABEL_SWEEP);
00309       snprintf (buffer, 64, "%u", input->niterations);
00310       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                   (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%.3lg", input->tolerance);
00313       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00314       snprintf (buffer, 64, "%u", input->nbest);
00315       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00316       input_save_direction_xml (node);
00317       break;
00318     default:
00319       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00320                   (const xmlChar *) LABEL_GENETIC);
00321       snprintf (buffer, 64, "%u", input->nsimulations);
00322       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00323                   (xmlChar *) buffer);
00324       snprintf (buffer, 64, "%u", input->niterations);
00325       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00326                   (xmlChar *) buffer);
00327       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00328       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00329       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00330       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00331                   (xmlChar *) buffer);
00332       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00333       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00334       break;
00335     }
00336   g_slice_free1 (64, buffer);
00337   if (input->threshold != 0.)
00338     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00339                         input->threshold);
00340
00341   // Setting the experimental data
00342   for (i = 0; i < input->nexperiments; ++i)
00343     {
00344       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00345       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00346                   (xmlChar *) input->experiment[i].name);
00347       if (input->experiment[i].weight != 1.)
00348         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00349                             input->experiment[i].weight);
00350       for (j = 0; j < input->experiment->ninputs; ++j)
00351         xmlSetProp (child, (const xmlChar *) template[j],
00352                     (xmlChar *) input->experiment[i].template[j]);
00353     }
00354
00355   // Setting the variables data
00356   for (i = 0; i < input->nvariables; ++i)
00357     {
00358       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00359       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00360                   (xmlChar *) input->variable[i].name);
00361       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00362                           input->variable[i].rangemin);
00363       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00364         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00365                             input->variable[i].rangeminabs);
00366       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00367                           input->variable[i].rangemax);
00368       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00369         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00370                             input->variable[i].rangemaxabs);
00371       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00372         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00373                            input->variable[i].precision);
00374       if (input->algorithm == ALGORITHM_SWEEP)
00375         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00376                            input->variable[i].nsweeps);
```

```
00377       else if (input->algorithm == ALGORITHM_GENETIC)
00378         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00379                            input->variable[i].nbits);
00380       if (input->nsteps)
00381         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00382                             input->variable[i].step);
00383     }
00384
00385   // Saving the error norm
00386   switch (input->norm)
00387     {
00388     case ERROR_NORM_MAXIMUM:
00389       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00390                   (const xmlChar *) LABEL_MAXIMUM);
00391       break;
00392     case ERROR_NORM_P:
00393       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00394                   (const xmlChar *) LABEL_P);
00395       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00396       break;
00397     case ERROR_NORM_TAXICAB:
00398       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00399                   (const xmlChar *) LABEL_TAXICAB);
00400     }
00401
00402 #if DEBUG_INTERFACE
00403   fprintf (stderr, "input_save: end\n");
00404 #endif
00405 }
```

Here is the call graph for this function:

5.11.2.6 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 731 of file interface.c.

```
00732 {
00733   unsigned int i;
00734 #if DEBUG_INTERFACE
00735   fprintf (stderr, "window_get_algorithm: start\n");
00736 #endif
00737   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00738 #if DEBUG_INTERFACE
00739   fprintf (stderr, "window_get_algorithm: %u\n", i);
00740   fprintf (stderr, "window_get_algorithm: end\n");
00741 #endif
00742   return i;
00743 }
```

Here is the call graph for this function:

5.11.2.7 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 751 of file interface.c.

```
00752 {
00753   unsigned int i;
00754 #if DEBUG_INTERFACE
00755   fprintf (stderr, "window_get_direction: start\n");
00756 #endif
00757   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00758 #if DEBUG_INTERFACE
00759   fprintf (stderr, "window_get_direction: %u\n", i);
00760   fprintf (stderr, "window_get_direction: end\n");
00761 #endif
00762   return i;
00763 }
```

Here is the call graph for this function:

5.11.2.8 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 771 of file interface.c.

```
00772 {
00773   unsigned int i;
00774 #if DEBUG_INTERFACE
00775   fprintf (stderr, "window_get_norm: start\n");
00776 #endif
00777   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00778 #if DEBUG_INTERFACE
00779   fprintf (stderr, "window_get_norm: %u\n", i);
00780   fprintf (stderr, "window_get_norm: end\n");
00781 #endif
00782   return i;
00783 }
```

Here is the call graph for this function:

5.11.2.9 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 1876 of file interface.c.

```
01877 {
01878   unsigned int i;
01879   char *buffer;
01880 #if DEBUG_INTERFACE
01881   fprintf (stderr, "window_read: start\n");
01882 #endif
01883
01884   // Reading new input file
01885   input_free ();
01886   if (!input_open (filename))
01887     {
01888 #if DEBUG_INTERFACE
01889       fprintf (stderr, "window_read: end\n");
01890 #endif
```

```
01891       return 0;
01892     }
01893
01894   // Setting GTK+ widgets data
01895   gtk_entry_set_text (window->entry_result, input->result);
01896   gtk_entry_set_text (window->entry_variables, input->
      variables);
01897   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01898   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01899                                  (window->button_simulator), buffer);
01900   g_free (buffer);
01901   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01902                                 (size_t) input->evaluator);
01903   if (input->evaluator)
01904     {
01905       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01906       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01907                                      (window->button_evaluator), buffer);
01908       g_free (buffer);
01909     }
01910   gtk_toggle_button_set_active
01911     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01912   switch (input->algorithm)
01913     {
01914     case ALGORITHM_MONTE_CARLO:
01915       gtk_spin_button_set_value (window->spin_simulations,
01916                                  (gdouble) input->nsimulations);
01917     case ALGORITHM_SWEEP:
01918       gtk_spin_button_set_value (window->spin_iterations,
01919                                  (gdouble) input->niterations);
01920       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01921       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01922       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
01923                                     input->nsteps);
01924       if (input->nsteps)
01925         {
01926           gtk_toggle_button_set_active
01927             (GTK_TOGGLE_BUTTON (window->button_direction
01928                                 [input->direction]), TRUE);
01929           gtk_spin_button_set_value (window->spin_steps,
01930                                      (gdouble) input->nsteps);
01931           gtk_spin_button_set_value (window->spin_relaxation,
01932                                      (gdouble) input->relaxation);
01933           switch (input->direction)
01934             {
01935             case DIRECTION_METHOD_RANDOM:
01936               gtk_spin_button_set_value (window->spin_estimates,
01937                                          (gdouble) input->nestimates);
01938             }
01939         }
01940       break;
01941     default:
01942       gtk_spin_button_set_value (window->spin_population,
01943                                  (gdouble) input->nsimulations);
01944       gtk_spin_button_set_value (window->spin_generations,
01945                                  (gdouble) input->niterations);
01946       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01947       gtk_spin_button_set_value (window->spin_reproduction,
01948                                  input->reproduction_ratio);
01949       gtk_spin_button_set_value (window->spin_adaptation,
01950                                  input->adaptation_ratio);
01951     }
01952   gtk_toggle_button_set_active
01953     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01954   gtk_spin_button_set_value (window->spin_p, input->p);
01955   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01956   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01957   g_signal_handler_block (window->button_experiment,
01958                           window->id_experiment_name);
01959   gtk_combo_box_text_remove_all (window->combo_experiment);
01960   for (i = 0; i < input->nexperiments; ++i)
01961     gtk_combo_box_text_append_text (window->combo_experiment,
01962                                     input->experiment[i].name);
01963   g_signal_handler_unblock
01964     (window->button_experiment, window->
      id_experiment_name);
01965   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
```

```
01966   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01968   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01969   gtk_combo_box_text_remove_all (window->combo_variable);
01970   for (i = 0; i < input->nvariables; ++i)
01971     gtk_combo_box_text_append_text (window->combo_variable,
01972                                     input->variable[i].name);
01973   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01974   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01975   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976   window_set_variable ();
01977   window_update ();
01978
01979 #if DEBUG_INTERFACE
01980   fprintf (stderr, "window_read: end\n");
01981 #endif
01982   return 1;
01983 }
```

Here is the call graph for this function:

5.11.2.10 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 824 of file interface.c.

```
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (gettext ("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  gettext ("_Cancel"),
00840                                  GTK_RESPONSE_CANCEL,
00841                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00842   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00843   buffer = g_build_filename (input->directory, input->name, NULL);
00844   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00845   g_free (buffer);
00846
00847   // Adding XML filter
00848   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter1, "XML");
00850   gtk_file_filter_add_pattern (filter1, "*.xml");
00851   gtk_file_filter_add_pattern (filter1, "*.XML");
00852   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00853
00854   // Adding JSON filter
00855   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00856   gtk_file_filter_set_name (filter2, "JSON");
00857   gtk_file_filter_add_pattern (filter2, "*.json");
00858   gtk_file_filter_add_pattern (filter2, "*.JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.js");
00860   gtk_file_filter_add_pattern (filter2, "*.JS");
00861   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863   if (input->type == INPUT_TYPE_XML)
00864     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00865   else
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00867
00868   // If OK response then saving
```

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

```
00949       gtk_widget_destroy (GTK_WIDGET (dlg));
00950 #if DEBUG_INTERFACE
00951       fprintf (stderr, "window_save: end\n");
00952 #endif
00953       return 1;
00954     }
00955
00956   // Closing and freeing memory
00957   gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959   fprintf (stderr, "window_save: end\n");
00960 #endif
00961   return 0;
00962 }
```

Here is the call graph for this function:

5.11.2.11 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 1520 of file interface.c.

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

5.12 interface.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <gsl/gsl_rng.h>
00045 #include <libxml/parser.h>
00046 #include <libintl.h>
00047 #include <glib.h>
00048 #include <glib/gstdio.h>
00049 #include <json-glib/json-glib.h>
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #elif !defined (__BSD_VISIBLE)
00053 #include <alloca.h>
00054 #endif
00055 #if HAVE_MPI
00056 #include <mpi.h>
00057 #endif
00058 #include <gio/gio.h>
00059 #include <gtk/gtk.h>
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 #include "interface.h"
00067
00068 #define DEBUG_INTERFACE 1
00069
00070
00074 #ifdef G_OS_WIN32
00075 #define INPUT_FILE "test-ga-win.xml"
00076 #else
00077 #define INPUT_FILE "test-ga.xml"
00078 #endif
00079
00080 const char *logo[] = {
00081   "32 32 3 1",
00082   "     c None",
00083   ".    c #0000FF",
00084   "+    c #FF0000",
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
00119 /*
```

```
00120 const char * logo[] = {
00121 "32 32 3 1",
00122 "      c #FFFFFFFFFFFF",
00123 ".     c #00000000FFFF",
00124 "X     c #FFFF00000000",
00125 "                                ",
00126 "                                ",
00127 "                                ",
00128 "     .       .       .      .   ",
00129 "     .       .       .      .   ",
00130 "     .       .       .      .   ",
00131 "     .       .       .      .   ",
00132 "     .       .     XXX      .   ",
00133 "     .       .    XXXXX     .   ",
00134 "     .       .    XXXXX     .   ",
00135 "     .       .    XXXXX     .   ",
00136 "    XXX      .     XXX    XXX    ",
00137 "   XXXXX     .       .    XXXXX  ",
00138 "   XXXXX     .       .    XXXXX  ",
00139 "   XXXXX     .       .    XXXXX  ",
00140 "    XXX      .       .     XXX   ",
00141 "     .       .       .       .   ",
00142 "     .      XXX      .       .   ",
00143 "     .     XXXXX     .       .   ",
00144 "     .     XXXXX     .       .   ",
00145 "     .     XXXXX     .       .   ",
00146 "     .      XXX      .       .   ",
00147 "     .       .       .       .   ",
00148 "     .       .       .       .   ",
00149 "     .       .       .       .   ",
00150 "     .       .       .       .   ",
00151 "     .       .       .       .   ",
00152 "     .       .       .       .   ",
00153 "     .       .       .       .   ",
00154 "                                ",
00155 "                                ",
00156 "                                "};
00157 */
00158
00159 Options options[1];
00161 Running running[1];
00163 Window window[1];
00165
00172 void
00173 input_save_direction_xml (xmlNode * node)
00174 {
00175 #if DEBUG_INTERFACE
00176   fprintf (stderr, "input_save_direction_xml: start\n");
00177 #endif
00178   if (input->nsteps)
00179     {
00180       xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
      input->nsteps);
00181       if (input->relaxation != DEFAULT_RELAXATION)
00182         xml_node_set_float (node, (const xmlChar *)
      LABEL_RELAXATION,
00183                             input->relaxation);
00184       switch (input->direction)
00185         {
00186         case DIRECTION_METHOD_COORDINATES:
00187           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00188                       (const xmlChar *) LABEL_COORDINATES);
00189           break;
00190         default:
00191           xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00192                       (const xmlChar *) LABEL_RANDOM);
00193           xml_node_set_uint (node, (const xmlChar *)
      LABEL_NESTIMATES,
00194                              input->nestimates);
00195         }
00196     }
00197 #if DEBUG_INTERFACE
00198   fprintf (stderr, "input_save_direction_xml: end\n");
00199 #endif
00200 }
00201
00208 void
00209 input_save_direction_json (JsonNode * node)
00210 {
00211   JsonObject *object;
00212 #if DEBUG_INTERFACE
00213   fprintf (stderr, "input_save_direction_json: start\n");
00214 #endif
00215   object = json_node_get_object (node);
00216   if (input->nsteps)
00217     {
00218       json_object_set_uint (object, LABEL_NSTEPS,
```

```
      input->nsteps);
00219       if (input->relaxation != DEFAULT_RELAXATION)
00220         json_object_set_float (object, LABEL_RELAXATION,
      input->relaxation);
00221       switch (input->direction)
00222         {
00223         case DIRECTION_METHOD_COORDINATES:
00224           json_object_set_string_member (object, LABEL_DIRECTION,
00225                                          LABEL_COORDINATES);
00226           break;
00227         default:
00228           json_object_set_string_member (object, LABEL_DIRECTION,
      LABEL_RANDOM);
00229           json_object_set_uint (object, LABEL_NESTIMATES,
      input->nestimates);
00230         }
00231     }
00232 #if DEBUG_INTERFACE
00233   fprintf (stderr, "input_save_direction_json: end\n");
00234 #endif
00235 }
00236
00243 void
00244 input_save_xml (xmlDoc * doc)
00245 {
00246   unsigned int i, j;
00247   char *buffer;
00248   xmlNode *node, *child;
00249   GFile *file, *file2;
00250
00251 #if DEBUG_INTERFACE
00252   fprintf (stderr, "input_save_xml: start\n");
00253 #endif
00254
00255   // Setting root XML node
00256   node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00257   xmlDocSetRootElement (doc, node);
00258
00259   // Adding properties to the root XML node
00260   if (xmlStrcmp
00261       ((const xmlChar *) input->result, (const xmlChar *) result_name))
00262     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00263                 (xmlChar *) input->result);
00264   if (xmlStrcmp
00265       ((const xmlChar *) input->variables, (const xmlChar *)
      variables_name))
00266     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00267                 (xmlChar *) input->variables);
00268   file = g_file_new_for_path (input->directory);
00269   file2 = g_file_new_for_path (input->simulator);
00270   buffer = g_file_get_relative_path (file, file2);
00271   g_object_unref (file2);
00272   xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273   g_free (buffer);
00274   if (input->evaluator)
00275     {
00276       file2 = g_file_new_for_path (input->evaluator);
00277       buffer = g_file_get_relative_path (file, file2);
00278       g_object_unref (file2);
00279       if (xmlStrlen ((xmlChar *) buffer))
00280         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00281                     (xmlChar *) buffer);
00282       g_free (buffer);
00283     }
00284   if (input->seed != DEFAULT_RANDOM_SEED)
00285     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
      input->seed);
00286
00287   // Setting the algorithm
00288   buffer = (char *) g_slice_alloc (64);
00289   switch (input->algorithm)
00290     {
00291     case ALGORITHM_MONTE_CARLO:
00292       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                   (const xmlChar *) LABEL_MONTE_CARLO);
00294       snprintf (buffer, 64, "%u", input->nsimulations);
00295       xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                   (xmlChar *) buffer);
00297       snprintf (buffer, 64, "%u", input->niterations);
00298       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                   (xmlChar *) buffer);
00300       snprintf (buffer, 64, "%.3lg", input->tolerance);
00301       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302       snprintf (buffer, 64, "%u", input->nbest);
00303       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304       input_save_direction_xml (node);
00305       break;
```

```
00306     case ALGORITHM_SWEEP:
00307       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                   (const xmlChar *) LABEL_SWEEP);
00309       snprintf (buffer, 64, "%u", input->niterations);
00310       xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                   (xmlChar *) buffer);
00312       snprintf (buffer, 64, "%.3lg", input->tolerance);
00313       xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00314       snprintf (buffer, 64, "%u", input->nbest);
00315       xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00316       input_save_direction_xml (node);
00317       break;
00318     default:
00319       xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00320                   (const xmlChar *) LABEL_GENETIC);
00321       snprintf (buffer, 64, "%u", input->nsimulations);
00322       xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00323                   (xmlChar *) buffer);
00324       snprintf (buffer, 64, "%u", input->niterations);
00325       xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00326                   (xmlChar *) buffer);
00327       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00328       xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00329       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00330       xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00331                   (xmlChar *) buffer);
00332       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00333       xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00334       break;
00335     }
00336   g_slice_free1 (64, buffer);
00337   if (input->threshold != 0.)
00338     xml_node_set_float (node, (const xmlChar *)
      LABEL_THRESHOLD,
00339                         input->threshold);
00340
00341   // Setting the experimental data
00342   for (i = 0; i < input->nexperiments; ++i)
00343     {
00344       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00345       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00346                   (xmlChar *) input->experiment[i].name);
00347       if (input->experiment[i].weight != 1.)
00348         xml_node_set_float (child, (const xmlChar *)
      LABEL_WEIGHT,
00349                             input->experiment[i].weight);
00350       for (j = 0; j < input->experiment->ninputs; ++j)
00351         xmlSetProp (child, (const xmlChar *) template[j],
00352                     (xmlChar *) input->experiment[i].template[j]);
00353     }
00354
00355   // Setting the variables data
00356   for (i = 0; i < input->nvariables; ++i)
00357     {
00358       child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00359       xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00360                   (xmlChar *) input->variable[i].name);
00361       xml_node_set_float (child, (const xmlChar *)
      LABEL_MINIMUM,
00362                           input->variable[i].rangemin);
00363       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00364         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MINIMUM,
00365                             input->variable[i].rangeminabs);
00366       xml_node_set_float (child, (const xmlChar *)
      LABEL_MAXIMUM,
00367                           input->variable[i].rangemax);
00368       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00369         xml_node_set_float (child, (const xmlChar *)
      LABEL_ABSOLUTE_MAXIMUM,
00370                             input->variable[i].rangemaxabs);
00371       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00372         xml_node_set_uint (child, (const xmlChar *)
      LABEL_PRECISION,
00373                            input->variable[i].precision);
00374       if (input->algorithm == ALGORITHM_SWEEP)
00375         xml_node_set_uint (child, (const xmlChar *)
      LABEL_NSWEEPS,
00376                            input->variable[i].nsweeps);
00377       else if (input->algorithm == ALGORITHM_GENETIC)
00378         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00379                            input->variable[i].nbits);
00380       if (input->nsteps)
00381         xml_node_set_float (child, (const xmlChar *)
      LABEL_STEP,
00382                             input->variable[i].step);
```

```
00383     }
00384
00385   // Saving the error norm
00386   switch (input->norm)
00387     {
00388     case ERROR_NORM_MAXIMUM:
00389       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00390                   (const xmlChar *) LABEL_MAXIMUM);
00391       break;
00392     case ERROR_NORM_P:
00393       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00394                   (const xmlChar *) LABEL_P);
00395       xml_node_set_float (node, (const xmlChar *) LABEL_P,
      input->p);
00396       break;
00397     case ERROR_NORM_TAXICAB:
00398       xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00399                   (const xmlChar *) LABEL_TAXICAB);
00400     }
00401
00402 #if DEBUG_INTERFACE
00403   fprintf (stderr, "input_save: end\n");
00404 #endif
00405 }
00406
00413 void
00414 input_save_json (JsonGenerator * generator)
00415 {
00416   unsigned int i, j;
00417   char *buffer;
00418   JsonNode *node, *child;
00419   JsonObject *object, *object2;
00420   JsonArray *array;
00421   GFile *file, *file2;
00422
00423 #if DEBUG_INTERFACE
00424   fprintf (stderr, "input_save_json: start\n");
00425 #endif
00426
00427   // Setting root JSON node
00428   node = json_node_alloc ();
00429   object = json_object_new ();
00430   json_node_init_object (node, object);
00431   json_generator_set_root (generator, node);
00432
00433   // Adding properties to the root JSON node
00434   if (strcmp (input->result, result_name))
00435     json_object_set_string_member (object, LABEL_RESULT_FILE,
      input->result);
00436   if (strcmp (input->variables, variables_name))
00437     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00438                                    input->variables);
00439   file = g_file_new_for_path (input->directory);
00440   file2 = g_file_new_for_path (input->simulator);
00441   buffer = g_file_get_relative_path (file, file2);
00442   g_object_unref (file2);
00443   json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00444   g_free (buffer);
00445   if (input->evaluator)
00446     {
00447       file2 = g_file_new_for_path (input->evaluator);
00448       buffer = g_file_get_relative_path (file, file2);
00449       g_object_unref (file2);
00450       if (strlen (buffer))
00451         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00452       g_free (buffer);
00453     }
00454   if (input->seed != DEFAULT_RANDOM_SEED)
00455     json_object_set_uint (object, LABEL_SEED,
      input->seed);
00456
00457   // Setting the algorithm
00458   buffer = (char *) g_slice_alloc (64);
00459   switch (input->algorithm)
00460     {
00461     case ALGORITHM_MONTE_CARLO:
00462       json_object_set_string_member (object, LABEL_ALGORITHM,
00463                                      LABEL_MONTE_CARLO);
00464       snprintf (buffer, 64, "%u", input->nsimulations);
00465       json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00466       snprintf (buffer, 64, "%u", input->niterations);
00467       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00468       snprintf (buffer, 64, "%.3lg", input->tolerance);
00469       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00470       snprintf (buffer, 64, "%u", input->nbest);
00471       json_object_set_string_member (object, LABEL_NBEST, buffer);
00472       input_save_direction_json (node);
```

```
00473       break;
00474     case ALGORITHM_SWEEP:
00475       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_SWEEP);
00476       snprintf (buffer, 64, "%u", input->niterations);
00477       json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00478       snprintf (buffer, 64, "%.3lg", input->tolerance);
00479       json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00480       snprintf (buffer, 64, "%u", input->nbest);
00481       json_object_set_string_member (object, LABEL_NBEST, buffer);
00482       input_save_direction_json (node);
00483       break;
00484     default:
00485       json_object_set_string_member (object, LABEL_ALGORITHM,
      LABEL_GENETIC);
00486       snprintf (buffer, 64, "%u", input->nsimulations);
00487       json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00488       snprintf (buffer, 64, "%u", input->niterations);
00489       json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00490       snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00491       json_object_set_string_member (object, LABEL_MUTATION, buffer);
00492       snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00493       json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00494       snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00495       json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00496       break;
00497     }
00498   g_slice_free1 (64, buffer);
00499   if (input->threshold != 0.)
00500     json_object_set_float (object, LABEL_THRESHOLD,
      input->threshold);
00501
00502   // Setting the experimental data
00503   array = json_array_new ();
00504   for (i = 0; i < input->nexperiments; ++i)
00505     {
00506       child = json_node_alloc ();
00507       object2 = json_object_new ();
00508       json_object_set_string_member (object2, LABEL_NAME,
00509                                      input->experiment[i].name);
00510       if (input->experiment[i].weight != 1.)
00511         json_object_set_float (object2, LABEL_WEIGHT,
00512                                input->experiment[i].weight);
00513       for (j = 0; j < input->experiment->ninputs; ++j)
00514         json_object_set_string_member (object2, template[j],
00515                                        input->experiment[i].
      template[j]);
00516       json_node_set_object (child, object2);
00517       json_array_add_element (array, child);
00518     }
00519   json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00520
00521   // Setting the variables data
00522   array = json_array_new ();
00523   for (i = 0; i < input->nvariables; ++i)
00524     {
00525       child = json_node_alloc ();
00526       object2 = json_object_new ();
00527       json_object_set_string_member (object2, LABEL_NAME,
00528                                      input->variable[i].name);
00529       json_object_set_float (object2, LABEL_MINIMUM,
00530                              input->variable[i].rangemin);
00531       if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532         json_object_set_float (object2,
      LABEL_ABSOLUTE_MINIMUM,
00533                                input->variable[i].rangeminabs);
00534       json_object_set_float (object2, LABEL_MAXIMUM,
00535                              input->variable[i].rangemax);
00536       if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
      LABEL_ABSOLUTE_MAXIMUM,
00538                                input->variable[i].rangemaxabs);
00539       if (input->variable[i].precision !=
      DEFAULT_PRECISION)
00540         json_object_set_uint (object2, LABEL_PRECISION,
00541                               input->variable[i].precision);
00542       if (input->algorithm == ALGORITHM_SWEEP)
00543         json_object_set_uint (object2, LABEL_NSWEEPS,
00544                               input->variable[i].nsweeps);
00545       else if (input->algorithm == ALGORITHM_GENETIC)
00546         json_object_set_uint (object2, LABEL_NBITS,
      input->variable[i].nbits);
00547       if (input->nsteps)
00548         json_object_set_float (object, LABEL_STEP,
      input->variable[i].step);
00549       json_node_set_object (child, object2);
00550       json_array_add_element (array, child);
```

```
00551     }
00552   json_object_set_array_member (object, LABEL_VARIABLES, array);
00553
00554   // Saving the error norm
00555   switch (input->norm)
00556     {
00557     case ERROR_NORM_MAXIMUM:
00558       json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00559       break;
00560     case ERROR_NORM_P:
00561       json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00562       json_object_set_float (object, LABEL_P, input->
      p);
00563       break;
00564     case ERROR_NORM_TAXICAB:
00565       json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00566     }
00567
00568 #if DEBUG_INTERFACE
00569   fprintf (stderr, "input_save_json: end\n");
00570 #endif
00571 }
00572
00579 void
00580 input_save (char *filename)
00581 {
00582   xmlDoc *doc;
00583   JsonGenerator *generator;
00584
00585 #if DEBUG_INTERFACE
00586   fprintf (stderr, "input_save: start\n");
00587 #endif
00588
00589   // Getting the input file directory
00590   input->name = g_path_get_basename (filename);
00591   input->directory = g_path_get_dirname (filename);
00592
00593   if (input->type == INPUT_TYPE_XML)
00594     {
00595       // Opening the input file
00596       doc = xmlNewDoc ((const xmlChar *) "1.0");
00597       input_save_xml (doc);
00598
00599       // Saving the XML file
00600       xmlSaveFormatFile (filename, doc, 1);
00601
00602       // Freeing memory
00603       xmlFreeDoc (doc);
00604     }
00605   else
00606     {
00607       // Opening the input file
00608       generator = json_generator_new ();
00609       json_generator_set_pretty (generator, TRUE);
00610       input_save_json (generator);
00611
00612       // Saving the JSON file
00613       json_generator_to_file (generator, filename, NULL);
00614
00615       // Freeing memory
00616       g_object_unref (generator);
00617     }
00618
00619 #if DEBUG_INTERFACE
00620   fprintf (stderr, "input_save: end\n");
00621 #endif
00622 }
00623
00628 void
00629 options_new ()
00630 {
00631 #if DEBUG_INTERFACE
00632   fprintf (stderr, "options_new: start\n");
00633 #endif
00634   options->label_seed = (GtkLabel *)
00635     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00636   options->spin_seed = (GtkSpinButton *)
00637     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00638   gtk_widget_set_tooltip_text
00639     (GTK_WIDGET (options->spin_seed),
00640      gettext ("Seed to init the pseudo-random numbers generator"));
00641   gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
      seed);
00642   options->label_threads = (GtkLabel *)
00643     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00644   options->spin_threads
00645     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
```

```
00646   gtk_widget_set_tooltip_text
00647     (GTK_WIDGET (options->spin_threads),
00648      gettext ("Number of threads to perform the calibration/optimization for "
00649               "the stochastic algorithm"));
00650   gtk_spin_button_set_value (options->spin_threads, (gdouble)
      nthreads);
00651   options->label_direction = (GtkLabel *)
00652     gtk_label_new (gettext ("Threads number for the direction search method"));
00653   options->spin_direction
00654     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00655   gtk_widget_set_tooltip_text
00656     (GTK_WIDGET (options->spin_direction),
00657      gettext ("Number of threads to perform the calibration/optimization for "
00658               "the direction search method"));
00659   gtk_spin_button_set_value (options->spin_direction,
00660                              (gdouble) nthreads_direction);
00661   options->grid = (GtkGrid *) gtk_grid_new ();
00662   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00663   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00664   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00665                    0, 1, 1, 1);
00666   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00667                    1, 1, 1, 1);
00668   gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00669                    0, 2, 1, 1);
00670   gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00671                    1, 2, 1, 1);
00672   gtk_widget_show_all (GTK_WIDGET (options->grid));
00673   options->dialog = (GtkDialog *)
00674     gtk_dialog_new_with_buttons (gettext ("Options"),
00675                                  window->window,
00676                                  GTK_DIALOG_MODAL,
00677                                  gettext ("_OK"), GTK_RESPONSE_OK,
00678                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
00679                                  NULL);
00680   gtk_container_add
00681     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00682      GTK_WIDGET (options->grid));
00683   if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00684     {
00685       input->seed
00686         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00687       nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00688       nthreads_direction
00689         = gtk_spin_button_get_value_as_int (options->spin_direction);
00690     }
00691   gtk_widget_destroy (GTK_WIDGET (options->dialog));
00692 #if DEBUG_INTERFACE
00693   fprintf (stderr, "options_new: end\n");
00694 #endif
00695 }
00696
00701 void
00702 running_new ()
00703 {
00704 #if DEBUG_INTERFACE
00705   fprintf (stderr, "running_new: start\n");
00706 #endif
00707   running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00708   running->spinner = (GtkSpinner *) gtk_spinner_new ();
00709   running->grid = (GtkGrid *) gtk_grid_new ();
00710   gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00711   gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00712   running->dialog = (GtkDialog *)
00713     gtk_dialog_new_with_buttons (gettext ("Calculating"),
00714                                  window->window, GTK_DIALOG_MODAL, NULL, NULL);
00715   gtk_container_add
00716     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00717      GTK_WIDGET (running->grid));
00718   gtk_spinner_start (running->spinner);
00719   gtk_widget_show_all (GTK_WIDGET (running->dialog));
00720 #if DEBUG_INTERFACE
00721   fprintf (stderr, "running_new: end\n");
00722 #endif
00723 }
00724
00730 unsigned int
00731 window_get_algorithm ()
00732 {
00733   unsigned int i;
00734 #if DEBUG_INTERFACE
00735   fprintf (stderr, "window_get_algorithm: start\n");
00736 #endif
00737   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00738 #if DEBUG_INTERFACE
00739   fprintf (stderr, "window_get_algorithm: %u\n", i);
```

```
00740   fprintf (stderr, "window_get_algorithm: end\n");
00741 #endif
00742   return i;
00743 }
00744
00750 unsigned int
00751 window_get_direction ()
00752 {
00753   unsigned int i;
00754 #if DEBUG_INTERFACE
00755   fprintf (stderr, "window_get_direction: start\n");
00756 #endif
00757   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00758 #if DEBUG_INTERFACE
00759   fprintf (stderr, "window_get_direction: %u\n", i);
00760   fprintf (stderr, "window_get_direction: end\n");
00761 #endif
00762   return i;
00763 }
00764
00770 unsigned int
00771 window_get_norm ()
00772 {
00773   unsigned int i;
00774 #if DEBUG_INTERFACE
00775   fprintf (stderr, "window_get_norm: start\n");
00776 #endif
00777   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00778 #if DEBUG_INTERFACE
00779   fprintf (stderr, "window_get_norm: %u\n", i);
00780   fprintf (stderr, "window_get_norm: end\n");
00781 #endif
00782   return i;
00783 }
00784
00789 void
00790 window_save_direction ()
00791 {
00792 #if DEBUG_INTERFACE
00793   fprintf (stderr, "window_save_direction: start\n");
00794 #endif
00795   if (gtk_toggle_button_get_active
00796       (GTK_TOGGLE_BUTTON (window->check_direction)))
00797     {
00798       input->nsteps = gtk_spin_button_get_value_as_int (window->
      spin_steps);
00799       input->relaxation = gtk_spin_button_get_value (window->
      spin_relaxation);
00800       switch (window_get_direction ())
00801         {
00802         case DIRECTION_METHOD_COORDINATES:
00803           input->direction = DIRECTION_METHOD_COORDINATES;
00804           break;
00805         default:
00806           input->direction = DIRECTION_METHOD_RANDOM;
00807           input->nestimates
00808             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00809         }
00810     }
00811   else
00812     input->nsteps = 0;
00813 #if DEBUG_INTERFACE
00814   fprintf (stderr, "window_save_direction: end\n");
00815 #endif
00816 }
00817
00823 int
00824 window_save ()
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (gettext ("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  gettext ("_Cancel"),
00840                                  GTK_RESPONSE_CANCEL,
00841                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
```

```
00842   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00843   buffer = g_build_filename (input->directory, input->name, NULL);
00844   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00845   g_free (buffer);
00846
00847   // Adding XML filter
00848   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter1, "XML");
00850   gtk_file_filter_add_pattern (filter1, "*.xml");
00851   gtk_file_filter_add_pattern (filter1, "*.XML");
00852   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00853
00854   // Adding JSON filter
00855   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00856   gtk_file_filter_set_name (filter2, "JSON");
00857   gtk_file_filter_add_pattern (filter2, "*.json");
00858   gtk_file_filter_add_pattern (filter2, "*.JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.js");
00860   gtk_file_filter_add_pattern (filter2, "*.JS");
00861   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863   if (input->type == INPUT_TYPE_XML)
00864     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00865   else
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00867
00868   // If OK response then saving
00869   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00870     {
00871       // Setting input file type
00872       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00873       buffer = (char *) gtk_file_filter_get_name (filter1);
00874       if (!strcmp (buffer, "XML"))
00875         input->type = INPUT_TYPE_XML;
00876       else
00877         input->type = INPUT_TYPE_JSON;
00878
00879       // Adding properties to the root XML node
00880       input->simulator = gtk_file_chooser_get_filename
00881         (GTK_FILE_CHOOSER (window->button_simulator));
00882       if (gtk_toggle_button_get_active
00883           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00884         input->evaluator = gtk_file_chooser_get_filename
00885           (GTK_FILE_CHOOSER (window->button_evaluator));
00886       else
00887         input->evaluator = NULL;
00888       if (input->type == INPUT_TYPE_XML)
00889         {
00890           input->result
00891             = (char *) xmlStrdup ((const xmlChar *)
00892                                   gtk_entry_get_text (window->entry_result));
00893           input->variables
00894             = (char *) xmlStrdup ((const xmlChar *)
00895                                   gtk_entry_get_text (window->entry_variables));
00896         }
00897       else
00898         {
00899           input->result = g_strdup (gtk_entry_get_text (window->
    entry_result));
00900           input->variables
00901             = g_strdup (gtk_entry_get_text (window->entry_variables));
00902         }
00903
00904       // Setting the algorithm
00905       switch (window_get_algorithm ())
00906         {
00907         case ALGORITHM_MONTE_CARLO:
00908           input->algorithm = ALGORITHM_MONTE_CARLO;
00909           input->nsimulations
00910             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00911           input->niterations
00912             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00913           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00914           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00915           window_save_direction ();
00916           break;
00917         case ALGORITHM_SWEEP:
00918           input->algorithm = ALGORITHM_SWEEP;
00919           input->niterations
00920             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00922           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00923           window_save_direction ();
```

```
00924           break;
00925         default:
00926           input->algorithm = ALGORITHM_GENETIC;
00927           input->nsimulations
00928             = gtk_spin_button_get_value_as_int (window->spin_population);
00929           input->niterations
00930             = gtk_spin_button_get_value_as_int (window->spin_generations);
00931           input->mutation_ratio
00932             = gtk_spin_button_get_value (window->spin_mutation);
00933           input->reproduction_ratio
00934             = gtk_spin_button_get_value (window->spin_reproduction);
00935           input->adaptation_ratio
00936             = gtk_spin_button_get_value (window->spin_adaptation);
00937           break;
00938         }
00939       input->norm = window_get_norm ();
00940       input->p = gtk_spin_button_get_value (window->spin_p);
00941       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00942
00943       // Saving the XML file
00944       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00945       input_save (buffer);
00946
00947       // Closing and freeing memory
00948       g_free (buffer);
00949       gtk_widget_destroy (GTK_WIDGET (dlg));
00950 #if DEBUG_INTERFACE
00951       fprintf (stderr, "window_save: end\n");
00952 #endif
00953       return 1;
00954     }
00955
00956   // Closing and freeing memory
00957   gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959   fprintf (stderr, "window_save: end\n");
00960 #endif
00961   return 0;
00962 }
00963
00968 void
00969 window_run ()
00970 {
00971   unsigned int i;
00972   char *msg, *msg2, buffer[64], buffer2[64];
00973 #if DEBUG_INTERFACE
00974   fprintf (stderr, "window_run: start\n");
00975 #endif
00976   if (!window_save ())
00977     {
00978 #if DEBUG_INTERFACE
00979       fprintf (stderr, "window_run: end\n");
00980 #endif
00981       return;
00982     }
00983   running_new ();
00984   while (gtk_events_pending ())
00985     gtk_main_iteration ();
00986   optimize_open ();
00987 #if DEBUG_INTERFACE
00988   fprintf (stderr, "window_run: closing running dialog\n");
00989 #endif
00990   gtk_spinner_stop (running->spinner);
00991   gtk_widget_destroy (GTK_WIDGET (running->dialog));
00992 #if DEBUG_INTERFACE
00993   fprintf (stderr, "window_run: displaying results\n");
00994 #endif
00995   snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00996   msg2 = g_strdup (buffer);
00997   for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00998     {
00999       snprintf (buffer, 64, "%s = %s\n",
01000                 input->variable[i].name, format[input->
      variable[i].precision]);
01001       snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01002       msg = g_strconcat (msg2, buffer2, NULL);
01003       g_free (msg2);
01004     }
01005   snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
01006             optimize->calculation_time);
01007   msg = g_strconcat (msg2, buffer, NULL);
01008   g_free (msg2);
01009   show_message (gettext ("Best result"), msg, INFO_TYPE);
01010   g_free (msg);
01011 #if DEBUG_INTERFACE
01012   fprintf (stderr, "window_run: freeing memory\n");
```

```
01013 #endif
01014   optimize_free ();
01015 #if DEBUG_INTERFACE
01016   fprintf (stderr, "window_run: end\n");
01017 #endif
01018 }
01019
01024 void
01025 window_help ()
01026 {
01027   char *buffer, *buffer2;
01028 #if DEBUG_INTERFACE
01029   fprintf (stderr, "window_help: start\n");
01030 #endif
01031   buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01032                               gettext ("user-manual.pdf"), NULL);
01033   buffer = g_filename_to_uri (buffer2, NULL, NULL);
01034   g_free (buffer2);
01035   gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01036 #if DEBUG_INTERFACE
01037   fprintf (stderr, "window_help: uri=%s\n", buffer);
01038 #endif
01039   g_free (buffer);
01040 #if DEBUG_INTERFACE
01041   fprintf (stderr, "window_help: end\n");
01042 #endif
01043 }
01044
01049 void
01050 window_about ()
01051 {
01052   static const gchar *authors[] = {
01053     "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01054     "Borja Latorre Garcés <borja.latorre@csic.es>",
01055     NULL
01056   };
01057 #if DEBUG_INTERFACE
01058   fprintf (stderr, "window_about: start\n");
01059 #endif
01060   gtk_show_about_dialog
01061     (window->window,
01062      "program_name", "MPCOTool",
01063      "comments",
01064      gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
01065               "A software to perform calibrations or optimizations of "
01066               "empirical parameters"),
01067      "authors", authors,
01068      "translator-credits", "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01069      "version", "3.0.0",
01070      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01071      "logo", window->logo,
01072      "website", "https://github.com/jburguete/mpcotool",
01073      "license-type", GTK_LICENSE_BSD, NULL);
01074 #if DEBUG_INTERFACE
01075   fprintf (stderr, "window_about: end\n");
01076 #endif
01077 }
01078
01084 void
01085 window_update_direction ()
01086 {
01087 #if DEBUG_INTERFACE
01088   fprintf (stderr, "window_update_direction: start\n");
01089 #endif
01090   gtk_widget_show (GTK_WIDGET (window->check_direction));
01091   if (gtk_toggle_button_get_active
01092       (GTK_TOGGLE_BUTTON (window->check_direction)))
01093     {
01094       gtk_widget_show (GTK_WIDGET (window->grid_direction));
01095       gtk_widget_show (GTK_WIDGET (window->label_step));
01096       gtk_widget_show (GTK_WIDGET (window->spin_step));
01097     }
01098   switch (window_get_direction ())
01099     {
01100     case DIRECTION_METHOD_COORDINATES:
01101       gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01102       gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01103       break;
01104     default:
01105       gtk_widget_show (GTK_WIDGET (window->label_estimates));
01106       gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01107     }
01108 #if DEBUG_INTERFACE
01109   fprintf (stderr, "window_update_direction: end\n");
01110 #endif
01111 }
01112
```

```
01117 void
01118 window_update ()
01119 {
01120   unsigned int i;
01121 #if DEBUG_INTERFACE
01122   fprintf (stderr, "window_update: start\n");
01123 #endif
01124   gtk_widget_set_sensitive
01125     (GTK_WIDGET (window->button_evaluator),
01126      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01127                                    (window->check_evaluator)));
01128   gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01129   gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01130   gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01131   gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01132   gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01133   gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01134   gtk_widget_hide (GTK_WIDGET (window->label_bests));
01135   gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01136   gtk_widget_hide (GTK_WIDGET (window->label_population));
01137   gtk_widget_hide (GTK_WIDGET (window->spin_population));
01138   gtk_widget_hide (GTK_WIDGET (window->label_generations));
01139   gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01140   gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01141   gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01142   gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01143   gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01144   gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01145   gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01146   gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01147   gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01148   gtk_widget_hide (GTK_WIDGET (window->label_bits));
01149   gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01150   gtk_widget_hide (GTK_WIDGET (window->check_direction));
01151   gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01152   gtk_widget_hide (GTK_WIDGET (window->label_step));
01153   gtk_widget_hide (GTK_WIDGET (window->spin_step));
01154   gtk_widget_hide (GTK_WIDGET (window->label_p));
01155   gtk_widget_hide (GTK_WIDGET (window->spin_p));
01156   i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01157   switch (window_get_algorithm ())
01158     {
01159     case ALGORITHM_MONTE_CARLO:
01160       gtk_widget_show (GTK_WIDGET (window->label_simulations));
01161       gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01162       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01163       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01164       if (i > 1)
01165         {
01166           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01167           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01168           gtk_widget_show (GTK_WIDGET (window->label_bests));
01169           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01170         }
01171       window_update_direction ();
01172       break;
01173     case ALGORITHM_SWEEP:
01174       gtk_widget_show (GTK_WIDGET (window->label_iterations));
01175       gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01176       if (i > 1)
01177         {
01178           gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01179           gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01180           gtk_widget_show (GTK_WIDGET (window->label_bests));
01181           gtk_widget_show (GTK_WIDGET (window->spin_bests));
01182         }
01183       gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01184       gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01185       gtk_widget_show (GTK_WIDGET (window->check_direction));
01186       window_update_direction ();
01187       break;
01188     default:
01189       gtk_widget_show (GTK_WIDGET (window->label_population));
01190       gtk_widget_show (GTK_WIDGET (window->spin_population));
01191       gtk_widget_show (GTK_WIDGET (window->label_generations));
01192       gtk_widget_show (GTK_WIDGET (window->spin_generations));
01193       gtk_widget_show (GTK_WIDGET (window->label_mutation));
01194       gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01195       gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01196       gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01197       gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01198       gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01199       gtk_widget_show (GTK_WIDGET (window->label_bits));
01200       gtk_widget_show (GTK_WIDGET (window->spin_bits));
01201     }
01202   gtk_widget_set_sensitive
01203     (GTK_WIDGET (window->button_remove_experiment),
```

```
      input->nexperiments > 1);
01204   gtk_widget_set_sensitive
01205     (GTK_WIDGET (window->button_remove_variable), input->
      nvariables > 1);
01206   for (i = 0; i < input->experiment->ninputs; ++i)
01207     {
01208       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01209       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01210       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01211       gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01212       g_signal_handler_block
01213         (window->check_template[i], window->id_template[i]);
01214       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01215       gtk_toggle_button_set_active
01216         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
01217       g_signal_handler_unblock
01218         (window->button_template[i], window->id_input[i]);
01219       g_signal_handler_unblock
01220         (window->check_template[i], window->id_template[i]);
01221     }
01222   if (i > 0)
01223     {
01224       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01225       gtk_widget_set_sensitive
01226         (GTK_WIDGET (window->button_template[i - 1]),
01227          gtk_toggle_button_get_active
01228          (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
01229     }
01230   if (i < MAX_NINPUTS)
01231     {
01232       gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01233       gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01234       gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01235       gtk_widget_set_sensitive
01236         (GTK_WIDGET (window->button_template[i]),
01237          gtk_toggle_button_get_active
01238          (GTK_TOGGLE_BUTTON (window->check_template[i])));
01239       g_signal_handler_block
01240         (window->check_template[i], window->id_template[i]);
01241       g_signal_handler_block (window->button_template[i], window->
      id_input[i]);
01242       gtk_toggle_button_set_active
01243         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
01244       g_signal_handler_unblock
01245         (window->button_template[i], window->id_input[i]);
01246       g_signal_handler_unblock
01247         (window->check_template[i], window->id_template[i]);
01248     }
01249   while (++i < MAX_NINPUTS)
01250     {
01251       gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01252       gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01253     }
01254   gtk_widget_set_sensitive
01255     (GTK_WIDGET (window->spin_minabs),
01256      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01257   gtk_widget_set_sensitive
01258     (GTK_WIDGET (window->spin_maxabs),
01259      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01260   if (window_get_norm () == ERROR_NORM_P)
01261     {
01262       gtk_widget_show (GTK_WIDGET (window->label_p));
01263       gtk_widget_show (GTK_WIDGET (window->spin_p));
01264     }
01265 #if DEBUG_INTERFACE
01266   fprintf (stderr, "window_update: end\n");
01267 #endif
01268 }
01269
01274 void
01275 window_set_algorithm ()
01276 {
01277   int i;
01278 #if DEBUG_INTERFACE
01279   fprintf (stderr, "window_set_algorithm: start\n");
01280 #endif
01281   i = window_get_algorithm ();
01282   switch (i)
01283     {
01284     case ALGORITHM_SWEEP:
01285       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01286       if (i < 0)
01287         i = 0;
01288       gtk_spin_button_set_value (window->spin_sweeps,
01289                                  (gdouble) input->variable[i].
      nsweeps);
```

```
01290       break;
01291     case ALGORITHM_GENETIC:
01292       i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01293       if (i < 0)
01294         i = 0;
01295       gtk_spin_button_set_value (window->spin_bits,
01296                                  (gdouble) input->variable[i].nbits);
01297     }
01298   window_update ();
01299 #if DEBUG_INTERFACE
01300   fprintf (stderr, "window_set_algorithm: end\n");
01301 #endif
01302 }
01303
01308 void
01309 window_set_experiment ()
01310 {
01311   unsigned int i, j;
01312   char *buffer1, *buffer2;
01313 #if DEBUG_INTERFACE
01314   fprintf (stderr, "window_set_experiment: start\n");
01315 #endif
01316   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01317   gtk_spin_button_set_value (window->spin_weight, input->
      experiment[i].weight);
01318   buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01319   buffer2 = g_build_filename (input->directory, buffer1, NULL);
01320   g_free (buffer1);
01321   g_signal_handler_block
01322     (window->button_experiment, window->id_experiment_name);
01323   gtk_file_chooser_set_filename
01324     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01325   g_signal_handler_unblock
01326     (window->button_experiment, window->id_experiment_name);
01327   g_free (buffer2);
01328   for (j = 0; j < input->experiment->ninputs; ++j)
01329     {
01330       g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01331       buffer2 = g_build_filename (input->directory,
01332                                   input->experiment[i].template[j], NULL);
01333       gtk_file_chooser_set_filename
01334         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01335       g_free (buffer2);
01336       g_signal_handler_unblock
01337         (window->button_template[j], window->id_input[j]);
01338     }
01339 #if DEBUG_INTERFACE
01340   fprintf (stderr, "window_set_experiment: end\n");
01341 #endif
01342 }
01343
01348 void
01349 window_remove_experiment ()
01350 {
01351   unsigned int i, j;
01352 #if DEBUG_INTERFACE
01353   fprintf (stderr, "window_remove_experiment: start\n");
01354 #endif
01355   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01356   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01357   gtk_combo_box_text_remove (window->combo_experiment, i);
01358   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01359   experiment_free (input->experiment + i, input->
      type);
01360   --input->nexperiments;
01361   for (j = i; j < input->nexperiments; ++j)
01362     memcpy (input->experiment + j, input->experiment + j + 1,
01363             sizeof (Experiment));
01364   j = input->nexperiments - 1;
01365   if (i > j)
01366     i = j;
01367   for (j = 0; j < input->experiment->ninputs; ++j)
01368     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01369   g_signal_handler_block
01370     (window->button_experiment, window->id_experiment_name);
01371   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01372   g_signal_handler_unblock
01373     (window->button_experiment, window->id_experiment_name);
01374   for (j = 0; j < input->experiment->ninputs; ++j)
01375     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01376   window_update ();
01377 #if DEBUG_INTERFACE
```

```
01378   fprintf (stderr, "window_remove_experiment: end\n");
01379 #endif
01380 }
01381
01386 void
01387 window_add_experiment ()
01388 {
01389   unsigned int i, j;
01390 #if DEBUG_INTERFACE
01391   fprintf (stderr, "window_add_experiment: start\n");
01392 #endif
01393   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01394   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01395   gtk_combo_box_text_insert_text
01396     (window->combo_experiment, i, input->experiment[i].
      name);
01397   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01398   input->experiment = (Experiment *) g_realloc
01399     (input->experiment, (input->nexperiments + 1) * sizeof (
      Experiment));
01400   for (j = input->nexperiments - 1; j > i; --j)
01401     memcpy (input->experiment + j + 1, input->experiment + j,
01402             sizeof (Experiment));
01403   input->experiment[j + 1].weight = input->experiment[j].
      weight;
01404   input->experiment[j + 1].ninputs = input->
      experiment[j].ninputs;
01405   if (input->type == INPUT_TYPE_XML)
01406     {
01407       input->experiment[j + 1].name
01408         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
      name);
01409       for (j = 0; j < input->experiment->ninputs; ++j)
01410         input->experiment[i + 1].template[j]
01411           = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
      template[j]);
01412     }
01413   else
01414     {
01415       input->experiment[j + 1].name = g_strdup (input->
      experiment[j].name);
01416       for (j = 0; j < input->experiment->ninputs; ++j)
01417         input->experiment[i + 1].template[j]
01418           = g_strdup (input->experiment[i].template[j]);
01419     }
01420   ++input->nexperiments;
01421   for (j = 0; j < input->experiment->ninputs; ++j)
01422     g_signal_handler_block (window->button_template[j], window->
      id_input[j]);
01423   g_signal_handler_block
01424     (window->button_experiment, window->id_experiment_name);
01425   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01426   g_signal_handler_unblock
01427     (window->button_experiment, window->id_experiment_name);
01428   for (j = 0; j < input->experiment->ninputs; ++j)
01429     g_signal_handler_unblock (window->button_template[j], window->
      id_input[j]);
01430   window_update ();
01431 #if DEBUG_INTERFACE
01432   fprintf (stderr, "window_add_experiment: end\n");
01433 #endif
01434 }
01435
01440 void
01441 window_name_experiment ()
01442 {
01443   unsigned int i;
01444   char *buffer;
01445   GFile *file1, *file2;
01446 #if DEBUG_INTERFACE
01447   fprintf (stderr, "window_name_experiment: start\n");
01448 #endif
01449   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01450   file1
01451     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01452   file2 = g_file_new_for_path (input->directory);
01453   buffer = g_file_get_relative_path (file2, file1);
01454   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01455   gtk_combo_box_text_remove (window->combo_experiment, i);
01456   gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01457   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01458   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01459   g_free (buffer);
```

```
01460   g_object_unref (file2);
01461   g_object_unref (file1);
01462 #if DEBUG_INTERFACE
01463   fprintf (stderr, "window_name_experiment: end\n");
01464 #endif
01465 }
01466
01471 void
01472 window_weight_experiment ()
01473 {
01474   unsigned int i;
01475 #if DEBUG_INTERFACE
01476   fprintf (stderr, "window_weight_experiment: start\n");
01477 #endif
01478   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01479   input->experiment[i].weight = gtk_spin_button_get_value (window->
      spin_weight);
01480 #if DEBUG_INTERFACE
01481   fprintf (stderr, "window_weight_experiment: end\n");
01482 #endif
01483 }
01484
01490 void
01491 window_inputs_experiment ()
01492 {
01493   unsigned int j;
01494 #if DEBUG_INTERFACE
01495   fprintf (stderr, "window_inputs_experiment: start\n");
01496 #endif
01497   j = input->experiment->ninputs - 1;
01498   if (j
01499       && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01500                                         (window->check_template[j])))
01501     --input->experiment->ninputs;
01502   if (input->experiment->ninputs < MAX_NINPUTS
01503       && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01504                                        (window->check_template[j])))
01505     ++input->experiment->ninputs;
01506   window_update ();
01507 #if DEBUG_INTERFACE
01508   fprintf (stderr, "window_inputs_experiment: end\n");
01509 #endif
01510 }
01511
01519 void
01520 window_template_experiment (void *data)
01521 {
01522   unsigned int i, j;
01523   char *buffer;
01524   GFile *file1, *file2;
01525 #if DEBUG_INTERFACE
01526   fprintf (stderr, "window_template_experiment: start\n");
01527 #endif
01528   i = (size_t) data;
01529   j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01530   file1
01531     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01532   file2 = g_file_new_for_path (input->directory);
01533   buffer = g_file_get_relative_path (file2, file1);
01534   if (input->type == INPUT_TYPE_XML)
01535     input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01536   else
01537     input->experiment[j].template[i] = g_strdup (buffer);
01538   g_free (buffer);
01539   g_object_unref (file2);
01540   g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542   fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }
01545
01550 void
01551 window_set_variable ()
01552 {
01553   unsigned int i;
01554 #if DEBUG_INTERFACE
01555   fprintf (stderr, "window_set_variable: start\n");
01556 #endif
01557   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01558   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01559   gtk_entry_set_text (window->entry_variable, input->variable[i].
      name);
01560   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01561   gtk_spin_button_set_value (window->spin_min, input->variable[i].
      rangemin);
```

```
01562   gtk_spin_button_set_value (window->spin_max, input->variable[i].
      rangemax);
01563   if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01564     {
01565       gtk_spin_button_set_value (window->spin_minabs,
01566                                  input->variable[i].rangeminabs);
01567       gtk_toggle_button_set_active
01568         (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01569     }
01570   else
01571     {
01572       gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01573       gtk_toggle_button_set_active
01574         (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01575     }
01576   if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01577     {
01578       gtk_spin_button_set_value (window->spin_maxabs,
01579                                  input->variable[i].rangemaxabs);
01580       gtk_toggle_button_set_active
01581         (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01582     }
01583   else
01584     {
01585       gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01586       gtk_toggle_button_set_active
01587         (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01588     }
01589   gtk_spin_button_set_value (window->spin_precision,
01590                              input->variable[i].precision);
01591   gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
      nsteps);
01592   if (input->nsteps)
01593     gtk_spin_button_set_value (window->spin_step, input->variable[i].
      step);
01594 #if DEBUG_INTERFACE
01595   fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01596            input->variable[i].precision);
01597 #endif
01598   switch (window_get_algorithm ())
01599     {
01600     case ALGORITHM_SWEEP:
01601       gtk_spin_button_set_value (window->spin_sweeps,
01602                                  (gdouble) input->variable[i].
      nsweeps);
01603 #if DEBUG_INTERFACE
01604       fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605                input->variable[i].nsweeps);
01606 #endif
01607       break;
01608     case ALGORITHM_GENETIC:
01609       gtk_spin_button_set_value (window->spin_bits,
01610                                  (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612       fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613                input->variable[i].nbits);
01614 #endif
01615       break;
01616     }
01617   window_update ();
01618 #if DEBUG_INTERFACE
01619   fprintf (stderr, "window_set_variable: end\n");
01620 #endif
01621 }
01622
01627 void
01628 window_remove_variable ()
01629 {
01630   unsigned int i, j;
01631 #if DEBUG_INTERFACE
01632   fprintf (stderr, "window_remove_variable: start\n");
01633 #endif
01634   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01635   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01636   gtk_combo_box_text_remove (window->combo_variable, i);
01637   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01638   xmlFree (input->variable[i].name);
01639   --input->nvariables;
01640   for (j = i; j < input->nvariables; ++j)
01641     memcpy (input->variable + j, input->variable + j + 1, sizeof (
      Variable));
01642   j = input->nvariables - 1;
01643   if (i > j)
01644     i = j;
01645   g_signal_handler_block (window->entry_variable, window->
```

```
      id_variable_label);
01646   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01647   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01648   window_update ();
01649 #if DEBUG_INTERFACE
01650   fprintf (stderr, "window_remove_variable: end\n");
01651 #endif
01652 }
01653
01658 void
01659 window_add_variable ()
01660 {
01661   unsigned int i, j;
01662 #if DEBUG_INTERFACE
01663   fprintf (stderr, "window_add_variable: start\n");
01664 #endif
01665   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01666   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01667   gtk_combo_box_text_insert_text (window->combo_variable, i,
01668                                   input->variable[i].name);
01669   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01670   input->variable = (Variable *) g_realloc
01671     (input->variable, (input->nvariables + 1) * sizeof (
      Variable));
01672   for (j = input->nvariables - 1; j > i; --j)
01673     memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01674   memcpy (input->variable + j + 1, input->variable + j, sizeof (
      Variable));
01675   if (input->type == INPUT_TYPE_XML)
01676     input->variable[j + 1].name
01677       = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01678   else
01679     input->variable[j + 1].name = g_strdup (input->
      variable[j].name);
01680   ++input->nvariables;
01681   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01682   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01683   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01684   window_update ();
01685 #if DEBUG_INTERFACE
01686   fprintf (stderr, "window_add_variable: end\n");
01687 #endif
01688 }
01689
01694 void
01695 window_label_variable ()
01696 {
01697   unsigned int i;
01698   const char *buffer;
01699 #if DEBUG_INTERFACE
01700   fprintf (stderr, "window_label_variable: start\n");
01701 #endif
01702   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01703   buffer = gtk_entry_get_text (window->entry_variable);
01704   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01705   gtk_combo_box_text_remove (window->combo_variable, i);
01706   gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01707   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01708   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01709 #if DEBUG_INTERFACE
01710   fprintf (stderr, "window_label_variable: end\n");
01711 #endif
01712 }
01713
01718 void
01719 window_precision_variable ()
01720 {
01721   unsigned int i;
01722 #if DEBUG_INTERFACE
01723   fprintf (stderr, "window_precision_variable: start\n");
01724 #endif
01725   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01726   input->variable[i].precision
01727     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01728   gtk_spin_button_set_digits (window->spin_min, input->variable[i].
      precision);
01729   gtk_spin_button_set_digits (window->spin_max, input->variable[i].
      precision);
01730   gtk_spin_button_set_digits (window->spin_minabs,
```

```
01731                               input->variable[i].precision);
01732   gtk_spin_button_set_digits (window->spin_maxabs,
01733                               input->variable[i].precision);
01734 #if DEBUG_INTERFACE
01735   fprintf (stderr, "window_precision_variable: end\n");
01736 #endif
01737 }
01738
01743 void
01744 window_rangemin_variable ()
01745 {
01746   unsigned int i;
01747 #if DEBUG_INTERFACE
01748   fprintf (stderr, "window_rangemin_variable: start\n");
01749 #endif
01750   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01751   input->variable[i].rangemin = gtk_spin_button_get_value (window->
      spin_min);
01752 #if DEBUG_INTERFACE
01753   fprintf (stderr, "window_rangemin_variable: end\n");
01754 #endif
01755 }
01756
01761 void
01762 window_rangemax_variable ()
01763 {
01764   unsigned int i;
01765 #if DEBUG_INTERFACE
01766   fprintf (stderr, "window_rangemax_variable: start\n");
01767 #endif
01768   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01769   input->variable[i].rangemax = gtk_spin_button_get_value (window->
      spin_max);
01770 #if DEBUG_INTERFACE
01771   fprintf (stderr, "window_rangemax_variable: end\n");
01772 #endif
01773 }
01774
01779 void
01780 window_rangeminabs_variable ()
01781 {
01782   unsigned int i;
01783 #if DEBUG_INTERFACE
01784   fprintf (stderr, "window_rangeminabs_variable: start\n");
01785 #endif
01786   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01787   input->variable[i].rangeminabs
01788     = gtk_spin_button_get_value (window->spin_minabs);
01789 #if DEBUG_INTERFACE
01790   fprintf (stderr, "window_rangeminabs_variable: end\n");
01791 #endif
01792 }
01793
01798 void
01799 window_rangemaxabs_variable ()
01800 {
01801   unsigned int i;
01802 #if DEBUG_INTERFACE
01803   fprintf (stderr, "window_rangemaxabs_variable: start\n");
01804 #endif
01805   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01806   input->variable[i].rangemaxabs
01807     = gtk_spin_button_get_value (window->spin_maxabs);
01808 #if DEBUG_INTERFACE
01809   fprintf (stderr, "window_rangemaxabs_variable: end\n");
01810 #endif
01811 }
01812
01817 void
01818 window_step_variable ()
01819 {
01820   unsigned int i;
01821 #if DEBUG_INTERFACE
01822   fprintf (stderr, "window_step_variable: start\n");
01823 #endif
01824   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01825   input->variable[i].step = gtk_spin_button_get_value (window->
      spin_step);
01826 #if DEBUG_INTERFACE
01827   fprintf (stderr, "window_step_variable: end\n");
01828 #endif
01829 }
01830
01835 void
01836 window_update_variable ()
01837 {
01838   int i;
```

```
01839 #if DEBUG_INTERFACE
01840   fprintf (stderr, "window_update_variable: start\n");
01841 #endif
01842   i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01843   if (i < 0)
01844     i = 0;
01845   switch (window_get_algorithm ())
01846     {
01847     case ALGORITHM_SWEEP:
01848       input->variable[i].nsweeps
01849         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01850 #if DEBUG_INTERFACE
01851       fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01852                input->variable[i].nsweeps);
01853 #endif
01854       break;
01855     case ALGORITHM_GENETIC:
01856       input->variable[i].nbits
01857         = gtk_spin_button_get_value_as_int (window->spin_bits);
01858 #if DEBUG_INTERFACE
01859       fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01860                input->variable[i].nbits);
01861 #endif
01862     }
01863 #if DEBUG_INTERFACE
01864   fprintf (stderr, "window_update_variable: end\n");
01865 #endif
01866 }
01867
01875 int
01876 window_read (char *filename)
01877 {
01878   unsigned int i;
01879   char *buffer;
01880 #if DEBUG_INTERFACE
01881   fprintf (stderr, "window_read: start\n");
01882 #endif
01883
01884   // Reading new input file
01885   input_free ();
01886   if (!input_open (filename))
01887     {
01888 #if DEBUG_INTERFACE
01889       fprintf (stderr, "window_read: end\n");
01890 #endif
01891       return 0;
01892     }
01893
01894   // Setting GTK+ widgets data
01895   gtk_entry_set_text (window->entry_result, input->result);
01896   gtk_entry_set_text (window->entry_variables, input->
      variables);
01897   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01898   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01899                                  (window->button_simulator), buffer);
01900   g_free (buffer);
01901   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01902                                 (size_t) input->evaluator);
01903   if (input->evaluator)
01904     {
01905       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01906       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01907                                      (window->button_evaluator), buffer);
01908       g_free (buffer);
01909     }
01910   gtk_toggle_button_set_active
01911     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01912   switch (input->algorithm)
01913     {
01914     case ALGORITHM_MONTE_CARLO:
01915       gtk_spin_button_set_value (window->spin_simulations,
01916                                  (gdouble) input->nsimulations);
01917     case ALGORITHM_SWEEP:
01918       gtk_spin_button_set_value (window->spin_iterations,
01919                                  (gdouble) input->niterations);
01920       gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
      nbest);
01921       gtk_spin_button_set_value (window->spin_tolerance, input->
      tolerance);
01922       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
01923                                     input->nsteps);
01924       if (input->nsteps)
01925         {
01926           gtk_toggle_button_set_active
```

```
01927             (GTK_TOGGLE_BUTTON (window->button_direction
01928                                 [input->direction]), TRUE);
01929           gtk_spin_button_set_value (window->spin_steps,
01930                                      (gdouble) input->nsteps);
01931           gtk_spin_button_set_value (window->spin_relaxation,
01932                                      (gdouble) input->relaxation);
01933           switch (input->direction)
01934             {
01935             case DIRECTION_METHOD_RANDOM:
01936               gtk_spin_button_set_value (window->spin_estimates,
01937                                          (gdouble) input->nestimates);
01938             }
01939         }
01940       break;
01941     default:
01942       gtk_spin_button_set_value (window->spin_population,
01943                                  (gdouble) input->nsimulations);
01944       gtk_spin_button_set_value (window->spin_generations,
01945                                  (gdouble) input->niterations);
01946       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01947       gtk_spin_button_set_value (window->spin_reproduction,
01948                                  input->reproduction_ratio);
01949       gtk_spin_button_set_value (window->spin_adaptation,
01950                                  input->adaptation_ratio);
01951     }
01952   gtk_toggle_button_set_active
01953     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01954   gtk_spin_button_set_value (window->spin_p, input->p);
01955   gtk_spin_button_set_value (window->spin_threshold, input->
      threshold);
01956   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01957   g_signal_handler_block (window->button_experiment,
01958                           window->id_experiment_name);
01959   gtk_combo_box_text_remove_all (window->combo_experiment);
01960   for (i = 0; i < input->nexperiments; ++i)
01961     gtk_combo_box_text_append_text (window->combo_experiment,
01962                                     input->experiment[i].name);
01963   g_signal_handler_unblock
01964     (window->button_experiment, window->id_experiment_name);
01965   g_signal_handler_unblock (window->combo_experiment, window->
      id_experiment);
01966   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01968   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01969   gtk_combo_box_text_remove_all (window->combo_variable);
01970   for (i = 0; i < input->nvariables; ++i)
01971     gtk_combo_box_text_append_text (window->combo_variable,
01972                                     input->variable[i].name);
01973   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01974   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01975   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976   window_set_variable ();
01977   window_update ();
01978
01979 #if DEBUG_INTERFACE
01980   fprintf (stderr, "window_read: end\n");
01981 #endif
01982   return 1;
01983 }
01984
01989 void
01990 window_open ()
01991 {
01992   GtkFileChooserDialog *dlg;
01993   GtkFileFilter *filter;
01994   char *buffer, *directory, *name;
01995
01996 #if DEBUG_INTERFACE
01997   fprintf (stderr, "window_open: start\n");
01998 #endif
01999
02000   // Saving a backup of the current input file
02001   directory = g_strdup (input->directory);
02002   name = g_strdup (input->name);
02003
02004   // Opening dialog
02005   dlg = (GtkFileChooserDialog *)
02006     gtk_file_chooser_dialog_new (gettext ("Open input file"),
02007                                  window->window,
02008                                  GTK_FILE_CHOOSER_ACTION_OPEN,
02009                                  gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
```

```
02010                                 gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02011
02012   // Adding XML filter
02013   filter = (GtkFileFilter *) gtk_file_filter_new ();
02014   gtk_file_filter_set_name (filter, "XML");
02015   gtk_file_filter_add_pattern (filter, "*.xml");
02016   gtk_file_filter_add_pattern (filter, "*.XML");
02017   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019   // Adding JSON filter
02020   filter = (GtkFileFilter *) gtk_file_filter_new ();
02021   gtk_file_filter_set_name (filter, "JSON");
02022   gtk_file_filter_add_pattern (filter, "*.json");
02023   gtk_file_filter_add_pattern (filter, "*.JSON");
02024   gtk_file_filter_add_pattern (filter, "*.js");
02025   gtk_file_filter_add_pattern (filter, "*.JS");
02026   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02027
02028   // If OK saving
02029   while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02030     {
02031
02032       // Traying to open the input file
02033       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02034       if (!window_read (buffer))
02035         {
02036 #if DEBUG_INTERFACE
02037           fprintf (stderr, "window_open: error reading input file\n");
02038 #endif
02039           g_free (buffer);
02040
02041           // Reading backup file on error
02042           buffer = g_build_filename (directory, name, NULL);
02043           if (!input_open (buffer))
02044             {
02045
02046               // Closing on backup file reading error
02047 #if DEBUG_INTERFACE
02048               fprintf (stderr, "window_read: error reading backup file\n");
02049 #endif
02050               g_free (buffer);
02051               break;
02052             }
02053           g_free (buffer);
02054         }
02055       else
02056         {
02057           g_free (buffer);
02058           break;
02059         }
02060     }
02061
02062   // Freeing and closing
02063   g_free (name);
02064   g_free (directory);
02065   gtk_widget_destroy (GTK_WIDGET (dlg));
02066 #if DEBUG_INTERFACE
02067   fprintf (stderr, "window_open: end\n");
02068 #endif
02069 }
02070
02075 void
02076 window_new ()
02077 {
02078   unsigned int i;
02079   char *buffer, *buffer2, buffer3[64];
02080   char *label_algorithm[NALGORITHMS] = {
02081     "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
02082   };
02083   char *tip_algorithm[NALGORITHMS] = {
02084     gettext ("Monte-Carlo brute force algorithm"),
02085     gettext ("Sweep brute force algorithm"),
02086     gettext ("Genetic algorithm")
02087   };
02088   char *label_direction[NDIRECTIONS] = {
02089     gettext ("_Coordinates descent"), gettext ("_Random")
02090   };
02091   char *tip_direction[NDIRECTIONS] = {
02092     gettext ("Coordinates direction estimate method"),
02093     gettext ("Random direction estimate method")
02094   };
02095   char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02096   char *tip_norm[NNORMS] = {
02097     gettext ("Euclidean error norm (L2)"),
02098     gettext ("Maximum error norm (L)"),
02099     gettext ("P error norm (Lp)"),
02100     gettext ("Taxicab error norm (L1)")
```

```
02101   };
02102
02103 #if DEBUG_INTERFACE
02104   fprintf (stderr, "window_new: start\n");
02105 #endif
02106
02107   // Creating the window
02108   window->window = main_window
02109     = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
02110
02111   // Finish when closing the window
02112   g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
02113
02114   // Setting the window title
02115   gtk_window_set_title (window->window, "MPCOTool");
02116
02117   // Creating the open button
02118   window->button_open = (GtkToolButton *) gtk_tool_button_new
02119     (gtk_image_new_from_icon_name ("document-open",
02120                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02121      gettext ("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),
02128      gettext ("Save"));
02129   g_signal_connect (window->button_save, "clicked", (void (*))
    window_save,
02130                     NULL);
02131
02132   // Creating the run button
02133   window->button_run = (GtkToolButton *) gtk_tool_button_new
02134     (gtk_image_new_from_icon_name ("system-run",
02135                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02136      gettext ("Run"));
02137   g_signal_connect (window->button_run, "clicked", window_run, NULL);
02138
02139   // Creating the options button
02140   window->button_options = (GtkToolButton *) gtk_tool_button_new
02141     (gtk_image_new_from_icon_name ("preferences-system",
02142                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02143      gettext ("Options"));
02144   g_signal_connect (window->button_options, "clicked", options_new, NULL);
02145
02146   // Creating the help button
02147   window->button_help = (GtkToolButton *) gtk_tool_button_new
02148     (gtk_image_new_from_icon_name ("help-browser",
02149                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02150      gettext ("Help"));
02151   g_signal_connect (window->button_help, "clicked", window_help, NULL);
02152
02153   // Creating the about button
02154   window->button_about = (GtkToolButton *) gtk_tool_button_new
02155     (gtk_image_new_from_icon_name ("help-about",
02156                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02157      gettext ("About"));
02158   g_signal_connect (window->button_about, "clicked", window_about, NULL);
02159
02160   // Creating the exit button
02161   window->button_exit = (GtkToolButton *) gtk_tool_button_new
02162     (gtk_image_new_from_icon_name ("application-exit",
02163                                    GTK_ICON_SIZE_LARGE_TOOLBAR),
02164      gettext ("Exit"));
02165   g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
02166
02167   // Creating the buttons bar
02168   window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02169   gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02171   gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02173   gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02175   gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02177   gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02179   gtk_toolbar_insert
02180     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02181   gtk_toolbar_insert
02182     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02183   gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02184
02185   // Creating the simulator program label and entry
02186   window->label_simulator
```

```
02187     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
02188   window->button_simulator = (GtkFileChooserButton *)
02189     gtk_file_chooser_button_new (gettext ("Simulator program"),
02190                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02191   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02192                                gettext ("Simulator program executable file"));
02193   gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02194
02195   // Creating the evaluator program label and entry
02196   window->check_evaluator = (GtkCheckButton *)
02197     gtk_check_button_new_with_mnemonic (gettext ("_Evaluator program"));
02198   g_signal_connect (window->check_evaluator, "toggled",
      window_update, NULL);
02199   window->button_evaluator = (GtkFileChooserButton *)
02200     gtk_file_chooser_button_new (gettext ("Evaluator program"),
02201                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02202   gtk_widget_set_tooltip_text
02203     (GTK_WIDGET (window->button_evaluator),
02204      gettext ("Optional evaluator program executable file"));
02205
02206   // Creating the results files labels and entries
02207   window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
02208   window->entry_result = (GtkEntry *) gtk_entry_new ();
02209   gtk_widget_set_tooltip_text
02210     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
02211   window->label_variables
02212     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
02213   window->entry_variables = (GtkEntry *) gtk_entry_new ();
02214   gtk_widget_set_tooltip_text
02215     (GTK_WIDGET (window->entry_variables),
02216      gettext ("All simulated results file"));
02217
02218   // Creating the files grid and attaching widgets
02219   window->grid_files = (GtkGrid *) gtk_grid_new ();
02220   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_simulator),
02221                    0, 0, 1, 1);
02222   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_simulator),
02223                    1, 0, 1, 1);
02224   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      check_evaluator),
02225                    0, 1, 1, 1);
02226   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      button_evaluator),
02227                    1, 1, 1, 1);
02228   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_result),
02229                    0, 2, 1, 1);
02230   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_result),
02231                    1, 2, 1, 1);
02232   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      label_variables),
02233                    0, 3, 1, 1);
02234   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
      entry_variables),
02235                    1, 3, 1, 1);
02236
02237   // Creating the algorithm properties
02238   window->label_simulations = (GtkLabel *) gtk_label_new
02239     (gettext ("Simulations number"));
02240   window->spin_simulations
02241     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02242   gtk_widget_set_tooltip_text
02243     (GTK_WIDGET (window->spin_simulations),
02244      gettext ("Number of simulations to perform for each iteration"));
02245   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02246   window->label_iterations = (GtkLabel *)
02247     gtk_label_new (gettext ("Iterations number"));
02248   window->spin_iterations
02249     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02250   gtk_widget_set_tooltip_text
02251     (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
02252   g_signal_connect
02253     (window->spin_iterations, "value-changed", window_update, NULL);
02254   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02255   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
02256   window->spin_tolerance
02257     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02258   gtk_widget_set_tooltip_text
02259     (GTK_WIDGET (window->spin_tolerance),
02260      gettext ("Tolerance to set the variable interval on the next iteration"));
02261   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
02262   window->spin_bests
02263     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02264   gtk_widget_set_tooltip_text
```

```
02265     (GTK_WIDGET (window->spin_bests),
02266      gettext ("Number of best simulations used to set the variable interval "
02267               "on the next iteration"));
02268   window->label_population
02269     = (GtkLabel *) gtk_label_new (gettext ("Population number"));
02270   window->spin_population
02271     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02272   gtk_widget_set_tooltip_text
02273     (GTK_WIDGET (window->spin_population),
02274      gettext ("Number of population for the genetic algorithm"));
02275   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02276   window->label_generations
02277     = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
02278   window->spin_generations
02279     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02280   gtk_widget_set_tooltip_text
02281     (GTK_WIDGET (window->spin_generations),
02282      gettext ("Number of generations for the genetic algorithm"));
02283   window->label_mutation
02284     = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
02285   window->spin_mutation
02286     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02287   gtk_widget_set_tooltip_text
02288     (GTK_WIDGET (window->spin_mutation),
02289      gettext ("Ratio of mutation for the genetic algorithm"));
02290   window->label_reproduction
02291     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
02292   window->spin_reproduction
02293     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02294   gtk_widget_set_tooltip_text
02295     (GTK_WIDGET (window->spin_reproduction),
02296      gettext ("Ratio of reproduction for the genetic algorithm"));
02297   window->label_adaptation
02298     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
02299   window->spin_adaptation
02300     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02301   gtk_widget_set_tooltip_text
02302     (GTK_WIDGET (window->spin_adaptation),
02303      gettext ("Ratio of adaptation for the genetic algorithm"));
02304   window->label_threshold = (GtkLabel *) gtk_label_new (gettext ("Threshold"));
02305   window->spin_threshold = (GtkSpinButton *) gtk_spin_button_new_with_range
02306     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02307   gtk_widget_set_tooltip_text
02308     (GTK_WIDGET (window->spin_threshold),
02309      gettext ("Threshold in the objective function to finish the simulations"));
02310   window->scrolled_threshold
02311     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02312   gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02313                      GTK_WIDGET (window->spin_threshold));
02314 //  gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02315 //  gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02316 //                         GTK_ALIGN_FILL);
02317
02318   // Creating the direction search method properties
02319   window->check_direction = (GtkCheckButton *)
02320     gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
02321   g_signal_connect (window->check_direction, "clicked",
      window_update, NULL);
02322   window->grid_direction = (GtkGrid *) gtk_grid_new ();
02323   window->button_direction[0] = (GtkRadioButton *)
02324     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02325   gtk_grid_attach (window->grid_direction,
02326                    GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02327   g_signal_connect (window->button_direction[0], "clicked",
      window_update,
02328                     NULL);
02329   for (i = 0; ++i < NDIRECTIONS;)
02330     {
02331       window->button_direction[i] = (GtkRadioButton *)
02332         gtk_radio_button_new_with_mnemonic
02333         (gtk_radio_button_get_group (window->button_direction[0]),
02334          label_direction[i]);
02335       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02336                                    tip_direction[i]);
02337       gtk_grid_attach (window->grid_direction,
02338                        GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02339       g_signal_connect (window->button_direction[i], "clicked",
02340                         window_update, NULL);
02341     }
02342   window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
02343   window->spin_steps = (GtkSpinButton *)
02344     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02345   gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02346   window->label_estimates
02347     = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
02348   window->spin_estimates = (GtkSpinButton *)
02349     gtk_spin_button_new_with_range (1., 1.e3, 1.);
```

```
02350   window->label_relaxation
02351     = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
02352   window->spin_relaxation = (GtkSpinButton *)
02353     gtk_spin_button_new_with_range (0., 2., 0.001);
02354   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_steps),
02355                    0, NDIRECTIONS, 1, 1);
02356   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_steps),
02357                    1, NDIRECTIONS, 1, 1);
02358   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      label_estimates),
02359                    0, NDIRECTIONS + 1, 1, 1);
02360   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_estimates),
02361                    1, NDIRECTIONS + 1, 1, 1);
02362   gtk_grid_attach (window->grid_direction,
02363                    GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
02364                    1);
02365   gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
      spin_relaxation),
02366                    1, NDIRECTIONS + 2, 1, 1);
02367
02368   // Creating the array of algorithms
02369   window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02370   window->button_algorithm[0] = (GtkRadioButton *)
02371     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02372   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02373                                tip_algorithm[0]);
02374   gtk_grid_attach (window->grid_algorithm,
02375                    GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02376   g_signal_connect (window->button_algorithm[0], "clicked",
02377                     window_set_algorithm, NULL);
02378   for (i = 0; ++i < NALGORITHMS;)
02379     {
02380       window->button_algorithm[i] = (GtkRadioButton *)
02381         gtk_radio_button_new_with_mnemonic
02382         (gtk_radio_button_get_group (window->button_algorithm[0]),
02383          label_algorithm[i]);
02384       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02385                                    tip_algorithm[i]);
02386       gtk_grid_attach (window->grid_algorithm,
02387                        GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02388       g_signal_connect (window->button_algorithm[i], "clicked",
02389                         window_set_algorithm, NULL);
02390     }
02391   gtk_grid_attach (window->grid_algorithm,
02392                    GTK_WIDGET (window->label_simulations), 0,
02393                    NALGORITHMS, 1, 1);
02394   gtk_grid_attach (window->grid_algorithm,
02395                    GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02396   gtk_grid_attach (window->grid_algorithm,
02397                    GTK_WIDGET (window->label_iterations), 0,
02398                    NALGORITHMS + 1, 1, 1);
02399   gtk_grid_attach (window->grid_algorithm,
02400                    GTK_WIDGET (window->spin_iterations), 1,
02401                    NALGORITHMS + 1, 1, 1);
02402   gtk_grid_attach (window->grid_algorithm,
02403                    GTK_WIDGET (window->label_tolerance), 0,
02404                    NALGORITHMS + 2, 1, 1);
02405   gtk_grid_attach (window->grid_algorithm,
02406                    GTK_WIDGET (window->spin_tolerance), 1,
02407                    NALGORITHMS + 2, 1, 1);
02408   gtk_grid_attach (window->grid_algorithm,
02409                    GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02410   gtk_grid_attach (window->grid_algorithm,
02411                    GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02412   gtk_grid_attach (window->grid_algorithm,
02413                    GTK_WIDGET (window->label_population), 0,
02414                    NALGORITHMS + 4, 1, 1);
02415   gtk_grid_attach (window->grid_algorithm,
02416                    GTK_WIDGET (window->spin_population), 1,
02417                    NALGORITHMS + 4, 1, 1);
02418   gtk_grid_attach (window->grid_algorithm,
02419                    GTK_WIDGET (window->label_generations), 0,
02420                    NALGORITHMS + 5, 1, 1);
02421   gtk_grid_attach (window->grid_algorithm,
02422                    GTK_WIDGET (window->spin_generations), 1,
02423                    NALGORITHMS + 5, 1, 1);
02424   gtk_grid_attach (window->grid_algorithm,
02425                    GTK_WIDGET (window->label_mutation), 0,
02426                    NALGORITHMS + 6, 1, 1);
02427   gtk_grid_attach (window->grid_algorithm,
02428                    GTK_WIDGET (window->spin_mutation), 1,
02429                    NALGORITHMS + 6, 1, 1);
02430   gtk_grid_attach (window->grid_algorithm,
02431                    GTK_WIDGET (window->label_reproduction), 0,
```

```
02432                    NALGORITHMS + 7, 1, 1);
02433   gtk_grid_attach (window->grid_algorithm,
02434                    GTK_WIDGET (window->spin_reproduction), 1,
02435                    NALGORITHMS + 7, 1, 1);
02436   gtk_grid_attach (window->grid_algorithm,
02437                    GTK_WIDGET (window->label_adaptation), 0,
02438                    NALGORITHMS + 8, 1, 1);
02439   gtk_grid_attach (window->grid_algorithm,
02440                    GTK_WIDGET (window->spin_adaptation), 1,
02441                    NALGORITHMS + 8, 1, 1);
02442   gtk_grid_attach (window->grid_algorithm,
02443                    GTK_WIDGET (window->check_direction), 0,
02444                    NALGORITHMS + 9, 2, 1);
02445   gtk_grid_attach (window->grid_algorithm,
02446                    GTK_WIDGET (window->grid_direction), 0,
02447                    NALGORITHMS + 10, 2, 1);
02448   gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
      label_threshold),
02449                    0, NALGORITHMS + 11, 1, 1);
02450   gtk_grid_attach (window->grid_algorithm,
02451                    GTK_WIDGET (window->scrolled_threshold), 1,
02452                    NALGORITHMS + 11, 1, 1);
02453   window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02454   gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02455                      GTK_WIDGET (window->grid_algorithm));
02456
02457   // Creating the variable widgets
02458   window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02459   gtk_widget_set_tooltip_text
02460     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02461   window->id_variable = g_signal_connect
02462     (window->combo_variable, "changed", window_set_variable, NULL);
02463   window->button_add_variable
02464     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02465                                                    GTK_ICON_SIZE_BUTTON);
02466   g_signal_connect
02467     (window->button_add_variable, "clicked",
      window_add_variable, NULL);
02468   gtk_widget_set_tooltip_text
02469     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02470   window->button_remove_variable
02471     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02472                                                    GTK_ICON_SIZE_BUTTON);
02473   g_signal_connect
02474     (window->button_remove_variable, "clicked",
      window_remove_variable, NULL);
02475   gtk_widget_set_tooltip_text
02476     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02477   window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02478   window->entry_variable = (GtkEntry *) gtk_entry_new ();
02479   gtk_widget_set_tooltip_text
02480     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02481   gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02482   window->id_variable_label = g_signal_connect
02483     (window->entry_variable, "changed", window_label_variable, NULL);
02484   window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02485   window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02486     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02487   gtk_widget_set_tooltip_text
02488     (GTK_WIDGET (window->spin_min),
02489      gettext ("Minimum initial value of the variable"));
02490   window->scrolled_min
02491     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02492   gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02493                      GTK_WIDGET (window->spin_min));
02494   g_signal_connect (window->spin_min, "value-changed",
02495                     window_rangemin_variable, NULL);
02496   window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02497   window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02498     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02499   gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_max),
02501      gettext ("Maximum initial value of the variable"));
02502   window->scrolled_max
02503     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02504   gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02505                      GTK_WIDGET (window->spin_max));
02506   g_signal_connect (window->spin_max, "value-changed",
02507                     window_rangemax_variable, NULL);
02508   window->check_minabs = (GtkCheckButton *)
02509     gtk_check_button_new_with_mnemonic (gettext ("_Absolute minimum"));
02510   g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02511   window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02512     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02513   gtk_widget_set_tooltip_text
02514     (GTK_WIDGET (window->spin_minabs),
02515      gettext ("Minimum allowed value of the variable"));
```

```
02516   window->scrolled_minabs
02517     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02518   gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02519                      GTK_WIDGET (window->spin_minabs));
02520   g_signal_connect (window->spin_minabs, "value-changed",
02521                     window_rangeminabs_variable, NULL);
02522   window->check_maxabs = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (gettext ("_Absolute maximum"));
02524   g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02525   window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02526     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02527   gtk_widget_set_tooltip_text
02528     (GTK_WIDGET (window->spin_maxabs),
02529      gettext ("Maximum allowed value of the variable"));
02530   window->scrolled_maxabs
02531     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02532   gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02533                      GTK_WIDGET (window->spin_maxabs));
02534   g_signal_connect (window->spin_maxabs, "value-changed",
02535                     window_rangemaxabs_variable, NULL);
02536   window->label_precision
02537     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02538   window->spin_precision = (GtkSpinButton *)
02539     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02540   gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_precision),
02542      gettext ("Number of precision floating point digits\n"
02543               "0 is for integer numbers"));
02544   g_signal_connect (window->spin_precision, "value-changed",
02545                     window_precision_variable, NULL);
02546   window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02547   window->spin_sweeps
02548     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02549   gtk_widget_set_tooltip_text
02550     (GTK_WIDGET (window->spin_sweeps),
02551      gettext ("Number of steps sweeping the variable"));
02552   g_signal_connect
02553     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02554   window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02555   window->spin_bits
02556     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02557   gtk_widget_set_tooltip_text
02558     (GTK_WIDGET (window->spin_bits),
02559      gettext ("Number of bits to encode the variable"));
02560   g_signal_connect
02561     (window->spin_bits, "value-changed", window_update_variable, NULL);
02562   window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02563   window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02564     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02565   gtk_widget_set_tooltip_text
02566     (GTK_WIDGET (window->spin_step),
02567      gettext ("Initial step size for the direction search method"));
02568   window->scrolled_step
02569     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02570   gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02571                      GTK_WIDGET (window->spin_step));
02572   g_signal_connect
02573     (window->spin_step, "value-changed", window_step_variable, NULL);
02574   window->grid_variable = (GtkGrid *) gtk_grid_new ();
02575   gtk_grid_attach (window->grid_variable,
02576                    GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02577   gtk_grid_attach (window->grid_variable,
02578                    GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02579   gtk_grid_attach (window->grid_variable,
02580                    GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02581   gtk_grid_attach (window->grid_variable,
02582                    GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02583   gtk_grid_attach (window->grid_variable,
02584                    GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02585   gtk_grid_attach (window->grid_variable,
02586                    GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02587   gtk_grid_attach (window->grid_variable,
02588                    GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02589   gtk_grid_attach (window->grid_variable,
02590                    GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02591   gtk_grid_attach (window->grid_variable,
02592                    GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02593   gtk_grid_attach (window->grid_variable,
02594                    GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02595   gtk_grid_attach (window->grid_variable,
02596                    GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02597   gtk_grid_attach (window->grid_variable,
02598                    GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02599   gtk_grid_attach (window->grid_variable,
02600                    GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02601   gtk_grid_attach (window->grid_variable,
02602                    GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
```

```
02603   gtk_grid_attach (window->grid_variable,
02604                    GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02605   gtk_grid_attach (window->grid_variable,
02606                    GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02607   gtk_grid_attach (window->grid_variable,
02608                    GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02609   gtk_grid_attach (window->grid_variable,
02610                    GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02611   gtk_grid_attach (window->grid_variable,
02612                    GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02613   gtk_grid_attach (window->grid_variable,
02614                    GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02615   gtk_grid_attach (window->grid_variable,
02616                    GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02617   window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02618   gtk_container_add (GTK_CONTAINER (window->frame_variable),
02619                      GTK_WIDGET (window->grid_variable));
02620
02621   // Creating the experiment widgets
02622   window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02623   gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02624                                gettext ("Experiment selector"));
02625   window->id_experiment = g_signal_connect
02626     (window->combo_experiment, "changed", window_set_experiment, NULL)
      ;
02627   window->button_add_experiment
02628     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02629                                                    GTK_ICON_SIZE_BUTTON);
02630   g_signal_connect
02631     (window->button_add_experiment, "clicked",
      window_add_experiment, NULL);
02632   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02633                                gettext ("Add experiment"));
02634   window->button_remove_experiment
02635     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02636                                                    GTK_ICON_SIZE_BUTTON);
02637   g_signal_connect (window->button_remove_experiment, "clicked",
02638                     window_remove_experiment, NULL);
02639   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02640                                gettext ("Remove experiment"));
02641   window->label_experiment
02642     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02643   window->button_experiment = (GtkFileChooserButton *)
02644     gtk_file_chooser_button_new (gettext ("Experimental data file"),
02645                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02646   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02647                                gettext ("Experimental data file"));
02648   window->id_experiment_name
02649     = g_signal_connect (window->button_experiment, "selection-changed",
02650                         window_name_experiment, NULL);
02651   gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02652   window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02653   window->spin_weight
02654     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02655   gtk_widget_set_tooltip_text
02656     (GTK_WIDGET (window->spin_weight),
02657      gettext ("Weight factor to build the objective function"));
02658   g_signal_connect
02659     (window->spin_weight, "value-changed", window_weight_experiment,
      NULL);
02660   window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02661   gtk_grid_attach (window->grid_experiment,
02662                    GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02663   gtk_grid_attach (window->grid_experiment,
02664                    GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02665   gtk_grid_attach (window->grid_experiment,
02666                    GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02667   gtk_grid_attach (window->grid_experiment,
02668                    GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02669   gtk_grid_attach (window->grid_experiment,
02670                    GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02671   gtk_grid_attach (window->grid_experiment,
02672                    GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02673   gtk_grid_attach (window->grid_experiment,
02674                    GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02675   for (i = 0; i < MAX_NINPUTS; ++i)
02676     {
02677       snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02678       window->check_template[i] = (GtkCheckButton *)
02679         gtk_check_button_new_with_label (buffer3);
02680       window->id_template[i]
02681         = g_signal_connect (window->check_template[i], "toggled",
02682                             window_inputs_experiment, NULL);
02683       gtk_grid_attach (window->grid_experiment,
02684                        GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02685       window->button_template[i] = (GtkFileChooserButton *)
02686         gtk_file_chooser_button_new (gettext ("Input template"),
```

```
02687                                     GTK_FILE_CHOOSER_ACTION_OPEN);
02688       gtk_widget_set_tooltip_text
02689         (GTK_WIDGET (window->button_template[i]),
02690          gettext ("Experimental input template file"));
02691       window->id_input[i]
02692         = g_signal_connect_swapped (window->button_template[i],
02693                                     "selection-changed",
02694                                     (void (*)) window_template_experiment,
02695                                     (void *) (size_t) i);
02696       gtk_grid_attach (window->grid_experiment,
02697                        GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02698     }
02699   window->frame_experiment
02700     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02701   gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02702                      GTK_WIDGET (window->grid_experiment));
02703
02704   // Creating the error norm widgets
02705   window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02706   window->grid_norm = (GtkGrid *) gtk_grid_new ();
02707   gtk_container_add (GTK_CONTAINER (window->frame_norm),
02708                      GTK_WIDGET (window->grid_norm));
02709   window->button_norm[0] = (GtkRadioButton *)
02710     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02711   gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02712                                tip_norm[0]);
02713   gtk_grid_attach (window->grid_norm,
02714                    GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02715   g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02716   for (i = 0; ++i < NNORMS;)
02717     {
02718       window->button_norm[i] = (GtkRadioButton *)
02719         gtk_radio_button_new_with_mnemonic
02720         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02721       gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02722                                    tip_norm[i]);
02723       gtk_grid_attach (window->grid_norm,
02724                        GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02725       g_signal_connect (window->button_norm[i], "clicked",
      window_update, NULL);
02726     }
02727   window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02728   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02729   window->spin_p = (GtkSpinButton *)
02730     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02731   gtk_widget_set_tooltip_text
02732     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02733   window->scrolled_p
02734     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02735   gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02736                      GTK_WIDGET (window->spin_p));
02737   gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02738   gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02739   gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02740                    1, 2, 1, 2);
02741
02742   // Creating the grid and attaching the widgets to the grid
02743   window->grid = (GtkGrid *) gtk_grid_new ();
02744   gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02745   gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02746   gtk_grid_attach (window->grid,
02747                    GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02748   gtk_grid_attach (window->grid,
02749                    GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02750   gtk_grid_attach (window->grid,
02751                    GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02752   gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02753   gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
      grid));
02754
02755   // Setting the window logo
02756   window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02757   gtk_window_set_icon (window->window, window->logo);
02758
02759   // Showing the window
02760   gtk_widget_show_all (GTK_WIDGET (window->window));
02761
02762   // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02763 #if GTK_MINOR_VERSION >= 16
02764   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02765   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02766   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02767   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02768   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02769   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02770   gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02771 #endif
```

```
02772
02773   // Reading initial example
02774   input_new ();
02775   buffer2 = g_get_current_dir ();
02776   buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02777   g_free (buffer2);
02778   window_read (buffer);
02779   g_free (buffer);
02780
02781 #if DEBUG_INTERFACE
02782   fprintf (stderr, "window_new: start\n");
02783 #endif
02784 }
```

5.13 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

- static GtkButton * **gtk_button_new_from_icon_name** (const char *name, GtkIconSize size)
- 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 ()

  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.

5.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burguete.

Copyright

Definition in file interface.h.

5.13.2 Function Documentation

5.13.2.1 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 561 of file utils.c.

```
00562 {
00563   unsigned int i;
00564   for (i = 0; i < n; ++i)
00565     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00566       break;
00567   return i;
00568 }
```

5.13.2.2 void input_save (char * *filename*)

Function to save the input file.

Parameters

filename	Input file name.

Definition at line 580 of file interface.c.

```
00581 {
00582   xmlDoc *doc;
00583   JsonGenerator *generator;
00584
00585 #if DEBUG_INTERFACE
00586   fprintf (stderr, "input_save: start\n");
00587 #endif
00588
00589   // Getting the input file directory
00590   input->name = g_path_get_basename (filename);
00591   input->directory = g_path_get_dirname (filename);
00592
00593   if (input->type == INPUT_TYPE_XML)
00594     {
00595       // Opening the input file
00596       doc = xmlNewDoc ((const xmlChar *) "1.0");
00597       input_save_xml (doc);
00598
00599       // Saving the XML file
00600       xmlSaveFormatFile (filename, doc, 1);
00601
00602       // Freeing memory
00603       xmlFreeDoc (doc);
00604     }
00605   else
00606     {
00607       // Opening the input file
00608       generator = json_generator_new ();
00609       json_generator_set_pretty (generator, TRUE);
00610       input_save_json (generator);
00611
00612       // Saving the JSON file
00613       json_generator_to_file (generator, filename, NULL);
00614
00615       // Freeing memory
00616       g_object_unref (generator);
00617     }
00618
00619 #if DEBUG_INTERFACE
00620   fprintf (stderr, "input_save: end\n");
00621 #endif
00622 }
```

Here is the call graph for this function:

5.13.2.3 unsigned int window_get_algorithm ( )

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 731 of file interface.c.

```
00732 {
00733   unsigned int i;
00734 #if DEBUG_INTERFACE
00735   fprintf (stderr, "window_get_algorithm: start\n");
00736 #endif
00737   i = gtk_array_get_active (window->button_algorithm,
      NALGORITHMS);
00738 #if DEBUG_INTERFACE
00739   fprintf (stderr, "window_get_algorithm: %u\n", i);
00740   fprintf (stderr, "window_get_algorithm: end\n");
00741 #endif
00742   return i;
00743 }
```

Here is the call graph for this function:

5.13.2.4 unsigned int window_get_direction ( )

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 751 of file interface.c.

```
00752 {
00753   unsigned int i;
00754 #if DEBUG_INTERFACE
00755   fprintf (stderr, "window_get_direction: start\n");
00756 #endif
00757   i = gtk_array_get_active (window->button_direction,
      NDIRECTIONS);
00758 #if DEBUG_INTERFACE
00759   fprintf (stderr, "window_get_direction: %u\n", i);
00760   fprintf (stderr, "window_get_direction: end\n");
00761 #endif
00762   return i;
00763 }
```

Here is the call graph for this function:

5.13.2.5 unsigned int window_get_norm ( )

Function to get the norm method number.

Returns

Norm method number.

Definition at line 771 of file interface.c.

```
00772 {
00773   unsigned int i;
00774 #if DEBUG_INTERFACE
00775   fprintf (stderr, "window_get_norm: start\n");
00776 #endif
00777   i = gtk_array_get_active (window->button_norm,
      NNORMS);
00778 #if DEBUG_INTERFACE
00779   fprintf (stderr, "window_get_norm: %u\n", i);
00780   fprintf (stderr, "window_get_norm: end\n");
00781 #endif
00782   return i;
00783 }
```

Here is the call graph for this function:

5.13.2.6 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 1876 of file interface.c.

```
01877 {
01878   unsigned int i;
01879   char *buffer;
01880 #if DEBUG_INTERFACE
01881   fprintf (stderr, "window_read: start\n");
01882 #endif
01883
01884   // Reading new input file
01885   input_free ();
01886   if (!input_open (filename))
01887     {
01888 #if DEBUG_INTERFACE
01889       fprintf (stderr, "window_read: end\n");
01890 #endif
01891       return 0;
01892     }
01893
01894   // Setting GTK+ widgets data
01895   gtk_entry_set_text (window->entry_result, input->result);
01896   gtk_entry_set_text (window->entry_variables, input->
      variables);
01897   buffer = g_build_filename (input->directory, input->
      simulator, NULL);
01898   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01899                                  (window->button_simulator), buffer);
01900   g_free (buffer);
01901   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01902                                 (size_t) input->evaluator);
01903   if (input->evaluator)
01904     {
01905       buffer = g_build_filename (input->directory, input->
      evaluator, NULL);
01906       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01907                                      (window->button_evaluator), buffer);
01908       g_free (buffer);
01909     }
01910   gtk_toggle_button_set_active
01911     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
      algorithm]), TRUE);
01912   switch (input->algorithm)
01913     {
01914     case ALGORITHM_MONTE_CARLO:
01915       gtk_spin_button_set_value (window->spin_simulations,
01916                                  (gdouble) input->nsimulations);
01917     case ALGORITHM_SWEEP:
01918       gtk_spin_button_set_value (window->spin_iterations,
01919                                  (gdouble) input->niterations);
01920       gtk_spin_button_set_value (window->spin_bests, (gdouble)
      input->nbest);
01921       gtk_spin_button_set_value (window->spin_tolerance,
      input->tolerance);
01922       gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
      check_direction),
01923                                     input->nsteps);
01924       if (input->nsteps)
01925         {
01926           gtk_toggle_button_set_active
01927             (GTK_TOGGLE_BUTTON (window->button_direction
01928                                 [input->direction]), TRUE);
01929           gtk_spin_button_set_value (window->spin_steps,
01930                                      (gdouble) input->nsteps);
01931           gtk_spin_button_set_value (window->spin_relaxation,
01932                                      (gdouble) input->relaxation);
01933           switch (input->direction)
01934             {
01935             case DIRECTION_METHOD_RANDOM:
01936               gtk_spin_button_set_value (window->spin_estimates,
01937                                          (gdouble) input->nestimates);
01938             }
01939         }
01940       break;
01941     default:
01942       gtk_spin_button_set_value (window->spin_population,
01943                                  (gdouble) input->nsimulations);
01944       gtk_spin_button_set_value (window->spin_generations,
01945                                  (gdouble) input->niterations);
01946       gtk_spin_button_set_value (window->spin_mutation, input->
      mutation_ratio);
01947       gtk_spin_button_set_value (window->spin_reproduction,
01948                                  input->reproduction_ratio);
01949       gtk_spin_button_set_value (window->spin_adaptation,
01950                                  input->adaptation_ratio);
01951     }
01952   gtk_toggle_button_set_active
01953     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01954   gtk_spin_button_set_value (window->spin_p, input->p);
01955   gtk_spin_button_set_value (window->spin_threshold, input->
```

```
      threshold);
01956   g_signal_handler_block (window->combo_experiment, window->
      id_experiment);
01957   g_signal_handler_block (window->button_experiment,
01958                           window->id_experiment_name);
01959   gtk_combo_box_text_remove_all (window->combo_experiment);
01960   for (i = 0; i < input->nexperiments; ++i)
01961     gtk_combo_box_text_append_text (window->combo_experiment,
01962                                     input->experiment[i].name);
01963   g_signal_handler_unblock
01964     (window->button_experiment, window->
      id_experiment_name);
01965   g_signal_handler_unblock (window->combo_experiment,
      window->id_experiment);
01966   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967   g_signal_handler_block (window->combo_variable, window->
      id_variable);
01968   g_signal_handler_block (window->entry_variable, window->
      id_variable_label);
01969   gtk_combo_box_text_remove_all (window->combo_variable);
01970   for (i = 0; i < input->nvariables; ++i)
01971     gtk_combo_box_text_append_text (window->combo_variable,
01972                                     input->variable[i].name);
01973   g_signal_handler_unblock (window->entry_variable, window->
      id_variable_label);
01974   g_signal_handler_unblock (window->combo_variable, window->
      id_variable);
01975   gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976   window_set_variable ();
01977   window_update ();
01978
01979 #if DEBUG_INTERFACE
01980   fprintf (stderr, "window_read: end\n");
01981 #endif
01982   return 1;
01983 }
```

Here is the call graph for this function:

5.13.2.7 int window_save ( )

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 824 of file interface.c.

```
00825 {
00826   GtkFileChooserDialog *dlg;
00827   GtkFileFilter *filter1, *filter2;
00828   char *buffer;
00829
00830 #if DEBUG_INTERFACE
00831   fprintf (stderr, "window_save: start\n");
00832 #endif
00833
00834   // Opening the saving dialog
00835   dlg = (GtkFileChooserDialog *)
00836     gtk_file_chooser_dialog_new (gettext ("Save file"),
00837                                  window->window,
00838                                  GTK_FILE_CHOOSER_ACTION_SAVE,
00839                                  gettext ("_Cancel"),
00840                                  GTK_RESPONSE_CANCEL,
00841                                  gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00842   gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00843   buffer = g_build_filename (input->directory, input->name, NULL);
00844   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00845   g_free (buffer);
00846
00847   // Adding XML filter
00848   filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00849   gtk_file_filter_set_name (filter1, "XML");
00850   gtk_file_filter_add_pattern (filter1, "*.xml");
00851   gtk_file_filter_add_pattern (filter1, "*.XML");
00852   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00853
00854   // Adding JSON filter
```

```
00855   filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00856   gtk_file_filter_set_name (filter2, "JSON");
00857   gtk_file_filter_add_pattern (filter2, "*.json");
00858   gtk_file_filter_add_pattern (filter2, "*.JSON");
00859   gtk_file_filter_add_pattern (filter2, "*.js");
00860   gtk_file_filter_add_pattern (filter2, "*.JS");
00861   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863   if (input->type == INPUT_TYPE_XML)
00864     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00865   else
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00867
00868   // If OK response then saving
00869   if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00870     {
00871       // Setting input file type
00872       filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00873       buffer = (char *) gtk_file_filter_get_name (filter1);
00874       if (!strcmp (buffer, "XML"))
00875         input->type = INPUT_TYPE_XML;
00876       else
00877         input->type = INPUT_TYPE_JSON;
00878
00879       // Adding properties to the root XML node
00880       input->simulator = gtk_file_chooser_get_filename
00881         (GTK_FILE_CHOOSER (window->button_simulator));
00882       if (gtk_toggle_button_get_active
00883           (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00884         input->evaluator = gtk_file_chooser_get_filename
00885           (GTK_FILE_CHOOSER (window->button_evaluator));
00886       else
00887         input->evaluator = NULL;
00888       if (input->type == INPUT_TYPE_XML)
00889         {
00890           input->result
00891             = (char *) xmlStrdup ((const xmlChar *)
00892                                   gtk_entry_get_text (window->entry_result));
00893           input->variables
00894             = (char *) xmlStrdup ((const xmlChar *)
00895                                   gtk_entry_get_text (window->
    entry_variables));
00896         }
00897       else
00898         {
00899           input->result = g_strdup (gtk_entry_get_text (window->
    entry_result));
00900           input->variables
00901             = g_strdup (gtk_entry_get_text (window->entry_variables));
00902         }
00903
00904       // Setting the algorithm
00905       switch (window_get_algorithm ())
00906         {
00907         case ALGORITHM_MONTE_CARLO:
00908           input->algorithm = ALGORITHM_MONTE_CARLO;
00909           input->nsimulations
00910             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00911           input->niterations
00912             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00913           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00914           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00915           window_save_direction ();
00916           break;
00917         case ALGORITHM_SWEEP:
00918           input->algorithm = ALGORITHM_SWEEP;
00919           input->niterations
00920             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921           input->tolerance = gtk_spin_button_get_value (window->
    spin_tolerance);
00922           input->nbest = gtk_spin_button_get_value_as_int (window->
    spin_bests);
00923           window_save_direction ();
00924           break;
00925         default:
00926           input->algorithm = ALGORITHM_GENETIC;
00927           input->nsimulations
00928             = gtk_spin_button_get_value_as_int (window->spin_population);
00929           input->niterations
00930             = gtk_spin_button_get_value_as_int (window->spin_generations);
00931           input->mutation_ratio
00932             = gtk_spin_button_get_value (window->spin_mutation);
00933           input->reproduction_ratio
00934             = gtk_spin_button_get_value (window->spin_reproduction);
00935           input->adaptation_ratio
```

```
00936             = gtk_spin_button_get_value (window->spin_adaptation);
00937           break;
00938         }
00939       input->norm = window_get_norm ();
00940       input->p = gtk_spin_button_get_value (window->spin_p);
00941       input->threshold = gtk_spin_button_get_value (window->
      spin_threshold);
00942
00943       // Saving the XML file
00944       buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00945       input_save (buffer);
00946
00947       // Closing and freeing memory
00948       g_free (buffer);
00949       gtk_widget_destroy (GTK_WIDGET (dlg));
00950 #if DEBUG_INTERFACE
00951       fprintf (stderr, "window_save: end\n");
00952 #endif
00953       return 1;
00954     }
00955
00956   // Closing and freeing memory
00957   gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959   fprintf (stderr, "window_save: end\n");
00960 #endif
00961   return 0;
00962 }
```

Here is the call graph for this function:

5.13.2.8 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 1520 of file interface.c.

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

5.14 interface.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
```

```
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef INTERFACE__H
00039 #define INTERFACE__H 1
00040
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
00049 {
00050   GtkDialog *dialog;
00051   GtkGrid *grid;
00052   GtkLabel *label_seed;
00054   GtkSpinButton *spin_seed;
00056   GtkLabel *label_threads;
00057   GtkSpinButton *spin_threads;
00058   GtkLabel *label_direction;
00059   GtkSpinButton *spin_direction;
00061 } Options;
00062
00067 typedef struct
00068 {
00069   GtkDialog *dialog;
00070   GtkLabel *label;
00071   GtkSpinner *spinner;
00072   GtkGrid *grid;
00073 } Running;
00074
00079 typedef struct
00080 {
00081   GtkWindow *window;
00082   GtkGrid *grid;
00083   GtkToolbar *bar_buttons;
00084   GtkToolButton *button_open;
00085   GtkToolButton *button_save;
00086   GtkToolButton *button_run;
00087   GtkToolButton *button_options;
00088   GtkToolButton *button_help;
00089   GtkToolButton *button_about;
00090   GtkToolButton *button_exit;
00091   GtkGrid *grid_files;
00092   GtkLabel *label_simulator;
00093   GtkFileChooserButton *button_simulator;
00095   GtkCheckButton *check_evaluator;
00096   GtkFileChooserButton *button_evaluator;
00098   GtkLabel *label_result;
00099   GtkEntry *entry_result;
00100   GtkLabel *label_variables;
00101   GtkEntry *entry_variables;
00102   GtkFrame *frame_norm;
00103   GtkGrid *grid_norm;
00104   GtkRadioButton *button_norm[NNORMS];
00106   GtkLabel *label_p;
00107   GtkSpinButton *spin_p;
00108   GtkScrolledWindow *scrolled_p;
00110   GtkFrame *frame_algorithm;
00111   GtkGrid *grid_algorithm;
00112   GtkRadioButton *button_algorithm[NALGORITHMS];
00114   GtkLabel *label_simulations;
00115   GtkSpinButton *spin_simulations;
00117   GtkLabel *label_iterations;
00118   GtkSpinButton *spin_iterations;
00120   GtkLabel *label_tolerance;
00121   GtkSpinButton *spin_tolerance;
00122   GtkLabel *label_bests;
00123   GtkSpinButton *spin_bests;
00124   GtkLabel *label_population;
00125   GtkSpinButton *spin_population;
```

```
00127   GtkLabel *label_generations;
00128   GtkSpinButton *spin_generations;
00130   GtkLabel *label_mutation;
00131   GtkSpinButton *spin_mutation;
00132   GtkLabel *label_reproduction;
00133   GtkSpinButton *spin_reproduction;
00135   GtkLabel *label_adaptation;
00136   GtkSpinButton *spin_adaptation;
00138   GtkCheckButton *check_direction;
00140   GtkGrid *grid_direction;
00142   GtkRadioButton *button_direction[NDIRECTIONS];
00144   GtkLabel *label_steps;
00145   GtkSpinButton *spin_steps;
00146   GtkLabel *label_estimates;
00147   GtkSpinButton *spin_estimates;
00149   GtkLabel *label_relaxation;
00151   GtkSpinButton *spin_relaxation;
00153   GtkLabel *label_threshold;
00154   GtkSpinButton *spin_threshold;
00155   GtkScrolledWindow *scrolled_threshold;
00157   GtkFrame *frame_variable;
00158   GtkGrid *grid_variable;
00159   GtkComboBoxText *combo_variable;
00161   GtkButton *button_add_variable;
00162   GtkButton *button_remove_variable;
00163   GtkLabel *label_variable;
00164   GtkEntry *entry_variable;
00165   GtkLabel *label_min;
00166   GtkSpinButton *spin_min;
00167   GtkScrolledWindow *scrolled_min;
00168   GtkLabel *label_max;
00169   GtkSpinButton *spin_max;
00170   GtkScrolledWindow *scrolled_max;
00171   GtkCheckButton *check_minabs;
00172   GtkSpinButton *spin_minabs;
00173   GtkScrolledWindow *scrolled_minabs;
00174   GtkCheckButton *check_maxabs;
00175   GtkSpinButton *spin_maxabs;
00176   GtkScrolledWindow *scrolled_maxabs;
00177   GtkLabel *label_precision;
00178   GtkSpinButton *spin_precision;
00179   GtkLabel *label_sweeps;
00180   GtkSpinButton *spin_sweeps;
00181   GtkLabel *label_bits;
00182   GtkSpinButton *spin_bits;
00183   GtkLabel *label_step;
00184   GtkSpinButton *spin_step;
00185   GtkScrolledWindow *scrolled_step;
00186   GtkFrame *frame_experiment;
00187   GtkGrid *grid_experiment;
00188   GtkComboBoxText *combo_experiment;
00189   GtkButton *button_add_experiment;
00190   GtkButton *button_remove_experiment;
00191   GtkLabel *label_experiment;
00192   GtkFileChooserButton *button_experiment;
00194   GtkLabel *label_weight;
00195   GtkSpinButton *spin_weight;
00196   GtkCheckButton *check_template[MAX_NINPUTS];
00198   GtkFileChooserButton *button_template[MAX_NINPUTS];
00200   GdkPixbuf *logo;
00201   Experiment *experiment;
00202   Variable *variable;
00203   char *application_directory;
00204   gulong id_experiment;
00205   gulong id_experiment_name;
00206   gulong id_variable;
00207   gulong id_variable_label;
00208   gulong id_template[MAX_NINPUTS];
00210   gulong id_input[MAX_NINPUTS];
00212   unsigned int nexperiments;
00213   unsigned int nvariables;
00214 } Window;
00215
00216 // Global variables
00217 extern const char *logo[];
00218 extern Options options[1];
00219 extern Running running[1];
00220 extern Window window[1];
00221
00222 // Inline functions
00223 #if GTK_MAJOR_VERSION <= 3 && 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 ();
00271
00272 #endif
```

5.15 main.c File Reference

Main source file.

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

Include dependency graph for main.c:

Macros

- #define **_GNU_SOURCE**
- #define DEBUG_MAIN 0

  Macro to debug main functions.

Functions

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

5.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file main.c.

5.16 main.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <locale.h>
00045 #include <gsl/gsl_rng.h>
00046 #include <libxml/parser.h>
00047 #include <libintl.h>
00048 #include <glib.h>
00049 #include <json-glib/json-glib.h>
```

```
00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #if HAVE_GTK
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #endif
00060 #include "genetic/genetic.h"
00061 #include "utils.h"
00062 #include "experiment.h"
00063 #include "variable.h"
00064 #include "input.h"
00065 #include "optimize.h"
00066 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069
00070 #define DEBUG_MAIN 0
00071
00072
00081 int
00082 main (int argn, char **argc)
00083 {
00084 #if HAVE_GTK
00085   char *buffer;
00086 #endif
00087
00088   // Starting pseudo-random numbers generator
00089 #if DEBUG_MAIN
00090   fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00091 #endif
00092   optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00093
00094   // Allowing spaces in the XML data file
00095 #if DEBUG_MAIN
00096   fprintf (stderr, "main: allowing spaces in the XML data file\n");
00097 #endif
00098   xmlKeepBlanksDefault (0);
00099
00100   // Starting MPI
00101 #if HAVE_MPI
00102 #if DEBUG_MAIN
00103   fprintf (stderr, "main: starting MPI\n");
00104 #endif
00105   MPI_Init (&argn, &argc);
00106   MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00107   MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00108   printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00109 #else
00110   ntasks = 1;
00111 #endif
00112
00113   // Resetting result and variables file names
00114 #if DEBUG_MAIN
00115   fprintf (stderr, "main: resetting result and variables file names\n");
00116 #endif
00117   input->result = input->variables = NULL;
00118
00119 #if HAVE_GTK
00120
00121   // Getting threads number and pseudo-random numbers generator seed
00122   nthreads_direction = nthreads = cores_number ();
00123   optimize->seed = DEFAULT_RANDOM_SEED;
00124
00125   // Setting local language and international floating point numbers notation
00126   setlocale (LC_ALL, "");
00127   setlocale (LC_NUMERIC, "C");
00128   window->application_directory = g_get_current_dir ();
00129   buffer = g_build_filename (window->application_directory,
      LOCALE_DIR, NULL);
00130   bindtextdomain (PROGRAM_INTERFACE, buffer);
00131   bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00132   textdomain (PROGRAM_INTERFACE);
00133
00134   // Initing GTK+
00135   gtk_disable_setlocale ();
00136   gtk_init (&argn, &argc);
00137
00138   // Opening the main window
00139   window_new ();
00140   gtk_main ();
00141
00142   // Freeing memory
00143   input_free ();
```

```
00144   g_free (buffer);
00145   gtk_widget_destroy (GTK_WIDGET (window->window));
00146   g_free (window->application_directory);
00147
00148 #else
00149
00150   // Checking syntax
00151   if (argn < 2)
00152     {
00153       printf ("The syntax is:\n"
00154               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00155               "[variables_file]\n");
00156       return 1;
00157     }
00158
00159   // Getting threads number and pseudo-random numbers generator seed
00160 #if DEBUG_MAIN
00161   fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00162            "generator seed\n");
00163 #endif
00164   nthreads_direction = nthreads = cores_number ();
00165   optimize->seed = DEFAULT_RANDOM_SEED;
00166   if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00167     {
00168       nthreads_direction = nthreads = atoi (argc[2]);
00169       if (!nthreads)
00170         {
00171           printf ("Bad threads number\n");
00172           return 2;
00173         }
00174       argc += 2;
00175       argn -= 2;
00176       if (argn > 2 && !strcmp (argc[1], "-seed"))
00177         {
00178           optimize->seed = atoi (argc[2]);
00179           argc += 2;
00180           argn -= 2;
00181         }
00182     }
00183   else if (argn > 2 && !strcmp (argc[1], "-seed"))
00184     {
00185       optimize->seed = atoi (argc[2]);
00186       argc += 2;
00187       argn -= 2;
00188       if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00189         {
00190           nthreads_direction = nthreads = atoi (argc[2]);
00191           if (!nthreads)
00192             {
00193               printf ("Bad threads number\n");
00194               return 2;
00195             }
00196           argc += 2;
00197           argn -= 2;
00198         }
00199     }
00200   printf ("nthreads=%u\n", nthreads);
00201   printf ("seed=%lu\n", optimize->seed);
00202
00203   // Checking arguments
00204 #if DEBUG_MAIN
00205   fprintf (stderr, "main: checking arguments\n");
00206 #endif
00207   if (argn > 4 || argn < 2)
00208     {
00209       printf ("The syntax is:\n"
00210               "./mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00211               "[variables_file]\n");
00212       return 1;
00213     }
00214   if (argn > 2)
00215     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00216   if (argn == 4)
00217     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00218
00219   // Making optimization
00220 #if DEBUG_MAIN
00221   fprintf (stderr, "main: making optimization\n");
00222 #endif
00223   if (input_open (argc[1]))
00224     optimize_open ();
00225
00226   // Freeing memory
00227 #if DEBUG_MAIN
00228   fprintf (stderr, "main: freeing memory and closing\n");
00229 #endif
00230   optimize_free ();
```

```
00231
00232 #endif
00233
00234   // Closing MPI
00235 #if HAVE_MPI
00236   MPI_Finalize ();
00237 #endif
00238
00239   // Freeing memory
00240   gsl_rng_free (optimize->rng);
00241
00242   // Closing
00243   return 0;
00244 }
```

5.17 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 <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 **_GNU_SOURCE**
- #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.

5.17.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file optimize.c.

5.17.2 Function Documentation

5.17.2.1 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 462 of file optimize.c.

```
00463 {
00464   unsigned int i, j;
00465   double e;
00466 #if DEBUG_OPTIMIZE
00467   fprintf (stderr, "optimize_best: start\n");
00468   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00469            optimize->nsaveds, optimize->nbest);
00470 #endif
00471   if (optimize->nsaveds < optimize->nbest
00472       || value < optimize->error_best[optimize->nsaveds - 1])
00473     {
00474       if (optimize->nsaveds < optimize->nbest)
00475         ++optimize->nsaveds;
```

```
00476       optimize->error_best[optimize->nsaveds - 1] = value;
00477       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00478       for (i = optimize->nsaveds; --i;)
00479         {
00480           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00481             {
00482               j = optimize->simulation_best[i];
00483               e = optimize->error_best[i];
00484               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00485               optimize->error_best[i] = optimize->
      error_best[i - 1];
00486               optimize->simulation_best[i - 1] = j;
00487               optimize->error_best[i - 1] = e;
00488             }
00489           else
00490             break;
00491         }
00492     }
00493 #if DEBUG_OPTIMIZE
00494   fprintf (stderr, "optimize_best: end\n");
00495 #endif
00496 }
```

5.17.2.2 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 787 of file optimize.c.

```
00788 {
00789 #if DEBUG_OPTIMIZE
00790   fprintf (stderr, "optimize_best_direction: start\n");
00791   fprintf (stderr,
00792            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00793            simulation, value, optimize->error_best[0]);
00794 #endif
00795   if (value < optimize->error_best[0])
00796     {
00797       optimize->error_best[0] = value;
00798       optimize->simulation_best[0] = simulation;
00799 #if DEBUG_OPTIMIZE
00800       fprintf (stderr,
00801                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00802                simulation, value);
00803 #endif
00804     }
00805 #if DEBUG_OPTIMIZE
00806   fprintf (stderr, "optimize_best_direction: end\n");
00807 #endif
00808 }
```

5.17.2.3 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

simulation	Simulation number.

Definition at line 817 of file optimize.c.

```
00818 {
00819   unsigned int i, j;
00820   double e;
00821 #if DEBUG_OPTIMIZE
00822   fprintf (stderr, "optimize_direction_sequential: start\n");
00823   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
```

```
00824             "nend_direction=%u\n",
00825             optimize->nstart_direction, optimize->
      nend_direction);
00826 #endif
00827   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829       j = simulation + i;
00830       e = optimize_norm (j);
00831       optimize_best_direction (j, e);
00832       optimize_save_variables (j, e);
00833       if (e < optimize->threshold)
00834         {
00835           optimize->stop = 1;
00836           break;
00837         }
00838 #if DEBUG_OPTIMIZE
00839       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00840 #endif
00841     }
00842 #if DEBUG_OPTIMIZE
00843   fprintf (stderr, "optimize_direction_sequential: end\n");
00844 #endif
00845 }
```

Here is the call graph for this function:

5.17.2.4 void * optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 855 of file optimize.c.

```
00856 {
00857   unsigned int i, thread;
00858   double e;
00859 #if DEBUG_OPTIMIZE
00860   fprintf (stderr, "optimize_direction_thread: start\n");
00861 #endif
00862   thread = data->thread;
00863 #if DEBUG_OPTIMIZE
00864   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865            thread,
00866            optimize->thread_direction[thread],
00867            optimize->thread_direction[thread + 1]);
00868 #endif
00869   for (i = optimize->thread_direction[thread];
00870        i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872       e = optimize_norm (i);
00873       g_mutex_lock (mutex);
00874       optimize_best_direction (i, e);
00875       optimize_save_variables (i, e);
00876       if (e < optimize->threshold)
00877         optimize->stop = 1;
00878       g_mutex_unlock (mutex);
00879       if (optimize->stop)
00880         break;
00881 #if DEBUG_OPTIMIZE
00882       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886   fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888   g_thread_exit (NULL);
00889   return NULL;
00890 }
```

Here is the call graph for this function:

5.17.2.5 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 929 of file optimize.c.

```
00931 {
00932   double x;
00933 #if DEBUG_OPTIMIZE
00934   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00935 #endif
00936   x = optimize->direction[variable];
00937   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00938     {
00939       if (estimate & 1)
00940         x += optimize->step[variable];
00941       else
00942         x -= optimize->step[variable];
00943     }
00944 #if DEBUG_OPTIMIZE
00945   fprintf (stderr,
00946            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00947            variable, x);
00948   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00949 #endif
00950   return x;
00951 }
```

5.17.2.6 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 902 of file optimize.c.

```
00904 {
00905   double x;
00906 #if DEBUG_OPTIMIZE
00907   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908 #endif
00909   x = optimize->direction[variable]
00910     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00911 #if DEBUG_OPTIMIZE
00912   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00913            variable, x);
00914   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915 #endif
00916   return x;
00917 }
```

5.17.2.7 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 1096 of file optimize.c.

```
01097 {
01098   unsigned int j;
01099   double objective;
01100   char buffer[64];
01101 #if DEBUG_OPTIMIZE
01102   fprintf (stderr, "optimize_genetic_objective: start\n");
01103 #endif
01104   for (j = 0; j < optimize->nvariables; ++j)
01105     {
01106       optimize->value[entity->id * optimize->nvariables + j]
01107         = genetic_get_variable (entity, optimize->genetic_variable + j);
01108     }
01109   objective = optimize_norm (entity->id);
01110   g_mutex_lock (mutex);
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01114       fprintf (optimize->file_variables, buffer,
01115                genetic_get_variable (entity, optimize->genetic_variable + j));
01116     }
01117   fprintf (optimize->file_variables, "%.14le\n", objective);
01118   g_mutex_unlock (mutex);
01119 #if DEBUG_OPTIMIZE
01120   fprintf (stderr, "optimize_genetic_objective: end\n");
01121 #endif
01122   return objective;
01123 }
```

5.17.2.8 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 103 of file optimize.c.

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

```
00144         {
00145           length = strlen (buffer3);
00146           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                              optimize->label[i], 0, NULL);
00148           g_free (buffer3);
00149         }
00150       g_regex_unref (regex);
00151       length = strlen (buffer2);
00152       snprintf (buffer, 32, "@value%u@", i + 1);
00153       regex = g_regex_new (buffer, 0, 0, NULL);
00154       snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->
      nvariables + i]);
00156
00157 #if DEBUG_OPTIMIZE
00158       fprintf (stderr, "optimize_input: value=%s\n", value);
00159 #endif
00160       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00161                                          0, NULL);
00162       g_free (buffer2);
00163       g_regex_unref (regex);
00164     }
00165
00166   // Saving input file
00167   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168   g_free (buffer3);
00169   fclose (file);
00170
00171 optimize_input_end:
00172 #if DEBUG_OPTIMIZE
00173   fprintf (stderr, "optimize_input: end\n");
00174 #endif
00175   return;
00176 }
```

5.17.2.9 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 585 of file optimize.c.

```
00587 {
00588   unsigned int i, j, k, s[optimize->nbest];
00589   double e[optimize->nbest];
00590 #if DEBUG_OPTIMIZE
00591   fprintf (stderr, "optimize_merge: start\n");
00592 #endif
00593   i = j = k = 0;
00594   do
00595     {
00596       if (i == optimize->nsaveds)
00597         {
00598           s[k] = simulation_best[j];
00599           e[k] = error_best[j];
00600           ++j;
00601           ++k;
00602           if (j == nsaveds)
00603             break;
00604         }
00605       else if (j == nsaveds)
00606         {
00607           s[k] = optimize->simulation_best[i];
00608           e[k] = optimize->error_best[i];
00609           ++i;
00610           ++k;
00611           if (i == optimize->nsaveds)
00612             break;
00613         }
00614       else if (optimize->error_best[i] > error_best[j])
00615         {
00616           s[k] = simulation_best[j];
00617           e[k] = error_best[j];
00618           ++j;
00619           ++k;
```

```
00620         }
00621       else
00622         {
00623           s[k] = optimize->simulation_best[i];
00624           e[k] = optimize->error_best[i];
00625           ++i;
00626           ++k;
00627         }
00628     }
00629   while (k < optimize->nbest);
00630   optimize->nsaveds = k;
00631   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632   memcpy (optimize->error_best, e, k * sizeof (double));
00633 #if DEBUG_OPTIMIZE
00634   fprintf (stderr, "optimize_merge: end\n");
00635 #endif
00636 }
```

5.17.2.10 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 295 of file optimize.c.

```
00296 {
00297   double e, ei;
00298   unsigned int i;
00299 #if DEBUG_OPTIMIZE
00300   fprintf (stderr, "optimize_norm_euclidian: start\n");
00301 #endif
00302   e = 0.;
00303   for (i = 0; i < optimize->nexperiments; ++i)
00304     {
00305       ei = optimize_parse (simulation, i);
00306       e += ei * ei;
00307     }
00308   e = sqrt (e);
00309 #if DEBUG_OPTIMIZE
00310   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00311   fprintf (stderr, "optimize_norm_euclidian: end\n");
00312 #endif
00313   return e;
00314 }
```

Here is the call graph for this function:

5.17.2.11 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 324 of file optimize.c.

```
00325 {
00326   double e, ei;
00327   unsigned int i;
00328 #if DEBUG_OPTIMIZE
00329   fprintf (stderr, "optimize_norm_maximum: start\n");
00330 #endif
00331   e = 0.;
00332   for (i = 0; i < optimize->nexperiments; ++i)
00333     {
00334       ei = fabs (optimize_parse (simulation, i));
00335       e = fmax (e, ei);
00336     }
00337 #if DEBUG_OPTIMIZE
00338   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00339   fprintf (stderr, "optimize_norm_maximum: end\n");
00340 #endif
00341   return e;
00342 }
```

Here is the call graph for this function:

5.17.2.12 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 352 of file optimize.c.

```
00353 {
00354   double e, ei;
00355   unsigned int i;
00356 #if DEBUG_OPTIMIZE
00357   fprintf (stderr, "optimize_norm_p: start\n");
00358 #endif
00359   e = 0.;
00360   for (i = 0; i < optimize->nexperiments; ++i)
00361     {
00362       ei = fabs (optimize_parse (simulation, i));
00363       e += pow (ei, optimize->p);
00364     }
00365   e = pow (e, 1. / optimize->p);
00366 #if DEBUG_OPTIMIZE
00367   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00368   fprintf (stderr, "optimize_norm_p: end\n");
00369 #endif
00370   return e;
00371 }
```

Here is the call graph for this function:

5.17.2.13 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 381 of file optimize.c.

```
00382 {
00383   double e;
00384   unsigned int i;
00385 #if DEBUG_OPTIMIZE
00386   fprintf (stderr, "optimize_norm_taxicab: start\n");
00387 #endif
00388   e = 0.;
00389   for (i = 0; i < optimize->nexperiments; ++i)
00390     e += fabs (optimize_parse (simulation, i));
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00393   fprintf (stderr, "optimize_norm_taxicab: end\n");
00394 #endif
00395   return e;
00396 }
```

Here is the call graph for this function:

5.17.2.14 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 189 of file optimize.c.

```
00190 {
00191   unsigned int i;
00192   double e;
00193   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00194     *buffer3, *buffer4;
00195   FILE *file_result;
00196
00197 #if DEBUG_OPTIMIZE
00198   fprintf (stderr, "optimize_parse: start\n");
00199   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00200            experiment);
00201 #endif
00202
00203   // Opening input files
00204   for (i = 0; i < optimize->ninputs; ++i)
00205     {
00206       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00207 #if DEBUG_OPTIMIZE
00208       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00209 #endif
00210       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00211     }
00212   for (; i < MAX_NINPUTS; ++i)
00213     strcpy (&input[i][0], "");
00214 #if DEBUG_OPTIMIZE
00215   fprintf (stderr, "optimize_parse: parsing end\n");
00216 #endif
00217
00218   // Performing the simulation
00219   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00220   buffer2 = g_path_get_dirname (optimize->simulator);
00221   buffer3 = g_path_get_basename (optimize->simulator);
00222   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00223   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00224             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00225             input[6], input[7], output);
00226   g_free (buffer4);
00227   g_free (buffer3);
00228   g_free (buffer2);
00229 #if DEBUG_OPTIMIZE
00230   fprintf (stderr, "optimize_parse: %s\n", buffer);
00231 #endif
00232   system (buffer);
```

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

Here is the call graph for this function:

5.17.2.15 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 434 of file optimize.c.

```
00435 {
00436   unsigned int i;
00437   char buffer[64];
00438 #if DEBUG_OPTIMIZE
00439   fprintf (stderr, "optimize_save_variables: start\n");
00440 #endif
00441   for (i = 0; i < optimize->nvariables; ++i)
00442     {
00443       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444       fprintf (optimize->file_variables, buffer,
00445                optimize->value[simulation * optimize->
      nvariables + i]);
00446     }
```

```
00447   fprintf (optimize->file_variables, "%.14le\n", error);
00448 #if DEBUG_OPTIMIZE
00449   fprintf (stderr, "optimize_save_variables: end\n");
00450 #endif
00451 }
```

5.17.2.16 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 960 of file optimize.c.

```
00961 {
00962   GThread *thread[nthreads_direction];
00963   ParallelData data[nthreads_direction];
00964   unsigned int i, j, k, b;
00965 #if DEBUG_OPTIMIZE
00966   fprintf (stderr, "optimize_step_direction: start\n");
00967 #endif
00968   for (i = 0; i < optimize->nestimates; ++i)
00969     {
00970       k = (simulation + i) * optimize->nvariables;
00971       b = optimize->simulation_best[0] * optimize->
      nvariables;
00972 #if DEBUG_OPTIMIZE
00973       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00974                simulation + i, optimize->simulation_best[0]);
00975 #endif
00976       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00977         {
00978 #if DEBUG_OPTIMIZE
00979           fprintf (stderr,
00980                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00981                    i, j, optimize->value[b]);
00982 #endif
00983           optimize->value[k]
00984             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00985           optimize->value[k] = fmin (fmax (optimize->value[k],
00986                                            optimize->rangeminabs[j]),
00987                                      optimize->rangemaxabs[j]);
00988 #if DEBUG_OPTIMIZE
00989           fprintf (stderr,
00990                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00991                    i, j, optimize->value[k]);
00992 #endif
00993         }
00994     }
00995   if (nthreads_direction == 1)
00996     optimize_direction_sequential (simulation);
00997   else
00998     {
00999       for (i = 0; i <= nthreads_direction; ++i)
01000         {
01001           optimize->thread_direction[i]
01002             = simulation + optimize->nstart_direction
01003             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01004             / nthreads_direction;
01005 #if DEBUG_OPTIMIZE
01006           fprintf (stderr,
01007                    "optimize_step_direction: i=%u thread_direction=%u\n",
01008                    i, optimize->thread_direction[i]);
01009 #endif
01010         }
01011       for (i = 0; i < nthreads_direction; ++i)
01012         {
01013           data[i].thread = i;
01014           thread[i] = g_thread_new
01015             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01016         }
01017       for (i = 0; i < nthreads_direction; ++i)
01018         g_thread_join (thread[i]);
01019     }
01020 #if DEBUG_OPTIMIZE
01021   fprintf (stderr, "optimize_step_direction: end\n");
01022 #endif
01023 }
```

Here is the call graph for this function:

5.17.2.17 void * optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 539 of file optimize.c.

```
00540 {
00541   unsigned int i, thread;
00542   double e;
00543 #if DEBUG_OPTIMIZE
00544   fprintf (stderr, "optimize_thread: start\n");
00545 #endif
00546   thread = data->thread;
00547 #if DEBUG_OPTIMIZE
00548   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00549            optimize->thread[thread], optimize->thread[thread + 1]);
00550 #endif
00551   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00552     {
00553       e = optimize_norm (i);
00554       g_mutex_lock (mutex);
00555       optimize_best (i, e);
00556       optimize_save_variables (i, e);
00557       if (e < optimize->threshold)
00558         optimize->stop = 1;
00559       g_mutex_unlock (mutex);
00560       if (optimize->stop)
00561         break;
00562 #if DEBUG_OPTIMIZE
00563       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00564 #endif
00565     }
00566 #if DEBUG_OPTIMIZE
00567   fprintf (stderr, "optimize_thread: end\n");
00568 #endif
00569   g_thread_exit (NULL);
00570   return NULL;
00571 }
```

Here is the call graph for this function:

5.18 optimize.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <math.h>
00044 #include <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 #elif !defined(__BSD_VISIBLE)
00053 #include <alloca.h>
00054 #endif
00055 #if HAVE_MPI
00056 #include <mpi.h>
00057 #endif
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
00065 #define DEBUG_OPTIMIZE 0
00066
00067
00071 #ifdef G_OS_WIN32
00072 #define RM "del"
00073 #else
00074 #define RM "rm"
00075 #endif
00076
00077 int ntasks;
00078 unsigned int nthreads;
00079 unsigned int nthreads_direction;
00081 GMutex mutex[1];
00082 void (*optimize_algorithm) ();
00084 double (*optimize_estimate_direction) (unsigned int variable,
00085                                        unsigned int estimate);
00087 double (*optimize_norm) (unsigned int simulation);
00089 Optimize optimize[1];
00090
00102 void
00103 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00104 {
00105   unsigned int i;
00106   char buffer[32], value[32], *buffer2, *buffer3, *content;
00107   FILE *file;
00108   gsize length;
00109   GRegex *regex;
00110
00111 #if DEBUG_OPTIMIZE
00112   fprintf (stderr, "optimize_input: start\n");
00113 #endif
00114
00115   // Checking the file
00116   if (!template)
00117     goto optimize_input_end;
00118
00119   // Opening template
00120   content = g_mapped_file_get_contents (template);
00121   length = g_mapped_file_get_length (template);
00122 #if DEBUG_OPTIMIZE
00123   fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00124 #endif
00125   file = g_fopen (input, "w");
00126
00127   // Parsing template
00128   for (i = 0; i < optimize->nvariables; ++i)
00129     {
00130 #if DEBUG_OPTIMIZE
00131       fprintf (stderr, "optimize_input: variable=%u\n", i);
00132 #endif
```

```
00133       snprintf (buffer, 32, "@variable%u@", i + 1);
00134       regex = g_regex_new (buffer, 0, 0, NULL);
00135       if (i == 0)
00136         {
00137           buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                              optimize->label[i], 0, NULL);
00139 #if DEBUG_OPTIMIZE
00140           fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00141 #endif
00142         }
00143       else
00144         {
00145           length = strlen (buffer3);
00146           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                              optimize->label[i], 0, NULL);
00148           g_free (buffer3);
00149         }
00150       g_regex_unref (regex);
00151       length = strlen (buffer2);
00152       snprintf (buffer, 32, "@value%u@", i + 1);
00153       regex = g_regex_new (buffer, 0, 0, NULL);
00154       snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->nvariables + i]);
00156
00157 #if DEBUG_OPTIMIZE
00158       fprintf (stderr, "optimize_input: value=%s\n", value);
00159 #endif
00160       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00161                                          0, NULL);
00162       g_free (buffer2);
00163       g_regex_unref (regex);
00164     }
00165
00166   // Saving input file
00167   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168   g_free (buffer3);
00169   fclose (file);
00170
00171 optimize_input_end:
00172 #if DEBUG_OPTIMIZE
00173   fprintf (stderr, "optimize_input: end\n");
00174 #endif
00175   return;
00176 }
00177
00188 double
00189 optimize_parse (unsigned int simulation, unsigned int experiment)
00190 {
00191   unsigned int i;
00192   double e;
00193   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00194     *buffer3, *buffer4;
00195   FILE *file_result;
00196
00197 #if DEBUG_OPTIMIZE
00198   fprintf (stderr, "optimize_parse: start\n");
00199   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00200            experiment);
00201 #endif
00202
00203   // Opening input files
00204   for (i = 0; i < optimize->ninputs; ++i)
00205     {
00206       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00207 #if DEBUG_OPTIMIZE
00208       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00209 #endif
00210       optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00211     }
00212   for (; i < MAX_NINPUTS; ++i)
00213     strcpy (&input[i][0], "");
00214 #if DEBUG_OPTIMIZE
00215   fprintf (stderr, "optimize_parse: parsing end\n");
00216 #endif
00217
00218   // Performing the simulation
00219   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00220   buffer2 = g_path_get_dirname (optimize->simulator);
00221   buffer3 = g_path_get_basename (optimize->simulator);
00222   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00223   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00224             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00225             input[6], input[7], output);
00226   g_free (buffer4);
00227   g_free (buffer3);
00228   g_free (buffer2);
00229 #if DEBUG_OPTIMIZE
```

```
00230   fprintf (stderr, "optimize_parse: %s\n", buffer);
00231 #endif
00232   system (buffer);
00233
00234   // Checking the objective value function
00235   if (optimize->evaluator)
00236     {
00237       snprintf (result, 32, "result-%u-%u", simulation, experiment);
00238       buffer2 = g_path_get_dirname (optimize->evaluator);
00239       buffer3 = g_path_get_basename (optimize->evaluator);
00240       buffer4 = g_build_filename (buffer2, buffer3, NULL);
00241       snprintf (buffer, 512, "\"%s\" %s %s %s",
00242                 buffer4, output, optimize->experiment[experiment], result);
00243       g_free (buffer4);
00244       g_free (buffer3);
00245       g_free (buffer2);
00246 #if DEBUG_OPTIMIZE
00247       fprintf (stderr, "optimize_parse: %s\n", buffer);
00248 #endif
00249       system (buffer);
00250       file_result = g_fopen (result, "r");
00251       e = atof (fgets (buffer, 512, file_result));
00252       fclose (file_result);
00253     }
00254   else
00255     {
00256       strcpy (result, "");
00257       file_result = g_fopen (output, "r");
00258       e = atof (fgets (buffer, 512, file_result));
00259       fclose (file_result);
00260     }
00261
00262   // Removing files
00263 #if !DEBUG_OPTIMIZE
00264   for (i = 0; i < optimize->ninputs; ++i)
00265     {
00266       if (optimize->file[i][0])
00267         {
00268           snprintf (buffer, 512, RM " %s", &input[i][0]);
00269           system (buffer);
00270         }
00271     }
00272   snprintf (buffer, 512, RM " %s %s", output, result);
00273   system (buffer);
00274 #endif
00275
00276   // Processing pending events
00277   show_pending ();
00278
00279 #if DEBUG_OPTIMIZE
00280   fprintf (stderr, "optimize_parse: end\n");
00281 #endif
00282
00283   // Returning the objective function
00284   return e * optimize->weight[experiment];
00285 }
00286
00294 double
00295 optimize_norm_euclidian (unsigned int simulation)
00296 {
00297   double e, ei;
00298   unsigned int i;
00299 #if DEBUG_OPTIMIZE
00300   fprintf (stderr, "optimize_norm_euclidian: start\n");
00301 #endif
00302   e = 0.;
00303   for (i = 0; i < optimize->nexperiments; ++i)
00304     {
00305       ei = optimize_parse (simulation, i);
00306       e += ei * ei;
00307     }
00308   e = sqrt (e);
00309 #if DEBUG_OPTIMIZE
00310   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00311   fprintf (stderr, "optimize_norm_euclidian: end\n");
00312 #endif
00313   return e;
00314 }
00315
00323 double
00324 optimize_norm_maximum (unsigned int simulation)
00325 {
00326   double e, ei;
00327   unsigned int i;
00328 #if DEBUG_OPTIMIZE
00329   fprintf (stderr, "optimize_norm_maximum: start\n");
00330 #endif
```

```
00331   e = 0.;
00332   for (i = 0; i < optimize->nexperiments; ++i)
00333     {
00334       ei = fabs (optimize_parse (simulation, i));
00335       e = fmax (e, ei);
00336     }
00337 #if DEBUG_OPTIMIZE
00338   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00339   fprintf (stderr, "optimize_norm_maximum: end\n");
00340 #endif
00341   return e;
00342 }
00343
00351 double
00352 optimize_norm_p (unsigned int simulation)
00353 {
00354   double e, ei;
00355   unsigned int i;
00356 #if DEBUG_OPTIMIZE
00357   fprintf (stderr, "optimize_norm_p: start\n");
00358 #endif
00359   e = 0.;
00360   for (i = 0; i < optimize->nexperiments; ++i)
00361     {
00362       ei = fabs (optimize_parse (simulation, i));
00363       e += pow (ei, optimize->p);
00364     }
00365   e = pow (e, 1. / optimize->p);
00366 #if DEBUG_OPTIMIZE
00367   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00368   fprintf (stderr, "optimize_norm_p: end\n");
00369 #endif
00370   return e;
00371 }
00372
00380 double
00381 optimize_norm_taxicab (unsigned int simulation)
00382 {
00383   double e;
00384   unsigned int i;
00385 #if DEBUG_OPTIMIZE
00386   fprintf (stderr, "optimize_norm_taxicab: start\n");
00387 #endif
00388   e = 0.;
00389   for (i = 0; i < optimize->nexperiments; ++i)
00390     e += fabs (optimize_parse (simulation, i));
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00393   fprintf (stderr, "optimize_norm_taxicab: end\n");
00394 #endif
00395   return e;
00396 }
00397
00402 void
00403 optimize_print ()
00404 {
00405   unsigned int i;
00406   char buffer[512];
00407 #if HAVE_MPI
00408   if (optimize->mpi_rank)
00409     return;
00410 #endif
00411   printf ("%s\n", gettext ("Best result"));
00412   fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
00413   printf ("error = %.15le\n", optimize->error_old[0]);
00414   fprintf (optimize->file_result, "error = %.15le\n", optimize->
      error_old[0]);
00415   for (i = 0; i < optimize->nvariables; ++i)
00416     {
00417       snprintf (buffer, 512, "%s = %s\n",
00418                 optimize->label[i], format[optimize->precision[i]]);
00419       printf (buffer, optimize->value_old[i]);
00420       fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00421     }
00422   fflush (optimize->file_result);
00423 }
00424
00433 void
00434 optimize_save_variables (unsigned int simulation, double error)
00435 {
00436   unsigned int i;
00437   char buffer[64];
00438 #if DEBUG_OPTIMIZE
00439   fprintf (stderr, "optimize_save_variables: start\n");
00440 #endif
00441   for (i = 0; i < optimize->nvariables; ++i)
00442     {
```

```
00443       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444       fprintf (optimize->file_variables, buffer,
00445                optimize->value[simulation * optimize->nvariables + i]);
00446     }
00447   fprintf (optimize->file_variables, "%.14le\n", error);
00448 #if DEBUG_OPTIMIZE
00449   fprintf (stderr, "optimize_save_variables: end\n");
00450 #endif
00451 }
00452
00461 void
00462 optimize_best (unsigned int simulation, double value)
00463 {
00464   unsigned int i, j;
00465   double e;
00466 #if DEBUG_OPTIMIZE
00467   fprintf (stderr, "optimize_best: start\n");
00468   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00469            optimize->nsaveds, optimize->nbest);
00470 #endif
00471   if (optimize->nsaveds < optimize->nbest
00472       || value < optimize->error_best[optimize->nsaveds - 1])
00473     {
00474       if (optimize->nsaveds < optimize->nbest)
00475         ++optimize->nsaveds;
00476       optimize->error_best[optimize->nsaveds - 1] = value;
00477       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00478       for (i = optimize->nsaveds; --i;)
00479         {
00480           if (optimize->error_best[i] < optimize->error_best[i - 1])
00481             {
00482               j = optimize->simulation_best[i];
00483               e = optimize->error_best[i];
00484               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00485               optimize->error_best[i] = optimize->error_best[i - 1];
00486               optimize->simulation_best[i - 1] = j;
00487               optimize->error_best[i - 1] = e;
00488             }
00489           else
00490             break;
00491         }
00492     }
00493 #if DEBUG_OPTIMIZE
00494   fprintf (stderr, "optimize_best: end\n");
00495 #endif
00496 }
00497
00502 void
00503 optimize_sequential ()
00504 {
00505   unsigned int i;
00506   double e;
00507 #if DEBUG_OPTIMIZE
00508   fprintf (stderr, "optimize_sequential: start\n");
00509   fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00510            optimize->nstart, optimize->nend);
00511 #endif
00512   for (i = optimize->nstart; i < optimize->nend; ++i)
00513     {
00514       e = optimize_norm (i);
00515       optimize_best (i, e);
00516       optimize_save_variables (i, e);
00517       if (e < optimize->threshold)
00518         {
00519           optimize->stop = 1;
00520           break;
00521         }
00522 #if DEBUG_OPTIMIZE
00523       fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00524 #endif
00525     }
00526 #if DEBUG_OPTIMIZE
00527   fprintf (stderr, "optimize_sequential: end\n");
00528 #endif
00529 }
00530
00538 void *
00539 optimize_thread (ParallelData * data)
00540 {
00541   unsigned int i, thread;
00542   double e;
00543 #if DEBUG_OPTIMIZE
00544   fprintf (stderr, "optimize_thread: start\n");
00545 #endif
00546   thread = data->thread;
00547 #if DEBUG_OPTIMIZE
```

```
00548   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00549            optimize->thread[thread], optimize->thread[thread + 1]);
00550 #endif
00551   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00552     {
00553       e = optimize_norm (i);
00554       g_mutex_lock (mutex);
00555       optimize_best (i, e);
00556       optimize_save_variables (i, e);
00557       if (e < optimize->threshold)
00558         optimize->stop = 1;
00559       g_mutex_unlock (mutex);
00560       if (optimize->stop)
00561         break;
00562 #if DEBUG_OPTIMIZE
00563       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00564 #endif
00565     }
00566 #if DEBUG_OPTIMIZE
00567   fprintf (stderr, "optimize_thread: end\n");
00568 #endif
00569   g_thread_exit (NULL);
00570   return NULL;
00571 }
00572
00584 void
00585 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00586                 double *error_best)
00587 {
00588   unsigned int i, j, k, s[optimize->nbest];
00589   double e[optimize->nbest];
00590 #if DEBUG_OPTIMIZE
00591   fprintf (stderr, "optimize_merge: start\n");
00592 #endif
00593   i = j = k = 0;
00594   do
00595     {
00596       if (i == optimize->nsaveds)
00597         {
00598           s[k] = simulation_best[j];
00599           e[k] = error_best[j];
00600           ++j;
00601           ++k;
00602           if (j == nsaveds)
00603             break;
00604         }
00605       else if (j == nsaveds)
00606         {
00607           s[k] = optimize->simulation_best[i];
00608           e[k] = optimize->error_best[i];
00609           ++i;
00610           ++k;
00611           if (i == optimize->nsaveds)
00612             break;
00613         }
00614       else if (optimize->error_best[i] > error_best[j])
00615         {
00616           s[k] = simulation_best[j];
00617           e[k] = error_best[j];
00618           ++j;
00619           ++k;
00620         }
00621       else
00622         {
00623           s[k] = optimize->simulation_best[i];
00624           e[k] = optimize->error_best[i];
00625           ++i;
00626           ++k;
00627         }
00628     }
00629   while (k < optimize->nbest);
00630   optimize->nsaveds = k;
00631   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632   memcpy (optimize->error_best, e, k * sizeof (double));
00633 #if DEBUG_OPTIMIZE
00634   fprintf (stderr, "optimize_merge: end\n");
00635 #endif
00636 }
00637
00642 #if HAVE_MPI
00643 void
00644 optimize_synchronise ()
00645 {
00646   unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00647   double error_best[optimize->nbest];
00648   MPI_Status mpi_stat;
00649 #if DEBUG_OPTIMIZE
```

```
00650   fprintf (stderr, "optimize_synchronise: start\n");
00651 #endif
00652   if (optimize->mpi_rank == 0)
00653     {
00654       for (i = 1; i < ntasks; ++i)
00655         {
00656           MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00657           MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00658                     MPI_COMM_WORLD, &mpi_stat);
00659           MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00660                     MPI_COMM_WORLD, &mpi_stat);
00661           optimize_merge (nsaveds, simulation_best, error_best);
00662           MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00663           if (stop)
00664             optimize->stop = 1;
00665         }
00666       for (i = 1; i < ntasks; ++i)
00667         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00668     }
00669   else
00670     {
00671       MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00672       MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00673                 MPI_COMM_WORLD);
00674       MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00675                 MPI_COMM_WORLD);
00676       MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00677       MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00678       if (stop)
00679         optimize->stop = 1;
00680     }
00681 #if DEBUG_OPTIMIZE
00682   fprintf (stderr, "optimize_synchronise: end\n");
00683 #endif
00684 }
00685 #endif
00686
00691 void
00692 optimize_sweep ()
00693 {
00694   unsigned int i, j, k, l;
00695   double e;
00696   GThread *thread[nthreads];
00697   ParallelData data[nthreads];
00698 #if DEBUG_OPTIMIZE
00699   fprintf (stderr, "optimize_sweep: start\n");
00700 #endif
00701   for (i = 0; i < optimize->nsimulations; ++i)
00702     {
00703       k = i;
00704       for (j = 0; j < optimize->nvariables; ++j)
00705         {
00706           l = k % optimize->nsweeps[j];
00707           k /= optimize->nsweeps[j];
00708           e = optimize->rangemin[j];
00709           if (optimize->nsweeps[j] > 1)
00710             e += l * (optimize->rangemax[j] - optimize->rangemin[j])
00711               / (optimize->nsweeps[j] - 1);
00712           optimize->value[i * optimize->nvariables + j] = e;
00713         }
00714     }
00715   optimize->nsaveds = 0;
00716   if (nthreads <= 1)
00717     optimize_sequential ();
00718   else
00719     {
00720       for (i = 0; i < nthreads; ++i)
00721         {
00722           data[i].thread = i;
00723           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00724         }
00725       for (i = 0; i < nthreads; ++i)
00726         g_thread_join (thread[i]);
00727     }
00728 #if HAVE_MPI
00729   // Communicating tasks results
00730   optimize_synchronise ();
00731 #endif
00732 #if DEBUG_OPTIMIZE
00733   fprintf (stderr, "optimize_sweep: end\n");
00734 #endif
00735 }
00736
00741 void
00742 optimize_MonteCarlo ()
00743 {
00744   unsigned int i, j;
```

```
00745   GThread *thread[nthreads];
00746   ParallelData data[nthreads];
00747 #if DEBUG_OPTIMIZE
00748   fprintf (stderr, "optimize_MonteCarlo: start\n");
00749 #endif
00750   for (i = 0; i < optimize->nsimulations; ++i)
00751     for (j = 0; j < optimize->nvariables; ++j)
00752       optimize->value[i * optimize->nvariables + j]
00753         = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00754         * (optimize->rangemax[j] - optimize->rangemin[j]);
00755   optimize->nsaveds = 0;
00756   if (nthreads <= 1)
00757     optimize_sequential ();
00758   else
00759     {
00760       for (i = 0; i < nthreads; ++i)
00761         {
00762           data[i].thread = i;
00763           thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00764         }
00765       for (i = 0; i < nthreads; ++i)
00766         g_thread_join (thread[i]);
00767     }
00768 #if HAVE_MPI
00769   // Communicating tasks results
00770   optimize_synchronise ();
00771 #endif
00772 #if DEBUG_OPTIMIZE
00773   fprintf (stderr, "optimize_MonteCarlo: end\n");
00774 #endif
00775 }
00776
00786 void
00787 optimize_best_direction (unsigned int simulation, double value)
00788 {
00789 #if DEBUG_OPTIMIZE
00790   fprintf (stderr, "optimize_best_direction: start\n");
00791   fprintf (stderr,
00792            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00793            simulation, value, optimize->error_best[0]);
00794 #endif
00795   if (value < optimize->error_best[0])
00796     {
00797       optimize->error_best[0] = value;
00798       optimize->simulation_best[0] = simulation;
00799 #if DEBUG_OPTIMIZE
00800       fprintf (stderr,
00801                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00802                simulation, value);
00803 #endif
00804     }
00805 #if DEBUG_OPTIMIZE
00806   fprintf (stderr, "optimize_best_direction: end\n");
00807 #endif
00808 }
00809
00816 void
00817 optimize_direction_sequential (unsigned int simulation)
00818 {
00819   unsigned int i, j;
00820   double e;
00821 #if DEBUG_OPTIMIZE
00822   fprintf (stderr, "optimize_direction_sequential: start\n");
00823   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00824            "nend_direction=%u\n",
00825            optimize->nstart_direction, optimize->nend_direction);
00826 #endif
00827   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829       j = simulation + i;
00830       e = optimize_norm (j);
00831       optimize_best_direction (j, e);
00832       optimize_save_variables (j, e);
00833       if (e < optimize->threshold)
00834         {
00835           optimize->stop = 1;
00836           break;
00837         }
00838 #if DEBUG_OPTIMIZE
00839       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00840 #endif
00841     }
00842 #if DEBUG_OPTIMIZE
00843   fprintf (stderr, "optimize_direction_sequential: end\n");
00844 #endif
00845 }
00846
```

```
00854 void *
00855 optimize_direction_thread (ParallelData * data)
00856 {
00857   unsigned int i, thread;
00858   double e;
00859 #if DEBUG_OPTIMIZE
00860   fprintf (stderr, "optimize_direction_thread: start\n");
00861 #endif
00862   thread = data->thread;
00863 #if DEBUG_OPTIMIZE
00864   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865            thread,
00866            optimize->thread_direction[thread],
00867            optimize->thread_direction[thread + 1]);
00868 #endif
00869   for (i = optimize->thread_direction[thread];
00870        i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872       e = optimize_norm (i);
00873       g_mutex_lock (mutex);
00874       optimize_best_direction (i, e);
00875       optimize_save_variables (i, e);
00876       if (e < optimize->threshold)
00877         optimize->stop = 1;
00878       g_mutex_unlock (mutex);
00879       if (optimize->stop)
00880         break;
00881 #if DEBUG_OPTIMIZE
00882       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886   fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888   g_thread_exit (NULL);
00889   return NULL;
00890 }
00891
00901 double
00902 optimize_estimate_direction_random (unsigned int variable,
00903                                     unsigned int estimate)
00904 {
00905   double x;
00906 #if DEBUG_OPTIMIZE
00907   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908 #endif
00909   x = optimize->direction[variable]
00910     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00911 #if DEBUG_OPTIMIZE
00912   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00913            variable, x);
00914   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915 #endif
00916   return x;
00917 }
00918
00928 double
00929 optimize_estimate_direction_coordinates (unsigned int variable,
00930                                          unsigned int estimate)
00931 {
00932   double x;
00933 #if DEBUG_OPTIMIZE
00934   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00935 #endif
00936   x = optimize->direction[variable];
00937   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00938     {
00939       if (estimate & 1)
00940         x += optimize->step[variable];
00941       else
00942         x -= optimize->step[variable];
00943     }
00944 #if DEBUG_OPTIMIZE
00945   fprintf (stderr,
00946            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00947            variable, x);
00948   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00949 #endif
00950   return x;
00951 }
00952
00959 void
00960 optimize_step_direction (unsigned int simulation)
00961 {
00962   GThread *thread[nthreads_direction];
00963   ParallelData data[nthreads_direction];
00964   unsigned int i, j, k, b;
```

```
00965 #if DEBUG_OPTIMIZE
00966   fprintf (stderr, "optimize_step_direction: start\n");
00967 #endif
00968   for (i = 0; i < optimize->nestimates; ++i)
00969     {
00970       k = (simulation + i) * optimize->nvariables;
00971       b = optimize->simulation_best[0] * optimize->nvariables;
00972 #if DEBUG_OPTIMIZE
00973       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00974                simulation + i, optimize->simulation_best[0]);
00975 #endif
00976       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00977         {
00978 #if DEBUG_OPTIMIZE
00979           fprintf (stderr,
00980                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00981                    i, j, optimize->value[b]);
00982 #endif
00983           optimize->value[k]
00984             = optimize->value[b] + optimize_estimate_direction (j, i);
00985           optimize->value[k] = fmin (fmax (optimize->value[k],
00986                                            optimize->rangeminabs[j]),
00987                                      optimize->rangemaxabs[j]);
00988 #if DEBUG_OPTIMIZE
00989           fprintf (stderr,
00990                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00991                    i, j, optimize->value[k]);
00992 #endif
00993         }
00994     }
00995   if (nthreads_direction == 1)
00996     optimize_direction_sequential (simulation);
00997   else
00998     {
00999       for (i = 0; i <= nthreads_direction; ++i)
01000         {
01001           optimize->thread_direction[i]
01002             = simulation + optimize->nstart_direction
01003             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01004             / nthreads_direction;
01005 #if DEBUG_OPTIMIZE
01006           fprintf (stderr,
01007                    "optimize_step_direction: i=%u thread_direction=%u\n",
01008                    i, optimize->thread_direction[i]);
01009 #endif
01010         }
01011       for (i = 0; i < nthreads_direction; ++i)
01012         {
01013           data[i].thread = i;
01014           thread[i] = g_thread_new
01015             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01016         }
01017       for (i = 0; i < nthreads_direction; ++i)
01018         g_thread_join (thread[i]);
01019     }
01020 #if DEBUG_OPTIMIZE
01021   fprintf (stderr, "optimize_step_direction: end\n");
01022 #endif
01023 }
01024
01029 void
01030 optimize_direction ()
01031 {
01032   unsigned int i, j, k, b, s, adjust;
01033 #if DEBUG_OPTIMIZE
01034   fprintf (stderr, "optimize_direction: start\n");
01035 #endif
01036   for (i = 0; i < optimize->nvariables; ++i)
01037     optimize->direction[i] = 0.;
01038   b = optimize->simulation_best[0] * optimize->nvariables;
01039   s = optimize->nsimulations;
01040   adjust = 1;
01041   for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01042     {
01043 #if DEBUG_OPTIMIZE
01044       fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01045                i, optimize->simulation_best[0]);
01046 #endif
01047       optimize_step_direction (s);
01048       k = optimize->simulation_best[0] * optimize->nvariables;
01049 #if DEBUG_OPTIMIZE
01050       fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01051                i, optimize->simulation_best[0]);
01052 #endif
01053       if (k == b)
01054         {
```

```
01055           if (adjust)
01056             for (j = 0; j < optimize->nvariables; ++j)
01057               optimize->step[j] *= 0.5;
01058           for (j = 0; j < optimize->nvariables; ++j)
01059             optimize->direction[j] = 0.;
01060           adjust = 1;
01061         }
01062       else
01063         {
01064           for (j = 0; j < optimize->nvariables; ++j)
01065             {
01066 #if DEBUG_OPTIMIZE
01067               fprintf (stderr,
01068                        "optimize_direction: best%u=%.14le old%u=%.14le\n",
01069                        j, optimize->value[k + j], j, optimize->value[b + j]);
01070 #endif
01071               optimize->direction[j]
01072                 = (1. - optimize->relaxation) * optimize->direction[j]
01073                 + optimize->relaxation
01074                 * (optimize->value[k + j] - optimize->value[b + j]);
01075 #if DEBUG_OPTIMIZE
01076               fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01077                        j, optimize->direction[j]);
01078 #endif
01079             }
01080           adjust = 0;
01081         }
01082     }
01083 #if DEBUG_OPTIMIZE
01084   fprintf (stderr, "optimize_direction: end\n");
01085 #endif
01086 }
01087
01095 double
01096 optimize_genetic_objective (Entity * entity)
01097 {
01098   unsigned int j;
01099   double objective;
01100   char buffer[64];
01101 #if DEBUG_OPTIMIZE
01102   fprintf (stderr, "optimize_genetic_objective: start\n");
01103 #endif
01104   for (j = 0; j < optimize->nvariables; ++j)
01105     {
01106       optimize->value[entity->id * optimize->nvariables + j]
01107         = genetic_get_variable (entity, optimize->genetic_variable + j);
01108     }
01109   objective = optimize_norm (entity->id);
01110   g_mutex_lock (mutex);
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01114       fprintf (optimize->file_variables, buffer,
01115                genetic_get_variable (entity, optimize->genetic_variable + j));
01116     }
01117   fprintf (optimize->file_variables, "%.14le\n", objective);
01118   g_mutex_unlock (mutex);
01119 #if DEBUG_OPTIMIZE
01120   fprintf (stderr, "optimize_genetic_objective: end\n");
01121 #endif
01122   return objective;
01123 }
01124
01129 void
01130 optimize_genetic ()
01131 {
01132   char *best_genome;
01133   double best_objective, *best_variable;
01134 #if DEBUG_OPTIMIZE
01135   fprintf (stderr, "optimize_genetic: start\n");
01136   fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01137            nthreads);
01138   fprintf (stderr,
01139            "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01140            optimize->nvariables, optimize->nsimulations, optimize->
      niterations);
01141   fprintf (stderr,
01142            "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01143            optimize->mutation_ratio, optimize->reproduction_ratio,
01144            optimize->adaptation_ratio);
01145 #endif
01146   genetic_algorithm_default (optimize->nvariables,
01147                              optimize->genetic_variable,
01148                              optimize->nsimulations,
01149                              optimize->niterations,
01150                              optimize->mutation_ratio,
01151                              optimize->reproduction_ratio,
```

```
01152                               optimize->adaptation_ratio,
01153                               optimize->seed,
01154                               optimize->threshold,
01155                               &optimize_genetic_objective,
01156                               &best_genome, &best_variable, &best_objective);
01157 #if DEBUG_OPTIMIZE
01158   fprintf (stderr, "optimize_genetic: the best\n");
01159 #endif
01160   optimize->error_old = (double *) g_malloc (sizeof (double));
01161   optimize->value_old
01162     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01163   optimize->error_old[0] = best_objective;
01164   memcpy (optimize->value_old, best_variable,
01165           optimize->nvariables * sizeof (double));
01166   g_free (best_genome);
01167   g_free (best_variable);
01168   optimize_print ();
01169 #if DEBUG_OPTIMIZE
01170   fprintf (stderr, "optimize_genetic: end\n");
01171 #endif
01172 }
01173
01178 void
01179 optimize_save_old ()
01180 {
01181   unsigned int i, j;
01182 #if DEBUG_OPTIMIZE
01183   fprintf (stderr, "optimize_save_old: start\n");
01184   fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01185 #endif
01186   memcpy (optimize->error_old, optimize->error_best,
01187           optimize->nbest * sizeof (double));
01188   for (i = 0; i < optimize->nbest; ++i)
01189     {
01190       j = optimize->simulation_best[i];
01191 #if DEBUG_OPTIMIZE
01192       fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01193 #endif
01194       memcpy (optimize->value_old + i * optimize->nvariables,
01195               optimize->value + j * optimize->nvariables,
01196               optimize->nvariables * sizeof (double));
01197     }
01198 #if DEBUG_OPTIMIZE
01199   for (i = 0; i < optimize->nvariables; ++i)
01200     fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01201              i, optimize->value_old[i]);
01202   fprintf (stderr, "optimize_save_old: end\n");
01203 #endif
01204 }
01205
01211 void
01212 optimize_merge_old ()
01213 {
01214   unsigned int i, j, k;
01215   double v[optimize->nbest * optimize->nvariables], e[optimize->
      nbest],
01216     *enew, *eold;
01217 #if DEBUG_OPTIMIZE
01218   fprintf (stderr, "optimize_merge_old: start\n");
01219 #endif
01220   enew = optimize->error_best;
01221   eold = optimize->error_old;
01222   i = j = k = 0;
01223   do
01224     {
01225       if (*enew < *eold)
01226         {
01227           memcpy (v + k * optimize->nvariables,
01228                   optimize->value
01229                   + optimize->simulation_best[i] * optimize->
      nvariables,
01230                   optimize->nvariables * sizeof (double));
01231           e[k] = *enew;
01232           ++k;
01233           ++enew;
01234           ++i;
01235         }
01236       else
01237         {
01238           memcpy (v + k * optimize->nvariables,
01239                   optimize->value_old + j * optimize->nvariables,
01240                   optimize->nvariables * sizeof (double));
01241           e[k] = *eold;
01242           ++k;
01243           ++eold;
01244           ++j;
01245         }
```

```
01246     }
01247   while (k < optimize->nbest);
01248   memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01249   memcpy (optimize->error_old, e, k * sizeof (double));
01250 #if DEBUG_OPTIMIZE
01251   fprintf (stderr, "optimize_merge_old: end\n");
01252 #endif
01253 }
01254
01260 void
01261 optimize_refine ()
01262 {
01263   unsigned int i, j;
01264   double d;
01265 #if HAVE_MPI
01266   MPI_Status mpi_stat;
01267 #endif
01268 #if DEBUG_OPTIMIZE
01269   fprintf (stderr, "optimize_refine: start\n");
01270 #endif
01271 #if HAVE_MPI
01272   if (!optimize->mpi_rank)
01273     {
01274 #endif
01275       for (j = 0; j < optimize->nvariables; ++j)
01276         {
01277           optimize->rangemin[j] = optimize->rangemax[j]
01278             = optimize->value_old[j];
01279         }
01280       for (i = 0; ++i < optimize->nbest;)
01281         {
01282           for (j = 0; j < optimize->nvariables; ++j)
01283             {
01284               optimize->rangemin[j]
01285                 = fmin (optimize->rangemin[j],
01286                         optimize->value_old[i * optimize->nvariables + j]);
01287               optimize->rangemax[j]
01288                 = fmax (optimize->rangemax[j],
01289                         optimize->value_old[i * optimize->nvariables + j]);
01290             }
01291         }
01292       for (j = 0; j < optimize->nvariables; ++j)
01293         {
01294           d = optimize->tolerance
01295             * (optimize->rangemax[j] - optimize->rangemin[j]);
01296           switch (optimize->algorithm)
01297             {
01298             case ALGORITHM_MONTE_CARLO:
01299               d *= 0.5;
01300               break;
01301             default:
01302               if (optimize->nsweeps[j] > 1)
01303                 d /= optimize->nsweeps[j] - 1;
01304               else
01305                 d = 0.;
01306             }
01307           optimize->rangemin[j] -= d;
01308           optimize->rangemin[j]
01309             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01310           optimize->rangemax[j] += d;
01311           optimize->rangemax[j]
01312             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01313           printf ("%s min=%lg max=%lg\n", optimize->label[j],
01314                   optimize->rangemin[j], optimize->rangemax[j]);
01315           fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01316                    optimize->label[j], optimize->rangemin[j],
01317                    optimize->rangemax[j]);
01318         }
01319 #if HAVE_MPI
01320       for (i = 1; i < ntasks; ++i)
01321         {
01322           MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01323                     1, MPI_COMM_WORLD);
01324           MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01325                     1, MPI_COMM_WORLD);
01326         }
01327     }
01328   else
01329     {
01330       MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01331                 MPI_COMM_WORLD, &mpi_stat);
01332       MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01333                 MPI_COMM_WORLD, &mpi_stat);
01334     }
01335 #endif
01336 #if DEBUG_OPTIMIZE
01337   fprintf (stderr, "optimize_refine: end\n");
```

```
01338 #endif
01339 }
01340
01345 void
01346 optimize_step ()
01347 {
01348 #if DEBUG_OPTIMIZE
01349   fprintf (stderr, "optimize_step: start\n");
01350 #endif
01351   optimize_algorithm ();
01352   if (optimize->nsteps)
01353     optimize_direction ();
01354 #if DEBUG_OPTIMIZE
01355   fprintf (stderr, "optimize_step: end\n");
01356 #endif
01357 }
01358
01363 void
01364 optimize_iterate ()
01365 {
01366   unsigned int i;
01367 #if DEBUG_OPTIMIZE
01368   fprintf (stderr, "optimize_iterate: start\n");
01369 #endif
01370   optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01371   optimize->value_old = (double *)
01372     g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01373   optimize_step ();
01374   optimize_save_old ();
01375   optimize_refine ();
01376   optimize_print ();
01377   for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01378     {
01379       optimize_step ();
01380       optimize_merge_old ();
01381       optimize_refine ();
01382       optimize_print ();
01383     }
01384 #if DEBUG_OPTIMIZE
01385   fprintf (stderr, "optimize_iterate: end\n");
01386 #endif
01387 }
01388
01393 void
01394 optimize_free ()
01395 {
01396   unsigned int i, j;
01397 #if DEBUG_OPTIMIZE
01398   fprintf (stderr, "optimize_free: start\n");
01399 #endif
01400   for (j = 0; j < optimize->ninputs; ++j)
01401     {
01402       for (i = 0; i < optimize->nexperiments; ++i)
01403         g_mapped_file_unref (optimize->file[j][i]);
01404       g_free (optimize->file[j]);
01405     }
01406   g_free (optimize->error_old);
01407   g_free (optimize->value_old);
01408   g_free (optimize->value);
01409   g_free (optimize->genetic_variable);
01410 #if DEBUG_OPTIMIZE
01411   fprintf (stderr, "optimize_free: end\n");
01412 #endif
01413 }
01414
01419 void
01420 optimize_open ()
01421 {
01422   GTimeZone *tz;
01423   GDateTime *t0, *t;
01424   unsigned int i, j;
01425
01426 #if DEBUG_OPTIMIZE
01427   char *buffer;
01428   fprintf (stderr, "optimize_open: start\n");
01429 #endif
01430
01431   // Getting initial time
01432 #if DEBUG_OPTIMIZE
01433   fprintf (stderr, "optimize_open: getting initial time\n");
01434 #endif
01435   tz = g_time_zone_new_utc ();
01436   t0 = g_date_time_new_now (tz);
01437
01438   // Obtaining and initing the pseudo-random numbers generator seed
01439 #if DEBUG_OPTIMIZE
01440   fprintf (stderr, "optimize_open: getting initial seed\n");
```

```
01441 #endif
01442   if (optimize->seed == DEFAULT_RANDOM_SEED)
01443     optimize->seed = input->seed;
01444   gsl_rng_set (optimize->rng, optimize->seed);
01445
01446   // Replacing the working directory
01447 #if DEBUG_OPTIMIZE
01448   fprintf (stderr, "optimize_open: replacing the working directory\n");
01449 #endif
01450   g_chdir (input->directory);
01451
01452   // Getting results file names
01453   optimize->result = input->result;
01454   optimize->variables = input->variables;
01455
01456   // Obtaining the simulator file
01457   optimize->simulator = input->simulator;
01458
01459   // Obtaining the evaluator file
01460   optimize->evaluator = input->evaluator;
01461
01462   // Reading the algorithm
01463   optimize->algorithm = input->algorithm;
01464   switch (optimize->algorithm)
01465     {
01466     case ALGORITHM_MONTE_CARLO:
01467       optimize_algorithm = optimize_MonteCarlo;
01468       break;
01469     case ALGORITHM_SWEEP:
01470       optimize_algorithm = optimize_sweep;
01471       break;
01472     default:
01473       optimize_algorithm = optimize_genetic;
01474       optimize->mutation_ratio = input->mutation_ratio;
01475       optimize->reproduction_ratio = input->
      reproduction_ratio;
01476       optimize->adaptation_ratio = input->adaptation_ratio;
01477     }
01478   optimize->nvariables = input->nvariables;
01479   optimize->nsimulations = input->nsimulations;
01480   optimize->niterations = input->niterations;
01481   optimize->nbest = input->nbest;
01482   optimize->tolerance = input->tolerance;
01483   optimize->nsteps = input->nsteps;
01484   optimize->nestimates = 0;
01485   optimize->threshold = input->threshold;
01486   optimize->stop = 0;
01487   if (input->nsteps)
01488     {
01489       optimize->relaxation = input->relaxation;
01490       switch (input->direction)
01491         {
01492         case DIRECTION_METHOD_COORDINATES:
01493           optimize->nestimates = 2 * optimize->nvariables;
01494           optimize_estimate_direction =
      optimize_estimate_direction_coordinates;
01495           break;
01496         default:
01497           optimize->nestimates = input->nestimates;
01498           optimize_estimate_direction =
      optimize_estimate_direction_random;
01499         }
01500     }
01501
01502 #if DEBUG_OPTIMIZE
01503   fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01504 #endif
01505   optimize->simulation_best
01506     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01507   optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01508
01509   // Reading the experimental data
01510 #if DEBUG_OPTIMIZE
01511   buffer = g_get_current_dir ();
01512   fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01513   g_free (buffer);
01514 #endif
01515   optimize->nexperiments = input->nexperiments;
01516   optimize->ninputs = input->experiment->ninputs;
01517   optimize->experiment
01518     = (char **) alloca (input->nexperiments * sizeof (char *));
01519   optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01520   for (i = 0; i < input->experiment->ninputs; ++i)
01521     optimize->file[i] = (GMappedFile **)
01522       g_malloc (input->nexperiments * sizeof (GMappedFile *));
01523   for (i = 0; i < input->nexperiments; ++i)
01524     {
```

```
01525 #if DEBUG_OPTIMIZE
01526       fprintf (stderr, "optimize_open: i=%u\n", i);
01527 #endif
01528       optimize->experiment[i] = input->experiment[i].
      name;
01529       optimize->weight[i] = input->experiment[i].weight;
01530 #if DEBUG_OPTIMIZE
01531       fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01532                optimize->experiment[i], optimize->weight[i]);
01533 #endif
01534       for (j = 0; j < input->experiment->ninputs; ++j)
01535         {
01536 #if DEBUG_OPTIMIZE
01537           fprintf (stderr, "optimize_open: template%u\n", j + 1);
01538 #endif
01539           optimize->file[j][i]
01540             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01541         }
01542     }
01543
01544   // Reading the variables data
01545 #if DEBUG_OPTIMIZE
01546   fprintf (stderr, "optimize_open: reading variables\n");
01547 #endif
01548   optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01549   j = input->nvariables * sizeof (double);
01550   optimize->rangemin = (double *) alloca (j);
01551   optimize->rangeminabs = (double *) alloca (j);
01552   optimize->rangemax = (double *) alloca (j);
01553   optimize->rangemaxabs = (double *) alloca (j);
01554   optimize->step = (double *) alloca (j);
01555   j = input->nvariables * sizeof (unsigned int);
01556   optimize->precision = (unsigned int *) alloca (j);
01557   optimize->nsweeps = (unsigned int *) alloca (j);
01558   optimize->nbits = (unsigned int *) alloca (j);
01559   for (i = 0; i < input->nvariables; ++i)
01560     {
01561       optimize->label[i] = input->variable[i].name;
01562       optimize->rangemin[i] = input->variable[i].rangemin;
01563       optimize->rangeminabs[i] = input->variable[i].
      rangeminabs;
01564       optimize->rangemax[i] = input->variable[i].rangemax;
01565       optimize->rangemaxabs[i] = input->variable[i].
      rangemaxabs;
01566       optimize->precision[i] = input->variable[i].
      precision;
01567       optimize->step[i] = input->variable[i].step;
01568       optimize->nsweeps[i] = input->variable[i].nsweeps;
01569       optimize->nbits[i] = input->variable[i].nbits;
01570     }
01571   if (input->algorithm == ALGORITHM_SWEEP)
01572     {
01573       optimize->nsimulations = 1;
01574       for (i = 0; i < input->nvariables; ++i)
01575         {
01576           if (input->algorithm == ALGORITHM_SWEEP)
01577             {
01578               optimize->nsimulations *= optimize->nsweeps[i];
01579 #if DEBUG_OPTIMIZE
01580               fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01581                        optimize->nsweeps[i], optimize->nsimulations);
01582 #endif
01583             }
01584         }
01585     }
01586   if (optimize->nsteps)
01587     optimize->direction
01588       = (double *) alloca (optimize->nvariables * sizeof (double));
01589
01590   // Setting error norm
01591   switch (input->norm)
01592     {
01593     case ERROR_NORM_EUCLIDIAN:
01594       optimize_norm = optimize_norm_euclidian;
01595       break;
01596     case ERROR_NORM_MAXIMUM:
01597       optimize_norm = optimize_norm_maximum;
01598       break;
01599     case ERROR_NORM_P:
01600       optimize_norm = optimize_norm_p;
01601       optimize->p = input->p;
01602       break;
01603     default:
01604       optimize_norm = optimize_norm_taxicab;
01605     }
01606
01607   // Allocating values
```

```
01608 #if DEBUG_OPTIMIZE
01609   fprintf (stderr, "optimize_open: allocating variables\n");
01610   fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01611            optimize->nvariables, optimize->algorithm);
01612 #endif
01613   optimize->genetic_variable = NULL;
01614   if (optimize->algorithm == ALGORITHM_GENETIC)
01615     {
01616       optimize->genetic_variable = (GeneticVariable *)
01617         g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01618       for (i = 0; i < optimize->nvariables; ++i)
01619         {
01620 #if DEBUG_OPTIMIZE
01621           fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01622                    i, optimize->rangemin[i], optimize->rangemax[i],
01623                    optimize->nbits[i]);
01624 #endif
01625           optimize->genetic_variable[i].minimum = optimize->
      rangemin[i];
01626           optimize->genetic_variable[i].maximum = optimize->
      rangemax[i];
01627           optimize->genetic_variable[i].nbits = optimize->nbits[i];
01628         }
01629     }
01630 #if DEBUG_OPTIMIZE
01631   fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01632            optimize->nvariables, optimize->nsimulations);
01633 #endif
01634   optimize->value = (double *)
01635     g_malloc ((optimize->nsimulations
01636                + optimize->nestimates * optimize->nsteps)
01637               * optimize->nvariables * sizeof (double));
01638
01639   // Calculating simulations to perform for each task
01640 #if HAVE_MPI
01641 #if DEBUG_OPTIMIZE
01642   fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01643            optimize->mpi_rank, ntasks);
01644 #endif
01645   optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
      ntasks;
01646   optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
      ntasks;
01647   if (optimize->nsteps)
01648     {
01649       optimize->nstart_direction
01650         = optimize->mpi_rank * optimize->nestimates / ntasks;
01651       optimize->nend_direction
01652         = (1 + optimize->mpi_rank) * optimize->nestimates /
      ntasks;
01653     }
01654 #else
01655   optimize->nstart = 0;
01656   optimize->nend = optimize->nsimulations;
01657   if (optimize->nsteps)
01658     {
01659       optimize->nstart_direction = 0;
01660       optimize->nend_direction = optimize->nestimates;
01661     }
01662 #endif
01663 #if DEBUG_OPTIMIZE
01664   fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01665            optimize->nend);
01666 #endif
01667
01668   // Calculating simulations to perform for each thread
01669   optimize->thread
01670     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01671   for (i = 0; i <= nthreads; ++i)
01672     {
01673       optimize->thread[i] = optimize->nstart
01674         + i * (optimize->nend - optimize->nstart) / nthreads;
01675 #if DEBUG_OPTIMIZE
01676       fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01677                optimize->thread[i]);
01678 #endif
01679     }
01680   if (optimize->nsteps)
01681     optimize->thread_direction = (unsigned int *)
01682       alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01683
01684   // Opening result files
01685   optimize->file_result = g_fopen (optimize->result, "w");
01686   optimize->file_variables = g_fopen (optimize->variables, "w");
01687
01688   // Performing the algorithm
01689   switch (optimize->algorithm)
```

```
01690     {
01691       // Genetic algorithm
01692     case ALGORITHM_GENETIC:
01693       optimize_genetic ();
01694       break;
01695
01696       // Iterative algorithm
01697     default:
01698       optimize_iterate ();
01699     }
01700
01701   // Getting calculation time
01702   t = g_date_time_new_now (tz);
01703   optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01704   g_date_time_unref (t);
01705   g_date_time_unref (t0);
01706   g_time_zone_unref (tz);
01707   printf ("%s = %.6lg s\n",
01708           gettext ("Calculation time"), optimize->calculation_time);
01709   fprintf (optimize->file_result, "%s = %.6lg s\n",
01710            gettext ("Calculation time"), optimize->calculation_time);
01711
01712   // Closing result files
01713   fclose (optimize->file_variables);
01714   fclose (optimize->file_result);
01715
01716 #if DEBUG_OPTIMIZE
01717   fprintf (stderr, "optimize_open: end\n");
01718 #endif
01719 }
```

5.19 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.

5.19.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Definition in file optimize.h.

5.19.2 Function Documentation

5.19.2.1 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 462 of file optimize.c.

```
00463 {
00464   unsigned int i, j;
00465   double e;
00466 #if DEBUG_OPTIMIZE
00467   fprintf (stderr, "optimize_best: start\n");
00468   fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00469            optimize->nsaveds, optimize->nbest);
00470 #endif
00471   if (optimize->nsaveds < optimize->nbest
00472       || value < optimize->error_best[optimize->nsaveds - 1])
00473     {
00474       if (optimize->nsaveds < optimize->nbest)
00475         ++optimize->nsaveds;
```

```
00476       optimize->error_best[optimize->nsaveds - 1] = value;
00477       optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00478       for (i = optimize->nsaveds; --i;)
00479         {
00480           if (optimize->error_best[i] < optimize->
      error_best[i - 1])
00481             {
00482               j = optimize->simulation_best[i];
00483               e = optimize->error_best[i];
00484               optimize->simulation_best[i] = optimize->
      simulation_best[i - 1];
00485               optimize->error_best[i] = optimize->
      error_best[i - 1];
00486               optimize->simulation_best[i - 1] = j;
00487               optimize->error_best[i - 1] = e;
00488             }
00489           else
00490             break;
00491         }
00492     }
00493 #if DEBUG_OPTIMIZE
00494   fprintf (stderr, "optimize_best: end\n");
00495 #endif
00496 }
```

5.19.2.2 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 787 of file optimize.c.

```
00788 {
00789 #if DEBUG_OPTIMIZE
00790   fprintf (stderr, "optimize_best_direction: start\n");
00791   fprintf (stderr,
00792            "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00793            simulation, value, optimize->error_best[0]);
00794 #endif
00795   if (value < optimize->error_best[0])
00796     {
00797       optimize->error_best[0] = value;
00798       optimize->simulation_best[0] = simulation;
00799 #if DEBUG_OPTIMIZE
00800       fprintf (stderr,
00801                "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00802                simulation, value);
00803 #endif
00804     }
00805 #if DEBUG_OPTIMIZE
00806   fprintf (stderr, "optimize_best_direction: end\n");
00807 #endif
00808 }
```

5.19.2.3 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

simulation	Simulation number.

Definition at line 817 of file optimize.c.

```
00818 {
00819   unsigned int i, j;
00820   double e;
00821 #if DEBUG_OPTIMIZE
00822   fprintf (stderr, "optimize_direction_sequential: start\n");
00823   fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
```

```
00824            "nend_direction=%u\n",
00825            optimize->nstart_direction, optimize->
     nend_direction);
00826 #endif
00827   for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829       j = simulation + i;
00830       e = optimize_norm (j);
00831       optimize_best_direction (j, e);
00832       optimize_save_variables (j, e);
00833       if (e < optimize->threshold)
00834         {
00835           optimize->stop = 1;
00836           break;
00837         }
00838 #if DEBUG_OPTIMIZE
00839       fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00840 #endif
00841     }
00842 #if DEBUG_OPTIMIZE
00843   fprintf (stderr, "optimize_direction_sequential: end\n");
00844 #endif
00845 }
```

Here is the call graph for this function:

5.19.2.4 void* optimize_direction_thread (ParallelData * *data*)

Function to estimate the direction search on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 855 of file optimize.c.

```
00856 {
00857   unsigned int i, thread;
00858   double e;
00859 #if DEBUG_OPTIMIZE
00860   fprintf (stderr, "optimize_direction_thread: start\n");
00861 #endif
00862   thread = data->thread;
00863 #if DEBUG_OPTIMIZE
00864   fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865            thread,
00866            optimize->thread_direction[thread],
00867            optimize->thread_direction[thread + 1]);
00868 #endif
00869   for (i = optimize->thread_direction[thread];
00870        i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872       e = optimize_norm (i);
00873       g_mutex_lock (mutex);
00874       optimize_best_direction (i, e);
00875       optimize_save_variables (i, e);
00876       if (e < optimize->threshold)
00877         optimize->stop = 1;
00878       g_mutex_unlock (mutex);
00879       if (optimize->stop)
00880         break;
00881 #if DEBUG_OPTIMIZE
00882       fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886   fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888   g_thread_exit (NULL);
00889   return NULL;
00890 }
```

Here is the call graph for this function:

5.19.2.5 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 929 of file optimize.c.

```
00931 {
00932   double x;
00933 #if DEBUG_OPTIMIZE
00934   fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00935 #endif
00936   x = optimize->direction[variable];
00937   if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00938     {
00939       if (estimate & 1)
00940         x += optimize->step[variable];
00941       else
00942         x -= optimize->step[variable];
00943     }
00944 #if DEBUG_OPTIMIZE
00945   fprintf (stderr,
00946            "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00947            variable, x);
00948   fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00949 #endif
00950   return x;
00951 }
```

5.19.2.6 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 902 of file optimize.c.

```
00904 {
00905   double x;
00906 #if DEBUG_OPTIMIZE
00907   fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908 #endif
00909   x = optimize->direction[variable]
00910     + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
      step[variable];
00911 #if DEBUG_OPTIMIZE
00912   fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00913            variable, x);
00914   fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915 #endif
00916   return x;
00917 }
```

5.19.2.7 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 1096 of file optimize.c.

```
01097 {
01098   unsigned int j;
01099   double objective;
01100   char buffer[64];
01101 #if DEBUG_OPTIMIZE
01102   fprintf (stderr, "optimize_genetic_objective: start\n");
01103 #endif
01104   for (j = 0; j < optimize->nvariables; ++j)
01105     {
01106       optimize->value[entity->id * optimize->nvariables + j]
01107         = genetic_get_variable (entity, optimize->genetic_variable + j);
01108     }
01109   objective = optimize_norm (entity->id);
01110   g_mutex_lock (mutex);
01111   for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113       snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01114       fprintf (optimize->file_variables, buffer,
01115                genetic_get_variable (entity, optimize->genetic_variable + j));
01116     }
01117   fprintf (optimize->file_variables, "%.14le\n", objective);
01118   g_mutex_unlock (mutex);
01119 #if DEBUG_OPTIMIZE
01120   fprintf (stderr, "optimize_genetic_objective: end\n");
01121 #endif
01122   return objective;
01123 }
```

5.19.2.8 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 103 of file optimize.c.

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

```
00144         {
00145           length = strlen (buffer3);
00146           buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                              optimize->label[i], 0, NULL);
00148           g_free (buffer3);
00149         }
00150       g_regex_unref (regex);
00151       length = strlen (buffer2);
00152       snprintf (buffer, 32, "@value%u@", i + 1);
00153       regex = g_regex_new (buffer, 0, 0, NULL);
00154       snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->
      nvariables + i]);
00156
00157 #if DEBUG_OPTIMIZE
00158       fprintf (stderr, "optimize_input: value=%s\n", value);
00159 #endif
00160       buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00161                                          0, NULL);
00162       g_free (buffer2);
00163       g_regex_unref (regex);
00164     }
00165
00166   // Saving input file
00167   fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168   g_free (buffer3);
00169   fclose (file);
00170
00171 optimize_input_end:
00172 #if DEBUG_OPTIMIZE
00173   fprintf (stderr, "optimize_input: end\n");
00174 #endif
00175   return;
00176 }
```

5.19.2.9 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 585 of file optimize.c.

```
00587 {
00588   unsigned int i, j, k, s[optimize->nbest];
00589   double e[optimize->nbest];
00590 #if DEBUG_OPTIMIZE
00591   fprintf (stderr, "optimize_merge: start\n");
00592 #endif
00593   i = j = k = 0;
00594   do
00595     {
00596       if (i == optimize->nsaveds)
00597         {
00598           s[k] = simulation_best[j];
00599           e[k] = error_best[j];
00600           ++j;
00601           ++k;
00602           if (j == nsaveds)
00603             break;
00604         }
00605       else if (j == nsaveds)
00606         {
00607           s[k] = optimize->simulation_best[i];
00608           e[k] = optimize->error_best[i];
00609           ++i;
00610           ++k;
00611           if (i == optimize->nsaveds)
00612             break;
00613         }
00614       else if (optimize->error_best[i] > error_best[j])
00615         {
00616           s[k] = simulation_best[j];
00617           e[k] = error_best[j];
00618           ++j;
00619           ++k;
```

```
00620         }
00621       else
00622         {
00623           s[k] = optimize->simulation_best[i];
00624           e[k] = optimize->error_best[i];
00625           ++i;
00626           ++k;
00627         }
00628     }
00629   while (k < optimize->nbest);
00630   optimize->nsaveds = k;
00631   memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632   memcpy (optimize->error_best, e, k * sizeof (double));
00633 #if DEBUG_OPTIMIZE
00634   fprintf (stderr, "optimize_merge: end\n");
00635 #endif
00636 }
```

5.19.2.10 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 295 of file optimize.c.

```
00296 {
00297   double e, ei;
00298   unsigned int i;
00299 #if DEBUG_OPTIMIZE
00300   fprintf (stderr, "optimize_norm_euclidian: start\n");
00301 #endif
00302   e = 0.;
00303   for (i = 0; i < optimize->nexperiments; ++i)
00304     {
00305       ei = optimize_parse (simulation, i);
00306       e += ei * ei;
00307     }
00308   e = sqrt (e);
00309 #if DEBUG_OPTIMIZE
00310   fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00311   fprintf (stderr, "optimize_norm_euclidian: end\n");
00312 #endif
00313   return e;
00314 }
```

Here is the call graph for this function:

5.19.2.11 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 324 of file optimize.c.

```
00325 {
00326   double e, ei;
00327   unsigned int i;
00328 #if DEBUG_OPTIMIZE
00329   fprintf (stderr, "optimize_norm_maximum: start\n");
00330 #endif
00331   e = 0.;
00332   for (i = 0; i < optimize->nexperiments; ++i)
00333     {
00334       ei = fabs (optimize_parse (simulation, i));
00335       e = fmax (e, ei);
00336     }
00337 #if DEBUG_OPTIMIZE
00338   fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00339   fprintf (stderr, "optimize_norm_maximum: end\n");
00340 #endif
00341   return e;
00342 }
```

Here is the call graph for this function:

5.19.2.12 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 352 of file optimize.c.

```
00353 {
00354   double e, ei;
00355   unsigned int i;
00356 #if DEBUG_OPTIMIZE
00357   fprintf (stderr, "optimize_norm_p: start\n");
00358 #endif
00359   e = 0.;
00360   for (i = 0; i < optimize->nexperiments; ++i)
00361     {
00362       ei = fabs (optimize_parse (simulation, i));
00363       e += pow (ei, optimize->p);
00364     }
00365   e = pow (e, 1. / optimize->p);
00366 #if DEBUG_OPTIMIZE
00367   fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00368   fprintf (stderr, "optimize_norm_p: end\n");
00369 #endif
00370   return e;
00371 }
```

Here is the call graph for this function:

5.19.2.13 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 381 of file optimize.c.

```
00382 {
00383   double e;
00384   unsigned int i;
00385 #if DEBUG_OPTIMIZE
00386   fprintf (stderr, "optimize_norm_taxicab: start\n");
00387 #endif
00388   e = 0.;
00389   for (i = 0; i < optimize->nexperiments; ++i)
00390     e += fabs (optimize_parse (simulation, i));
00391 #if DEBUG_OPTIMIZE
00392   fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00393   fprintf (stderr, "optimize_norm_taxicab: end\n");
00394 #endif
00395   return e;
00396 }
```

Here is the call graph for this function:

5.19.2.14 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 189 of file optimize.c.

```
00190 {
00191   unsigned int i;
00192   double e;
00193   char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00194     *buffer3, *buffer4;
00195   FILE *file_result;
00196
00197 #if DEBUG_OPTIMIZE
00198   fprintf (stderr, "optimize_parse: start\n");
00199   fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00200            experiment);
00201 #endif
00202
00203   // Opening input files
00204   for (i = 0; i < optimize->ninputs; ++i)
00205     {
00206       snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00207 #if DEBUG_OPTIMIZE
00208       fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00209 #endif
00210       optimize_input (simulation, &input[i][0], optimize->
      file[i][experiment]);
00211     }
00212   for (; i < MAX_NINPUTS; ++i)
00213     strcpy (&input[i][0], "");
00214 #if DEBUG_OPTIMIZE
00215   fprintf (stderr, "optimize_parse: parsing end\n");
00216 #endif
00217
00218   // Performing the simulation
00219   snprintf (output, 32, "output-%u-%u", simulation, experiment);
00220   buffer2 = g_path_get_dirname (optimize->simulator);
00221   buffer3 = g_path_get_basename (optimize->simulator);
00222   buffer4 = g_build_filename (buffer2, buffer3, NULL);
00223   snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00224             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00225             input[6], input[7], output);
00226   g_free (buffer4);
00227   g_free (buffer3);
00228   g_free (buffer2);
00229 #if DEBUG_OPTIMIZE
00230   fprintf (stderr, "optimize_parse: %s\n", buffer);
00231 #endif
00232   system (buffer);
```

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

Here is the call graph for this function:

5.19.2.15 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 434 of file optimize.c.

```
00435 {
00436   unsigned int i;
00437   char buffer[64];
00438 #if DEBUG_OPTIMIZE
00439   fprintf (stderr, "optimize_save_variables: start\n");
00440 #endif
00441   for (i = 0; i < optimize->nvariables; ++i)
00442     {
00443       snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444       fprintf (optimize->file_variables, buffer,
00445                optimize->value[simulation * optimize->
      nvariables + i]);
00446     }
```

```
00447   fprintf (optimize->file_variables, "%.14le\n", error);
00448 #if DEBUG_OPTIMIZE
00449   fprintf (stderr, "optimize_save_variables: end\n");
00450 #endif
00451 }
```

5.19.2.16 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

simulation	Simulation number.

Definition at line 960 of file optimize.c.

```
00961 {
00962   GThread *thread[nthreads_direction];
00963   ParallelData data[nthreads_direction];
00964   unsigned int i, j, k, b;
00965 #if DEBUG_OPTIMIZE
00966   fprintf (stderr, "optimize_step_direction: start\n");
00967 #endif
00968   for (i = 0; i < optimize->nestimates; ++i)
00969     {
00970       k = (simulation + i) * optimize->nvariables;
00971       b = optimize->simulation_best[0] * optimize->
      nvariables;
00972 #if DEBUG_OPTIMIZE
00973       fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00974                simulation + i, optimize->simulation_best[0]);
00975 #endif
00976       for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00977         {
00978 #if DEBUG_OPTIMIZE
00979           fprintf (stderr,
00980                    "optimize_step_direction: estimate=%u best%u=%.14le\n",
00981                    i, j, optimize->value[b]);
00982 #endif
00983           optimize->value[k]
00984             = optimize->value[b] + optimize_estimate_direction (j,
      i);
00985           optimize->value[k] = fmin (fmax (optimize->value[k],
00986                                            optimize->rangeminabs[j]),
00987                                      optimize->rangemaxabs[j]);
00988 #if DEBUG_OPTIMIZE
00989           fprintf (stderr,
00990                    "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00991                    i, j, optimize->value[k]);
00992 #endif
00993         }
00994     }
00995   if (nthreads_direction == 1)
00996     optimize_direction_sequential (simulation);
00997   else
00998     {
00999       for (i = 0; i <= nthreads_direction; ++i)
01000         {
01001           optimize->thread_direction[i]
01002             = simulation + optimize->nstart_direction
01003             + i * (optimize->nend_direction - optimize->
      nstart_direction)
01004             / nthreads_direction;
01005 #if DEBUG_OPTIMIZE
01006           fprintf (stderr,
01007                    "optimize_step_direction: i=%u thread_direction=%u\n",
01008                    i, optimize->thread_direction[i]);
01009 #endif
01010         }
01011       for (i = 0; i < nthreads_direction; ++i)
01012         {
01013           data[i].thread = i;
01014           thread[i] = g_thread_new
01015             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01016         }
01017       for (i = 0; i < nthreads_direction; ++i)
01018         g_thread_join (thread[i]);
01019     }
01020 #if DEBUG_OPTIMIZE
01021   fprintf (stderr, "optimize_step_direction: end\n");
01022 #endif
01023 }
```

Here is the call graph for this function:

5.19.2.17 void* optimize_thread (ParallelData * *data*)

Function to optimize on a thread.

Parameters

data	Function data.

Returns

NULL

Definition at line 539 of file optimize.c.

```
00540 {
00541   unsigned int i, thread;
00542   double e;
00543 #if DEBUG_OPTIMIZE
00544   fprintf (stderr, "optimize_thread: start\n");
00545 #endif
00546   thread = data->thread;
00547 #if DEBUG_OPTIMIZE
00548   fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00549            optimize->thread[thread], optimize->thread[thread + 1]);
00550 #endif
00551   for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00552     {
00553       e = optimize_norm (i);
00554       g_mutex_lock (mutex);
00555       optimize_best (i, e);
00556       optimize_save_variables (i, e);
00557       if (e < optimize->threshold)
00558         optimize->stop = 1;
00559       g_mutex_unlock (mutex);
00560       if (optimize->stop)
00561         break;
00562 #if DEBUG_OPTIMIZE
00563       fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00564 #endif
00565     }
00566 #if DEBUG_OPTIMIZE
00567   fprintf (stderr, "optimize_thread: end\n");
00568 #endif
00569   g_thread_exit (NULL);
00570   return NULL;
00571 }
```

Here is the call graph for this function:

5.20 optimize.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
```

```
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef OPTIMIZE__H
00039 #define OPTIMIZE__H 1
00040
00045 typedef struct
00046 {
00047   GMappedFile **file[MAX_NINPUTS];
00048   char **experiment;
00049   char **label;
00050   gsl_rng *rng;
00051   GeneticVariable *genetic_variable;
00053   FILE *file_result;
00054   FILE *file_variables;
00055   char *result;
00056   char *variables;
00057   char *simulator;
00058   char *evaluator;
00060   double *value;
00061   double *rangemin;
00062   double *rangemax;
00063   double *rangeminabs;
00064   double *rangemaxabs;
00065   double *error_best;
00066   double *weight;
00067   double *step;
00069   double *direction;
00070   double *value_old;
00072   double *error_old;
00074   unsigned int *precision;
00075   unsigned int *nsweeps;
00076   unsigned int *nbits;
00078   unsigned int *thread;
00080   unsigned int *thread_direction;
00083   unsigned int *simulation_best;
00084   double tolerance;
00085   double mutation_ratio;
00086   double reproduction_ratio;
00087   double adaptation_ratio;
00088   double relaxation;
00089   double calculation_time;
00090   double p;
00091   double threshold;
00092   unsigned long int seed;
00094   unsigned int nvariables;
00095   unsigned int nexperiments;
00096   unsigned int ninputs;
00097   unsigned int nsimulations;
00098   unsigned int nsteps;
00100   unsigned int nestimates;
00102   unsigned int algorithm;
00103   unsigned int nstart;
00104   unsigned int nend;
00105   unsigned int nstart_direction;
00107   unsigned int nend_direction;
00109   unsigned int niterations;
00110   unsigned int nbest;
00111   unsigned int nsaveds;
00112   unsigned int stop;
00113 #if HAVE_MPI
00114   int mpi_rank;
00115 #endif
00116 } Optimize;
00117
00122 typedef struct
00123 {
00124   unsigned int thread;
00125 } ParallelData;
00126
00127 // Global variables
00128 extern int ntasks;
00129 extern unsigned int nthreads;
00130 extern unsigned int nthreads_direction;
00131 extern GMutex mutex[1];
00132 extern void (*optimize_algorithm) ();
00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134                                               unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
```

```
00137
00138 // Public functions
00139 void optimize_input (unsigned int simulation, char *input,
00140                      GMappedFile * template);
00141 double optimize_parse (unsigned int simulation, unsigned int experiment);
00142 double optimize_norm_euclidian (unsigned int simulation);
00143 double optimize_norm_maximum (unsigned int simulation);
00144 double optimize_norm_p (unsigned int simulation);
00145 double optimize_norm_taxicab (unsigned int simulation);
00146 void optimize_print ();
00147 void optimize_save_variables (unsigned int simulation, double error);
00148 void optimize_best (unsigned int simulation, double value);
00149 void optimize_sequential ();
00150 void *optimize_thread (ParallelData * data);
00151 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00152                      double *error_best);
00153 #if HAVE_MPI
00154 void optimize_synchronise ();
00155 #endif
00156 void optimize_sweep ();
00157 void optimize_MonteCarlo ();
00158 void optimize_best_direction (unsigned int simulation, double value);
00159 void optimize_direction_sequential (unsigned int simulation);
00160 void *optimize_direction_thread (ParallelData * data);
00161 double optimize_estimate_direction_random (unsigned int variable,
00162                                            unsigned int estimate);
00163 double optimize_estimate_direction_coordinates (unsigned int
    variable,
00164                                                 unsigned int estimate);
00165 void optimize_step_direction (unsigned int simulation);
00166 void optimize_direction ();
00167 double optimize_genetic_objective (Entity * entity);
00168 void optimize_genetic ();
00169 void optimize_save_old ();
00170 void optimize_merge_old ();
00171 void optimize_refine ();
00172 void optimize_step ();
00173 void optimize_iterate ();
00174 void optimize_free ();
00175 void optimize_open ();
00176
00177 #endif
```

5.21 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 <json-glib/json-glib.h>
#include <gtk/gtk.h>
#include "utils.h"
```

Include dependency graph for utils.c:

Functions

- void show_pending ()

  Function to show events on long computation.
- 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.
- 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.

5.21.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file utils.c.

5.21.2 Function Documentation

5.21.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 537 of file utils.c.

```
00538 {
00539 #ifdef G_OS_WIN32
00540   SYSTEM_INFO sysinfo;
00541   GetSystemInfo (&sysinfo);
00542   return sysinfo.dwNumberOfProcessors;
00543 #else
00544   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00545 #endif
00546 }
```

5.21.2.2 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 561 of file utils.c.

```
00562 {
00563   unsigned int i;
00564   for (i = 0; i < n; ++i)
00565     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00566       break;
00567   return i;
00568 }
```

5.21.2.3 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 427 of file utils.c.

```
00428 {
00429   const char *buffer;
00430   double x = 0.;
00431   buffer = json_object_get_string_member (object, prop);
00432   if (!buffer)
00433     *error_code = 1;
00434   else
00435     {
00436       if (sscanf (buffer, "%lf", &x) != 1)
00437         *error_code = 2;
00438       else
00439         *error_code = 0;
00440     }
00441   return x;
00442 }
```

5.21.2.4 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 460 of file utils.c.

```
00462 {
00463   double x;
00464   if (json_object_get_member (object, prop))
00465     x = json_object_get_float (object, prop, error_code);
00466   else
00467     {
00468       x = default_value;
00469       *error_code = 0;
00470     }
00471   return x;
00472 }
```

Here is the call graph for this function:

5.21.2.5 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 337 of file utils.c.

```
00338 {
00339   const char *buffer;
00340   int i = 0;
00341   buffer = json_object_get_string_member (object, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf (buffer, "%d", &i) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350     }
00351   return i;
00352 }
```

5.21.2.6 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 367 of file utils.c.

```
00368 {
00369   const char *buffer;
00370   unsigned int i = 0;
00371   buffer = json_object_get_string_member (object, prop);
00372   if (!buffer)
00373     *error_code = 1;
00374   else
00375     {
00376       if (sscanf (buffer, "%u", &i) != 1)
00377         *error_code = 2;
00378       else
00379         *error_code = 0;
00380     }
00381   return i;
00382 }
```

5.21.2.7 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 400 of file utils.c.

```
00402 {
00403   unsigned int i;
00404   if (json_object_get_member (object, prop))
00405     i = json_object_get_uint (object, prop, error_code);
00406   else
00407     {
00408       i = default_value;
00409       *error_code = 0;
00410     }
00411   return i;
00412 }
```

Here is the call graph for this function:

5.21.2.8 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 524 of file utils.c.

```
00525 {
00526   char buffer[64];
00527   snprintf (buffer, 64, "%.14lg", value);
00528   json_object_set_string_member (object, prop, buffer);
00529 }
```

5.21.2.9 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 486 of file utils.c.

```
00487 {
00488   char buffer[64];
00489   snprintf (buffer, 64, "%d", value);
00490   json_object_set_string_member (object, prop, buffer);
00491 }
```

5.21.2.10 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 505 of file utils.c.

```
00506 {
00507   char buffer[64];
00508   snprintf (buffer, 64, "%u", value);
00509   json_object_set_string_member (object, prop, buffer);
00510 }
```

5.21.2.11 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 110 of file utils.c.

```
00111 {
00112   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00113 }
```

Here is the call graph for this function:

5.21.2.12 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 80 of file utils.c.

```
00081 {
00082 #if HAVE_GTK
00083   GtkMessageDialog *dlg;
00084
00085   // Creating the dialog
00086   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00087     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00088
00089   // Setting the dialog title
00090   gtk_window_set_title (GTK_WINDOW (dlg), title);
00091
00092   // Showing the dialog and waiting response
00093   gtk_dialog_run (GTK_DIALOG (dlg));
00094
00095   // Closing and freeing memory
00096   gtk_widget_destroy (GTK_WIDGET (dlg));
00097
00098 #else
00099   printf ("%s: %s\n", title, msg);
00100 #endif
00101 }
```

5.21.2.13 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 220 of file utils.c.

```
00221 {
00222   double x = 0.;
00223   xmlChar *buffer;
00224   buffer = xmlGetProp (node, prop);
00225   if (!buffer)
00226     *error_code = 1;
00227   else
00228     {
00229       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00230         *error_code = 2;
00231       else
00232         *error_code = 0;
00233       xmlFree (buffer);
00234     }
00235   return x;
00236 }
```

5.21.2.14 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 254 of file utils.c.

```
00256 {
00257   double x;
00258   if (xmlHasProp (node, prop))
00259     x = xml_node_get_float (node, prop, error_code);
00260   else
00261     {
00262       x = default_value;
00263       *error_code = 0;
00264     }
00265   return x;
00266 }
```

Here is the call graph for this function:

5.21.2.15 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 128 of file utils.c.

```
00129 {
00130   int i = 0;
00131   xmlChar *buffer;
00132   buffer = xmlGetProp (node, prop);
00133   if (!buffer)
00134     *error_code = 1;
00135   else
00136     {
00137       if (sscanf ((char *) buffer, "%d", &i) != 1)
00138         *error_code = 2;
00139       else
00140         *error_code = 0;
00141       xmlFree (buffer);
00142     }
00143   return i;
00144 }
```

5.21.2.16 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 159 of file utils.c.

```
00160 {
00161   unsigned int i = 0;
00162   xmlChar *buffer;
00163   buffer = xmlGetProp (node, prop);
00164   if (!buffer)
00165     *error_code = 1;
00166   else
00167     {
00168       if (sscanf ((char *) buffer, "%u", &i) != 1)
00169         *error_code = 2;
00170       else
00171         *error_code = 0;
00172       xmlFree (buffer);
00173     }
00174   return i;
00175 }
```

5.21.2.17 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 193 of file utils.c.

```
00195 {
00196   unsigned int i;
00197   if (xmlHasProp (node, prop))
00198     i = xml_node_get_uint (node, prop, error_code);
00199   else
00200     {
00201       i = default_value;
00202       *error_code = 0;
00203     }
00204   return i;
00205 }
```

Here is the call graph for this function:

5.21.2.18 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 317 of file utils.c.

```
00318 {
00319   xmlChar buffer[64];
00320   snprintf ((char *) buffer, 64, "%.14lg", value);
00321   xmlSetProp (node, prop, buffer);
00322 }
```

5.21.2.19 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 279 of file utils.c.

```
00280 {
00281   xmlChar buffer[64];
00282   snprintf ((char *) buffer, 64, "%d", value);
00283   xmlSetProp (node, prop, buffer);
00284 }
```

5.21.2.20 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 298 of file utils.c.

```
00299 {
00300   xmlChar buffer[64];
00301   snprintf ((char *) buffer, 64, "%u", value);
00302   xmlSetProp (node, prop, buffer);
00303 }
```

5.22 utils.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <unistd.h>
00042 #include <libxml/parser.h>
00043 #include <libintl.h>
00044 #include <json-glib/json-glib.h>
00045 #if HAVE_GTK
00046 #include <gtk/gtk.h>
00047 #endif
00048 #include "utils.h"
00049
00050 #if HAVE_GTK
00051 GtkWindow *main_window;
00052 #endif
00053
00054 char *error_message;
00055
00060 void
00061 show_pending ()
00062 {
00063 #if HAVE_GTK
00064   while (gtk_events_pending ())
00065     gtk_main_iteration ();
00066 #endif
00067 }
00068
00079 void
00080 show_message (char *title, char *msg, int type)
00081 {
00082 #if HAVE_GTK
00083   GtkMessageDialog *dlg;
00084
```

```
00085   // Creating the dialog
00086   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00087     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00088
00089   // Setting the dialog title
00090   gtk_window_set_title (GTK_WINDOW (dlg), title);
00091
00092   // Showing the dialog and waiting response
00093   gtk_dialog_run (GTK_DIALOG (dlg));
00094
00095   // Closing and freeing memory
00096   gtk_widget_destroy (GTK_WIDGET (dlg));
00097
00098 #else
00099   printf ("%s: %s\n", title, msg);
00100 #endif
00101 }
00102
00109 void
00110 show_error (char *msg)
00111 {
00112   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00113 }
00114
00127 int
00128 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00129 {
00130   int i = 0;
00131   xmlChar *buffer;
00132   buffer = xmlGetProp (node, prop);
00133   if (!buffer)
00134     *error_code = 1;
00135   else
00136     {
00137       if (sscanf ((char *) buffer, "%d", &i) != 1)
00138         *error_code = 2;
00139       else
00140         *error_code = 0;
00141       xmlFree (buffer);
00142     }
00143   return i;
00144 }
00145
00158 unsigned int
00159 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00160 {
00161   unsigned int i = 0;
00162   xmlChar *buffer;
00163   buffer = xmlGetProp (node, prop);
00164   if (!buffer)
00165     *error_code = 1;
00166   else
00167     {
00168       if (sscanf ((char *) buffer, "%u", &i) != 1)
00169         *error_code = 2;
00170       else
00171         *error_code = 0;
00172       xmlFree (buffer);
00173     }
00174   return i;
00175 }
00176
00192 unsigned int
00193 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00194                                 unsigned int default_value, int *error_code)
00195 {
00196   unsigned int i;
00197   if (xmlHasProp (node, prop))
00198     i = xml_node_get_uint (node, prop, error_code);
00199   else
00200     {
00201       i = default_value;
00202       *error_code = 0;
00203     }
00204   return i;
00205 }
00206
00219 double
00220 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00221 {
00222   double x = 0.;
00223   xmlChar *buffer;
00224   buffer = xmlGetProp (node, prop);
00225   if (!buffer)
00226     *error_code = 1;
00227   else
00228     {
```

```
00229       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00230         *error_code = 2;
00231       else
00232         *error_code = 0;
00233       xmlFree (buffer);
00234     }
00235   return x;
00236 }
00237
00253 double
00254 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00255                                  double default_value, int *error_code)
00256 {
00257   double x;
00258   if (xmlHasProp (node, prop))
00259     x = xml_node_get_float (node, prop, error_code);
00260   else
00261     {
00262       x = default_value;
00263       *error_code = 0;
00264     }
00265   return x;
00266 }
00267
00278 void
00279 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00280 {
00281   xmlChar buffer[64];
00282   snprintf ((char *) buffer, 64, "%d", value);
00283   xmlSetProp (node, prop, buffer);
00284 }
00285
00297 void
00298 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00299 {
00300   xmlChar buffer[64];
00301   snprintf ((char *) buffer, 64, "%u", value);
00302   xmlSetProp (node, prop, buffer);
00303 }
00304
00316 void
00317 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00318 {
00319   xmlChar buffer[64];
00320   snprintf ((char *) buffer, 64, "%.14lg", value);
00321   xmlSetProp (node, prop, buffer);
00322 }
00323
00336 int
00337 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00338 {
00339   const char *buffer;
00340   int i = 0;
00341   buffer = json_object_get_string_member (object, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf (buffer, "%d", &i) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350     }
00351   return i;
00352 }
00353
00366 unsigned int
00367 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00368 {
00369   const char *buffer;
00370   unsigned int i = 0;
00371   buffer = json_object_get_string_member (object, prop);
00372   if (!buffer)
00373     *error_code = 1;
00374   else
00375     {
00376       if (sscanf (buffer, "%u", &i) != 1)
00377         *error_code = 2;
00378       else
00379         *error_code = 0;
00380     }
00381   return i;
00382 }
00383
00399 unsigned int
00400 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00401                                    unsigned int default_value, int *error_code)
```

```
00402 {
00403   unsigned int i;
00404   if (json_object_get_member (object, prop))
00405     i = json_object_get_uint (object, prop, error_code);
00406   else
00407     {
00408       i = default_value;
00409       *error_code = 0;
00410     }
00411   return i;
00412 }
00413
00426 double
00427 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00428 {
00429   const char *buffer;
00430   double x = 0.;
00431   buffer = json_object_get_string_member (object, prop);
00432   if (!buffer)
00433     *error_code = 1;
00434   else
00435     {
00436       if (sscanf (buffer, "%lf", &x) != 1)
00437         *error_code = 2;
00438       else
00439         *error_code = 0;
00440     }
00441   return x;
00442 }
00443
00459 double
00460 json_object_get_float_with_default (JsonObject * object, const char *prop
      ,
00461                                     double default_value, int *error_code)
00462 {
00463   double x;
00464   if (json_object_get_member (object, prop))
00465     x = json_object_get_float (object, prop, error_code);
00466   else
00467     {
00468       x = default_value;
00469       *error_code = 0;
00470     }
00471   return x;
00472 }
00473
00485 void
00486 json_object_set_int (JsonObject * object, const char *prop, int value)
00487 {
00488   char buffer[64];
00489   snprintf (buffer, 64, "%d", value);
00490   json_object_set_string_member (object, prop, buffer);
00491 }
00492
00504 void
00505 json_object_set_uint (JsonObject * object, const char *prop, unsigned int value)
00506 {
00507   char buffer[64];
00508   snprintf (buffer, 64, "%u", value);
00509   json_object_set_string_member (object, prop, buffer);
00510 }
00511
00523 void
00524 json_object_set_float (JsonObject * object, const char *prop, double value)
00525 {
00526   char buffer[64];
00527   snprintf (buffer, 64, "%.14lg", value);
00528   json_object_set_string_member (object, prop, buffer);
00529 }
00530
00536 int
00537 cores_number ()
00538 {
00539 #ifdef G_OS_WIN32
00540   SYSTEM_INFO sysinfo;
00541   GetSystemInfo (&sysinfo);
00542   return sysinfo.dwNumberOfProcessors;
00543 #else
00544   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00545 #endif
00546 }
00547
00548 #if HAVE_GTK
00549
00560 unsigned int
00561 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00562 {
```

```
00563   unsigned int i;
00564   for (i = 0; i < n; ++i)
00565     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00566       break;
00567   return i;
00568 }
00569
00570 #endif
```

5.23 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_pending ()

  Function to show events on long computation.
- 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.
- 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.

5.23.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file utils.h.

5.23.2 Function Documentation

5.23.2.1 int cores_number ( )

Function to obtain the cores number.

Returns

Cores number.

Definition at line 537 of file utils.c.

```
00538 {
00539 #ifdef G_OS_WIN32
00540   SYSTEM_INFO sysinfo;
00541   GetSystemInfo (&sysinfo);
00542   return sysinfo.dwNumberOfProcessors;
00543 #else
00544   return (int) sysconf (_SC_NPROCESSORS_ONLN);
00545 #endif
00546 }
```

5.23.2.2 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 561 of file utils.c.

```
00562 {
00563   unsigned int i;
00564   for (i = 0; i < n; ++i)
00565     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00566       break;
00567   return i;
00568 }
```

5.23.2.3 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 427 of file utils.c.

```
00428 {
00429   const char *buffer;
00430   double x = 0.;
00431   buffer = json_object_get_string_member (object, prop);
00432   if (!buffer)
00433     *error_code = 1;
00434   else
00435     {
00436       if (sscanf (buffer, "%lf", &x) != 1)
00437         *error_code = 2;
00438       else
00439         *error_code = 0;
00440     }
00441   return x;
00442 }
```

5.23.2.4 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 460 of file utils.c.

```
00462 {
00463   double x;
00464   if (json_object_get_member (object, prop))
00465     x = json_object_get_float (object, prop, error_code);
00466   else
00467     {
00468       x = default_value;
00469       *error_code = 0;
00470     }
00471   return x;
00472 }
```

Here is the call graph for this function:

5.23.2.5 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 337 of file utils.c.

```
00338 {
00339   const char *buffer;
00340   int i = 0;
00341   buffer = json_object_get_string_member (object, prop);
00342   if (!buffer)
00343     *error_code = 1;
00344   else
00345     {
00346       if (sscanf (buffer, "%d", &i) != 1)
00347         *error_code = 2;
00348       else
00349         *error_code = 0;
00350     }
00351   return i;
00352 }
```

5.23.2.6 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 367 of file utils.c.

```
00368 {
00369   const char *buffer;
00370   unsigned int i = 0;
00371   buffer = json_object_get_string_member (object, prop);
00372   if (!buffer)
00373     *error_code = 1;
00374   else
00375     {
00376       if (sscanf (buffer, "%u", &i) != 1)
00377         *error_code = 2;
00378       else
00379         *error_code = 0;
00380     }
00381   return i;
00382 }
```

5.23.2.7 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 400 of file utils.c.

```
00402 {
00403   unsigned int i;
00404   if (json_object_get_member (object, prop))
00405     i = json_object_get_uint (object, prop, error_code);
00406   else
00407     {
00408       i = default_value;
00409       *error_code = 0;
00410     }
00411   return i;
00412 }
```

Here is the call graph for this function:

5.23.2.8 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 524 of file utils.c.

```
00525 {
00526   char buffer[64];
00527   snprintf (buffer, 64, "%.14lg", value);
00528   json_object_set_string_member (object, prop, buffer);
00529 }
```

5.23.2.9 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 486 of file utils.c.

```
00487 {
00488   char buffer[64];
00489   snprintf (buffer, 64, "%d", value);
00490   json_object_set_string_member (object, prop, buffer);
00491 }
```

5.23.2.10 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 505 of file utils.c.

```
00506 {
00507   char buffer[64];
00508   snprintf (buffer, 64, "%u", value);
00509   json_object_set_string_member (object, prop, buffer);
00510 }
```

5.23.2.11 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

msg	Error message.

Definition at line 110 of file utils.c.

```
00111 {
00112   show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00113 }
```

Here is the call graph for this function:

5.23.2.12 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 80 of file utils.c.

```
00081 {
00082 #if HAVE_GTK
00083   GtkMessageDialog *dlg;
00084
00085   // Creating the dialog
00086   dlg = (GtkMessageDialog *) gtk_message_dialog_new
00087     (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00088
00089   // Setting the dialog title
00090   gtk_window_set_title (GTK_WINDOW (dlg), title);
00091
00092   // Showing the dialog and waiting response
00093   gtk_dialog_run (GTK_DIALOG (dlg));
00094
00095   // Closing and freeing memory
00096   gtk_widget_destroy (GTK_WIDGET (dlg));
00097
00098 #else
00099   printf ("%s: %s\n", title, msg);
00100 #endif
00101 }
```

5.23.2.13 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 220 of file utils.c.

```
00221 {
00222   double x = 0.;
00223   xmlChar *buffer;
00224   buffer = xmlGetProp (node, prop);
00225   if (!buffer)
00226     *error_code = 1;
00227   else
00228     {
00229       if (sscanf ((char *) buffer, "%lf", &x) != 1)
00230         *error_code = 2;
00231       else
00232         *error_code = 0;
00233       xmlFree (buffer);
00234     }
00235   return x;
00236 }
```

5.23.2.14 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 254 of file utils.c.

```
00256 {
00257   double x;
00258   if (xmlHasProp (node, prop))
00259     x = xml_node_get_float (node, prop, error_code);
00260   else
00261     {
00262       x = default_value;
00263       *error_code = 0;
00264     }
00265   return x;
00266 }
```

Here is the call graph for this function:

5.23.2.15 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 128 of file utils.c.

```
00129 {
00130   int i = 0;
00131   xmlChar *buffer;
00132   buffer = xmlGetProp (node, prop);
00133   if (!buffer)
00134     *error_code = 1;
00135   else
00136     {
00137       if (sscanf ((char *) buffer, "%d", &i) != 1)
00138         *error_code = 2;
00139       else
00140         *error_code = 0;
00141       xmlFree (buffer);
00142     }
00143   return i;
00144 }
```

5.23.2.16 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 159 of file utils.c.

```
00160 {
00161   unsigned int i = 0;
00162   xmlChar *buffer;
00163   buffer = xmlGetProp (node, prop);
00164   if (!buffer)
00165     *error_code = 1;
00166   else
00167     {
00168       if (sscanf ((char *) buffer, "%u", &i) != 1)
00169         *error_code = 2;
00170       else
00171         *error_code = 0;
00172       xmlFree (buffer);
00173     }
00174   return i;
00175 }
```

5.23.2.17 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 193 of file utils.c.

```
00195 {
00196   unsigned int i;
00197   if (xmlHasProp (node, prop))
00198     i = xml_node_get_uint (node, prop, error_code);
00199   else
00200     {
00201       i = default_value;
00202       *error_code = 0;
00203     }
00204   return i;
00205 }
```

Here is the call graph for this function:

5.23.2.18 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 317 of file utils.c.

```
00318 {
00319   xmlChar buffer[64];
00320   snprintf ((char *) buffer, 64, "%.14lg", value);
00321   xmlSetProp (node, prop, buffer);
00322 }
```

5.23.2.19 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 279 of file utils.c.

```
00280 {
00281   xmlChar buffer[64];
00282   snprintf ((char *) buffer, 64, "%d", value);
00283   xmlSetProp (node, prop, buffer);
00284 }
```

5.23.2.20 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 298 of file utils.c.

```
00299 {
00300   xmlChar buffer[64];
00301   snprintf ((char *) buffer, 64, "%u", value);
00302   xmlSetProp (node, prop, buffer);
00303 }
```

5.24 utils.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
```

```
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef UTILS__H
00039 #define UTILS__H 1
00040
00047 #if HAVE_GTK
00048 #define ERROR_TYPE GTK_MESSAGE_ERROR
00049 #define INFO_TYPE GTK_MESSAGE_INFO
00050 extern GtkWindow *main_window;
00051 #else
00052 #define ERROR_TYPE 0
00053 #define INFO_TYPE 0
00054 #endif
00055
00056 extern char *error_message;
00057
00058 // Public functions
00059 void show_pending ();
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 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00099 #endif
00100
00101 #endif
```

5.25 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 **_GNU_SOURCE**
- #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.

5.25.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Definition in file variable.c.

5.25.2 Function Documentation

5.25.2.1 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", gettext ("Variable"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117               message);
00118   error_message = g_strdup (buffer);
00119 }
```

5.25.2.2 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 }
```

5.25.2.3 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 }
```

5.25.2.4 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 302 of file variable.c.

```
00304 {
00305   JsonObject *object;
00306   const char *label;
00307   int error_code;
00308 #if DEBUG_VARIABLE
00309   fprintf (stderr, "variable_open_json: start\n");
00310 #endif
00311   object = json_node_get_object (node);
00312   label = json_object_get_string_member (object, LABEL_NAME);
00313   if (!label)
00314     {
00315       variable_error (variable, gettext ("no name"));
00316       goto exit_on_error;
00317     }
00318   variable->name = g_strdup (label);
00319   if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321       variable->rangemin
00322         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323       if (error_code)
00324         {
00325           variable_error (variable, gettext ("bad minimum"));
00326           goto exit_on_error;
00327         }
00328       variable->rangeminabs
00329         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00330                                               -G_MAXDOUBLE, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, gettext ("bad absolute minimum"));
00334           goto exit_on_error;
00335         }
00336       if (variable->rangemin < variable->rangeminabs)
00337         {
00338           variable_error (variable, gettext ("minimum range not allowed"));
00339           goto exit_on_error;
00340         }
00341     }
00342   else
00343     {
00344       variable_error (variable, gettext ("no minimum range"));
00345       goto exit_on_error;
00346     }
00347   if (json_object_get_member (object, LABEL_MAXIMUM))
00348     {
00349       variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351       if (error_code)
00352         {
00353           variable_error (variable, gettext ("bad maximum"));
00354           goto exit_on_error;
00355         }
00356       variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00358                                               G_MAXDOUBLE, &error_code);
00359       if (error_code)
00360         {
00361           variable_error (variable, gettext ("bad absolute maximum"));
00362           goto exit_on_error;
00363         }
00364       if (variable->rangemax > variable->rangemaxabs)
00365         {
00366           variable_error (variable, gettext ("maximum range not allowed"));
00367           goto exit_on_error;
00368         }
00369       if (variable->rangemax < variable->rangemin)
```

```
00370         {
00371           variable_error (variable, gettext ("bad range"));
00372           goto exit_on_error;
00373         }
00374     }
00375   else
00376     {
00377       variable_error (variable, gettext ("no maximum range"));
00378       goto exit_on_error;
00379     }
00380   variable->precision
00381     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00382                                          DEFAULT_PRECISION, &error_code);
00383   if (error_code || variable->precision >= NPRECISIONS)
00384     {
00385       variable_error (variable, gettext ("bad precision"));
00386       goto exit_on_error;
00387     }
00388   if (algorithm == ALGORITHM_SWEEP)
00389     {
00390       if (json_object_get_member (object, LABEL_NSWEEPS))
00391         {
00392           variable->nsweeps
00393             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00394           if (error_code || !variable->nsweeps)
00395             {
00396               variable_error (variable, gettext ("bad sweeps"));
00397               goto exit_on_error;
00398             }
00399         }
00400       else
00401         {
00402           variable_error (variable, gettext ("no sweeps number"));
00403           goto exit_on_error;
00404         }
00405 #if DEBUG_VARIABLE
00406       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00407 #endif
00408     }
00409   if (algorithm == ALGORITHM_GENETIC)
00410     {
00411       // Obtaining bits representing each variable
00412       if (json_object_get_member (object, LABEL_NBITS))
00413         {
00414           variable->nbits
00415             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00416           if (error_code || !variable->nbits)
00417             {
00418               variable_error (variable, gettext ("invalid bits number"));
00419               goto exit_on_error;
00420             }
00421         }
00422       else
00423         {
00424           variable_error (variable, gettext ("no bits number"));
00425           goto exit_on_error;
00426         }
00427     }
00428   else if (nsteps)
00429     {
00430       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00431       if (error_code || variable->step < 0.)
00432         {
00433           variable_error (variable, gettext ("bad step size"));
00434           goto exit_on_error;
00435         }
00436     }
00437
00438 #if DEBUG_VARIABLE
00439   fprintf (stderr, "variable_open_json: end\n");
00440 #endif
00441   return 1;
00442 exit_on_error:
00443   variable_free (variable, INPUT_TYPE_JSON);
00444 #if DEBUG_VARIABLE
00445   fprintf (stderr, "variable_open_json: end\n");
00446 #endif
00447   return 0;
00448 }
```

Here is the call graph for this function:

5.25.2.5 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 136 of file variable.c.

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

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

Here is the call graph for this function:

5.25.3 Variable Documentation

5.25.3.1 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.

5.25.3.2 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.

5.26 variable.c

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #define _GNU_SOURCE
00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
```

```
00045 #include "utils.h"
00046 #include "variable.h"
00047
00048 #define DEBUG_VARIABLE 0
00049
00050 const char *format[NPRECISIONS] = {
00051   "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00052   "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00053 };
00054
00055 const double precision[NPRECISIONS] = {
00056   1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00057   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", gettext ("Variable"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117               message);
00118   error_message = g_strdup (buffer);
00119 }
00120
00135 int
00136 variable_open_xml (Variable * variable, xmlNode * node, unsigned int algorithm,
00137                    unsigned int nsteps)
00138 {
00139   int error_code;
00140
00141 #if DEBUG_VARIABLE
00142   fprintf (stderr, "variable_open_xml: start\n");
00143 #endif
00144
00145   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146   if (!variable->name)
00147     {
00148       variable_error (variable, gettext ("no name"));
00149       goto exit_on_error;
00150     }
00151   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153       variable->rangemin
00154         = xml_node_get_float (node, (const xmlChar *)
      LABEL_MINIMUM,
00155                               &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, gettext ("bad minimum"));
00159           goto exit_on_error;
00160         }
00161       variable->rangeminabs = xml_node_get_float_with_default
00162         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163          &error_code);
00164       if (error_code)
00165         {
00166           variable_error (variable, gettext ("bad absolute minimum"));
```

```
00167           goto exit_on_error;
00168         }
00169       if (variable->rangemin < variable->rangeminabs)
00170         {
00171           variable_error (variable, gettext ("minimum range not allowed"));
00172           goto exit_on_error;
00173         }
00174     }
00175   else
00176     {
00177       variable_error (variable, gettext ("no minimum range"));
00178       goto exit_on_error;
00179     }
00180   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182       variable->rangemax
00183         = xml_node_get_float (node, (const xmlChar *)
      LABEL_MAXIMUM,
00184                               &error_code);
00185       if (error_code)
00186         {
00187           variable_error (variable, gettext ("bad maximum"));
00188           goto exit_on_error;
00189         }
00190       variable->rangemaxabs = xml_node_get_float_with_default
00191         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192          &error_code);
00193       if (error_code)
00194         {
00195           variable_error (variable, gettext ("bad absolute maximum"));
00196           goto exit_on_error;
00197         }
00198       if (variable->rangemax > variable->rangemaxabs)
00199         {
00200           variable_error (variable, gettext ("maximum range not allowed"));
00201           goto exit_on_error;
00202         }
00203       if (variable->rangemax < variable->rangemin)
00204         {
00205           variable_error (variable, gettext ("bad range"));
00206           goto exit_on_error;
00207         }
00208     }
00209   else
00210     {
00211       variable_error (variable, gettext ("no maximum range"));
00212       goto exit_on_error;
00213     }
00214   variable->precision
00215     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_PRECISION,
00216                                       DEFAULT_PRECISION, &error_code);
00217   if (error_code || variable->precision >= NPRECISIONS)
00218     {
00219       variable_error (variable, gettext ("bad precision"));
00220       goto exit_on_error;
00221     }
00222   if (algorithm == ALGORITHM_SWEEP)
00223     {
00224       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225         {
00226           variable->nsweeps
00227             = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NSWEEPS,
00228                                  &error_code);
00229           if (error_code || !variable->nsweeps)
00230             {
00231               variable_error (variable, gettext ("bad sweeps"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           variable_error (variable, gettext ("no sweeps number"));
00238           goto exit_on_error;
00239         }
00240 #if DEBUG_VARIABLE
00241       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243     }
00244   if (algorithm == ALGORITHM_GENETIC)
00245     {
00246       // Obtaining bits representing each variable
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248         {
00249           variable->nbits
00250             = xml_node_get_uint (node, (const xmlChar *)
```

```
      LABEL_NBITS,
00251                                  &error_code);
00252           if (error_code || !variable->nbits)
00253             {
00254               variable_error (variable, gettext ("invalid bits number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           variable_error (variable, gettext ("no bits number"));
00261           goto exit_on_error;
00262         }
00263     }
00264   else if (nsteps)
00265     {
00266       variable->step
00267         = xml_node_get_float (node, (const xmlChar *)
      LABEL_STEP, &error_code);
00268       if (error_code || variable->step < 0.)
00269         {
00270           variable_error (variable, gettext ("bad step size"));
00271           goto exit_on_error;
00272         }
00273     }
00274
00275 #if DEBUG_VARIABLE
00276   fprintf (stderr, "variable_open_xml: end\n");
00277 #endif
00278   return 1;
00279 exit_on_error:
00280   variable_free (variable, INPUT_TYPE_XML);
00281 #if DEBUG_VARIABLE
00282   fprintf (stderr, "variable_open_xml: end\n");
00283 #endif
00284   return 0;
00285 }
00286
00301 int
00302 variable_open_json (Variable * variable, JsonNode * node,
00303                     unsigned int algorithm, unsigned int nsteps)
00304 {
00305   JsonObject *object;
00306   const char *label;
00307   int error_code;
00308 #if DEBUG_VARIABLE
00309   fprintf (stderr, "variable_open_json: start\n");
00310 #endif
00311   object = json_node_get_object (node);
00312   label = json_object_get_string_member (object, LABEL_NAME);
00313   if (!label)
00314     {
00315       variable_error (variable, gettext ("no name"));
00316       goto exit_on_error;
00317     }
00318   variable->name = g_strdup (label);
00319   if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321       variable->rangemin
00322         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323       if (error_code)
00324         {
00325           variable_error (variable, gettext ("bad minimum"));
00326           goto exit_on_error;
00327         }
00328       variable->rangeminabs
00329         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MINIMUM,
00330                                             -G_MAXDOUBLE, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, gettext ("bad absolute minimum"));
00334           goto exit_on_error;
00335         }
00336       if (variable->rangemin < variable->rangeminabs)
00337         {
00338           variable_error (variable, gettext ("minimum range not allowed"));
00339           goto exit_on_error;
00340         }
00341     }
00342   else
00343     {
00344       variable_error (variable, gettext ("no minimum range"));
00345       goto exit_on_error;
00346     }
00347   if (json_object_get_member (object, LABEL_MAXIMUM))
00348     {
```

```
00349       variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351       if (error_code)
00352         {
00353           variable_error (variable, gettext ("bad maximum"));
00354           goto exit_on_error;
00355         }
00356       variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
      LABEL_ABSOLUTE_MAXIMUM,
00358                                               G_MAXDOUBLE, &error_code);
00359       if (error_code)
00360         {
00361           variable_error (variable, gettext ("bad absolute maximum"));
00362           goto exit_on_error;
00363         }
00364       if (variable->rangemax > variable->rangemaxabs)
00365         {
00366           variable_error (variable, gettext ("maximum range not allowed"));
00367           goto exit_on_error;
00368         }
00369       if (variable->rangemax < variable->rangemin)
00370         {
00371           variable_error (variable, gettext ("bad range"));
00372           goto exit_on_error;
00373         }
00374     }
00375   else
00376     {
00377       variable_error (variable, gettext ("no maximum range"));
00378       goto exit_on_error;
00379     }
00380   variable->precision
00381     = json_object_get_uint_with_default (object,
      LABEL_PRECISION,
00382                                          DEFAULT_PRECISION, &error_code);
00383   if (error_code || variable->precision >= NPRECISIONS)
00384     {
00385       variable_error (variable, gettext ("bad precision"));
00386       goto exit_on_error;
00387     }
00388   if (algorithm == ALGORITHM_SWEEP)
00389     {
00390       if (json_object_get_member (object, LABEL_NSWEEPS))
00391         {
00392           variable->nsweeps
00393             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00394           if (error_code || !variable->nsweeps)
00395             {
00396               variable_error (variable, gettext ("bad sweeps"));
00397               goto exit_on_error;
00398             }
00399         }
00400       else
00401         {
00402           variable_error (variable, gettext ("no sweeps number"));
00403           goto exit_on_error;
00404         }
00405 #if DEBUG_VARIABLE
00406       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00407 #endif
00408     }
00409   if (algorithm == ALGORITHM_GENETIC)
00410     {
00411       // Obtaining bits representing each variable
00412       if (json_object_get_member (object, LABEL_NBITS))
00413         {
00414           variable->nbits
00415             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00416           if (error_code || !variable->nbits)
00417             {
00418               variable_error (variable, gettext ("invalid bits number"));
00419               goto exit_on_error;
00420             }
00421         }
00422       else
00423         {
00424           variable_error (variable, gettext ("no bits number"));
00425           goto exit_on_error;
00426         }
00427     }
00428   else if (nsteps)
00429     {
00430       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00431       if (error_code || variable->step < 0.)
00432         {
```

```
00433           variable_error (variable, gettext ("bad step size"));
00434           goto exit_on_error;
00435         }
00436     }
00437
00438 #if DEBUG_VARIABLE
00439   fprintf (stderr, "variable_open_json: end\n");
00440 #endif
00441   return 1;
00442 exit_on_error:
00443   variable_free (variable, INPUT_TYPE_JSON);
00444 #if DEBUG_VARIABLE
00445   fprintf (stderr, "variable_open_json: end\n");
00446 #endif
00447   return 0;
00448 }
```

5.27 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.

5.27.1 Detailed Description

Header file to define the variable data.

Authors

Javier Burguete.

Copyright

Definition in file variable.h.

5.27.2 Enumeration Type Documentation

5.27.2.1 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 };
```

5.27.3 Function Documentation

5.27.3.1 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", gettext ("Variable"), message);
00115   else
00116     snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117               message);
00118   error_message = g_strdup (buffer);
00119 }
```

5.27.3.2 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 }
```

5.27.3.3 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 }
```

5.27.3.4 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 302 of file variable.c.

```
00304 {
00305   JsonObject *object;
00306   const char *label;
00307   int error_code;
00308 #if DEBUG_VARIABLE
00309   fprintf (stderr, "variable_open_json: start\n");
00310 #endif
00311   object = json_node_get_object (node);
00312   label = json_object_get_string_member (object, LABEL_NAME);
00313   if (!label)
00314     {
```

```
00315       variable_error (variable, gettext ("no name"));
00316       goto exit_on_error;
00317     }
00318   variable->name = g_strdup (label);
00319   if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321       variable->rangemin
00322         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323       if (error_code)
00324         {
00325           variable_error (variable, gettext ("bad minimum"));
00326           goto exit_on_error;
00327         }
00328       variable->rangeminabs
00329         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MINIMUM,
00330                                               -G_MAXDOUBLE, &error_code);
00331       if (error_code)
00332         {
00333           variable_error (variable, gettext ("bad absolute minimum"));
00334           goto exit_on_error;
00335         }
00336       if (variable->rangemin < variable->rangeminabs)
00337         {
00338           variable_error (variable, gettext ("minimum range not allowed"));
00339           goto exit_on_error;
00340         }
00341     }
00342   else
00343     {
00344       variable_error (variable, gettext ("no minimum range"));
00345       goto exit_on_error;
00346     }
00347   if (json_object_get_member (object, LABEL_MAXIMUM))
00348     {
00349       variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351       if (error_code)
00352         {
00353           variable_error (variable, gettext ("bad maximum"));
00354           goto exit_on_error;
00355         }
00356       variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
    LABEL_ABSOLUTE_MAXIMUM,
00358                                               G_MAXDOUBLE, &error_code);
00359       if (error_code)
00360         {
00361           variable_error (variable, gettext ("bad absolute maximum"));
00362           goto exit_on_error;
00363         }
00364       if (variable->rangemax > variable->rangemaxabs)
00365         {
00366           variable_error (variable, gettext ("maximum range not allowed"));
00367           goto exit_on_error;
00368         }
00369       if (variable->rangemax < variable->rangemin)
00370         {
00371           variable_error (variable, gettext ("bad range"));
00372           goto exit_on_error;
00373         }
00374     }
00375   else
00376     {
00377       variable_error (variable, gettext ("no maximum range"));
00378       goto exit_on_error;
00379     }
00380   variable->precision
00381     = json_object_get_uint_with_default (object,
    LABEL_PRECISION,
00382                                          DEFAULT_PRECISION, &error_code);
00383   if (error_code || variable->precision >= NPRECISIONS)
00384     {
00385       variable_error (variable, gettext ("bad precision"));
00386       goto exit_on_error;
00387     }
00388   if (algorithm == ALGORITHM_SWEEP)
00389     {
00390       if (json_object_get_member (object, LABEL_NSWEEPS))
00391         {
00392           variable->nsweeps
00393             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00394           if (error_code || !variable->nsweeps)
00395             {
00396               variable_error (variable, gettext ("bad sweeps"));
00397               goto exit_on_error;
00398             }
```

```
00399         }
00400       else
00401         {
00402           variable_error (variable, gettext ("no sweeps number"));
00403           goto exit_on_error;
00404         }
00405 #if DEBUG_VARIABLE
00406       fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00407 #endif
00408     }
00409   if (algorithm == ALGORITHM_GENETIC)
00410     {
00411       // Obtaining bits representing each variable
00412       if (json_object_get_member (object, LABEL_NBITS))
00413         {
00414           variable->nbits
00415             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00416           if (error_code || !variable->nbits)
00417             {
00418               variable_error (variable, gettext ("invalid bits number"));
00419               goto exit_on_error;
00420             }
00421         }
00422       else
00423         {
00424           variable_error (variable, gettext ("no bits number"));
00425           goto exit_on_error;
00426         }
00427     }
00428   else if (nsteps)
00429     {
00430       variable->step = json_object_get_float (object,
      LABEL_STEP, &error_code);
00431       if (error_code || variable->step < 0.)
00432         {
00433           variable_error (variable, gettext ("bad step size"));
00434           goto exit_on_error;
00435         }
00436     }
00437
00438 #if DEBUG_VARIABLE
00439   fprintf (stderr, "variable_open_json: end\n");
00440 #endif
00441   return 1;
00442 exit_on_error:
00443   variable_free (variable, INPUT_TYPE_JSON);
00444 #if DEBUG_VARIABLE
00445   fprintf (stderr, "variable_open_json: end\n");
00446 #endif
00447   return 0;
00448 }
```

Here is the call graph for this function:

5.27.3.5 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 136 of file variable.c.

```
00138 {
00139   int error_code;
00140
00141 #if DEBUG_VARIABLE
00142   fprintf (stderr, "variable_open_xml: start\n");
```

```
00143 #endif
00144
00145   variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146   if (!variable->name)
00147     {
00148       variable_error (variable, gettext ("no name"));
00149       goto exit_on_error;
00150     }
00151   if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153       variable->rangemin
00154         = xml_node_get_float (node, (const xmlChar *)
      LABEL_MINIMUM,
00155                               &error_code);
00156       if (error_code)
00157         {
00158           variable_error (variable, gettext ("bad minimum"));
00159           goto exit_on_error;
00160         }
00161       variable->rangeminabs = xml_node_get_float_with_default
00162         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163          &error_code);
00164       if (error_code)
00165         {
00166           variable_error (variable, gettext ("bad absolute minimum"));
00167           goto exit_on_error;
00168         }
00169       if (variable->rangemin < variable->rangeminabs)
00170         {
00171           variable_error (variable, gettext ("minimum range not allowed"));
00172           goto exit_on_error;
00173         }
00174     }
00175   else
00176     {
00177       variable_error (variable, gettext ("no minimum range"));
00178       goto exit_on_error;
00179     }
00180   if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181     {
00182       variable->rangemax
00183         = xml_node_get_float (node, (const xmlChar *)
      LABEL_MAXIMUM,
00184                               &error_code);
00185       if (error_code)
00186         {
00187           variable_error (variable, gettext ("bad maximum"));
00188           goto exit_on_error;
00189         }
00190       variable->rangemaxabs = xml_node_get_float_with_default
00191         (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192          &error_code);
00193       if (error_code)
00194         {
00195           variable_error (variable, gettext ("bad absolute maximum"));
00196           goto exit_on_error;
00197         }
00198       if (variable->rangemax > variable->rangemaxabs)
00199         {
00200           variable_error (variable, gettext ("maximum range not allowed"));
00201           goto exit_on_error;
00202         }
00203       if (variable->rangemax < variable->rangemin)
00204         {
00205           variable_error (variable, gettext ("bad range"));
00206           goto exit_on_error;
00207         }
00208     }
00209   else
00210     {
00211       variable_error (variable, gettext ("no maximum range"));
00212       goto exit_on_error;
00213     }
00214   variable->precision
00215     = xml_node_get_uint_with_default (node, (const xmlChar *)
      LABEL_PRECISION,
00216                                       DEFAULT_PRECISION, &error_code);
00217   if (error_code || variable->precision >= NPRECISIONS)
00218     {
00219       variable_error (variable, gettext ("bad precision"));
00220       goto exit_on_error;
00221     }
00222   if (algorithm == ALGORITHM_SWEEP)
00223     {
00224       if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225         {
00226           variable->nsweeps
```

```
00227             = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NSWEEPS,
00228                                  &error_code);
00229           if (error_code || !variable->nsweeps)
00230             {
00231               variable_error (variable, gettext ("bad sweeps"));
00232               goto exit_on_error;
00233             }
00234         }
00235       else
00236         {
00237           variable_error (variable, gettext ("no sweeps number"));
00238           goto exit_on_error;
00239         }
00240 #if DEBUG_VARIABLE
00241       fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243     }
00244   if (algorithm == ALGORITHM_GENETIC)
00245     {
00246       // Obtaining bits representing each variable
00247       if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248         {
00249           variable->nbits
00250             = xml_node_get_uint (node, (const xmlChar *)
      LABEL_NBITS,
00251                                  &error_code);
00252           if (error_code || !variable->nbits)
00253             {
00254               variable_error (variable, gettext ("invalid bits number"));
00255               goto exit_on_error;
00256             }
00257         }
00258       else
00259         {
00260           variable_error (variable, gettext ("no bits number"));
00261           goto exit_on_error;
00262         }
00263     }
00264   else if (nsteps)
00265     {
00266       variable->step
00267         = xml_node_get_float (node, (const xmlChar *)
      LABEL_STEP, &error_code);
00268       if (error_code || variable->step < 0.)
00269         {
00270           variable_error (variable, gettext ("bad step size"));
00271           goto exit_on_error;
00272         }
00273     }
00274
00275 #if DEBUG_VARIABLE
00276   fprintf (stderr, "variable_open_xml: end\n");
00277 #endif
00278   return 1;
00279 exit_on_error:
00280   variable_free (variable, INPUT_TYPE_XML);
00281 #if DEBUG_VARIABLE
00282   fprintf (stderr, "variable_open_xml: end\n");
00283 #endif
00284   return 0;
00285 }
```

Here is the call graph for this function:

5.28 variable.h

```
00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
```

```
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00038 #ifndef VARIABLE__H
00039 #define VARIABLE__H 1
00040
00045 enum Algorithm
00046 {
00047   ALGORITHM_MONTE_CARLO = 0,
00048   ALGORITHM_SWEEP = 1,
00049   ALGORITHM_GENETIC = 2
00050 };
00051
00056 typedef struct
00057 {
00058   char *name;
00059   double rangemin;
00060   double rangemax;
00061   double rangeminabs;
00062   double rangemaxabs;
00063   double step;
00064   unsigned int precision;
00065   unsigned int nsweeps;
00066   unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable, unsigned int type);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open_xml (Variable * variable, xmlNode * node,
00077                        unsigned int algorithm, unsigned int nsteps);
00078 int variable_open_json (Variable * variable, JsonNode * node,
00079                         unsigned int algorithm, unsigned int nsteps);
00080
00081 #endif
```

Index