

MPCOTool

3.0.4

Generated by Doxygen 1.8.12

Contents

1	MPCOTool	1
2	Data Structure Index	11
2.1	Data Structures	11
3	File Index	13
3.1	File List	13
4	Data Structure Documentation	15
4.1	Experiment Struct Reference	15
4.1.1	Detailed Description	15
4.2	Input Struct Reference	15
4.2.1	Detailed Description	17
4.3	Optimize Struct Reference	17
4.3.1	Detailed Description	19
4.3.2	Field Documentation	19
4.3.2.1	thread_direction	19
4.4	Options Struct Reference	20
4.4.1	Detailed Description	20
4.5	ParallelData Struct Reference	20
4.5.1	Detailed Description	21
4.6	Running Struct Reference	21
4.6.1	Detailed Description	21
4.7	Variable Struct Reference	21
4.7.1	Detailed Description	22
4.8	Window Struct Reference	22
4.8.1	Detailed Description	27

5	File Documentation	29
5.1	config.h File Reference	29
5.2	config.h	29
5.3	experiment.c File Reference	30
5.3.1	Detailed Description	31
5.3.2	Function Documentation	31
5.3.2.1	experiment_error()	31
5.3.2.2	experiment_free()	32
5.3.2.3	experiment_new()	32
5.3.2.4	experiment_open_json()	33
5.3.2.5	experiment_open_xml()	34
5.3.3	Variable Documentation	36
5.3.3.1	template	36
5.4	experiment.c	37
5.5	experiment.h File Reference	40
5.5.1	Detailed Description	41
5.5.2	Function Documentation	41
5.5.2.1	experiment_error()	41
5.5.2.2	experiment_free()	41
5.5.2.3	experiment_new()	43
5.5.2.4	experiment_open_json()	43
5.5.2.5	experiment_open_xml()	45
5.6	experiment.h	47
5.7	input.c File Reference	47
5.7.1	Detailed Description	48
5.7.2	Function Documentation	49
5.7.2.1	input_error()	49
5.7.2.2	input_open()	49
5.7.2.3	input_open_json()	50
5.7.2.4	input_open_xml()	55

5.8	input.c	60
5.9	input.h File Reference	72
5.9.1	Detailed Description	73
5.9.2	Enumeration Type Documentation	73
5.9.2.1	DirectionMethod	73
5.9.2.2	ErrorNorm	74
5.9.3	Function Documentation	74
5.9.3.1	input_error()	74
5.9.3.2	input_open()	75
5.9.3.3	input_open_json()	76
5.9.3.4	input_open_xml()	81
5.10	input.h	86
5.11	interface.c File Reference	87
5.11.1	Detailed Description	90
5.11.2	Function Documentation	90
5.11.2.1	input_save()	90
5.11.2.2	input_save_direction_json()	91
5.11.2.3	input_save_direction_xml()	92
5.11.2.4	input_save_json()	92
5.11.2.5	input_save_xml()	95
5.11.2.6	window_get_algorithm()	97
5.11.2.7	window_get_direction()	98
5.11.2.8	window_get_norm()	98
5.11.2.9	window_read()	98
5.11.2.10	window_save()	100
5.11.2.11	window_template_experiment()	102
5.12	interface.c	103
5.13	interface.h File Reference	134
5.13.1	Detailed Description	137
5.13.2	Function Documentation	137

5.13.2.1	<code>gtk_array_get_active()</code>	137
5.13.2.2	<code>input_save()</code>	137
5.13.2.3	<code>window_get_algorithm()</code>	138
5.13.2.4	<code>window_get_direction()</code>	139
5.13.2.5	<code>window_get_norm()</code>	139
5.13.2.6	<code>window_read()</code>	139
5.13.2.7	<code>window_save()</code>	141
5.13.2.8	<code>window_template_experiment()</code>	143
5.14	<code>interface.h</code>	144
5.15	<code>main.c</code> File Reference	147
5.15.1	Detailed Description	147
5.16	<code>main.c</code>	148
5.17	<code>optimize.c</code> File Reference	150
5.17.1	Detailed Description	153
5.17.2	Function Documentation	153
5.17.2.1	<code>optimize_best()</code>	153
5.17.2.2	<code>optimize_best_direction()</code>	154
5.17.2.3	<code>optimize_direction_sequential()</code>	154
5.17.2.4	<code>optimize_direction_thread()</code>	155
5.17.2.5	<code>optimize_estimate_direction_coordinates()</code>	156
5.17.2.6	<code>optimize_estimate_direction_random()</code>	156
5.17.2.7	<code>optimize_genetic_objective()</code>	157
5.17.2.8	<code>optimize_input()</code>	158
5.17.2.9	<code>optimize_merge()</code>	159
5.17.2.10	<code>optimize_norm_euclidian()</code>	160
5.17.2.11	<code>optimize_norm_maximum()</code>	160
5.17.2.12	<code>optimize_norm_p()</code>	161
5.17.2.13	<code>optimize_norm_taxicab()</code>	162
5.17.2.14	<code>optimize_parse()</code>	162
5.17.2.15	<code>optimize_save_variables()</code>	164

5.17.2.16	optimize_step_direction()	164
5.17.2.17	optimize_thread()	166
5.18	optimize.c	167
5.19	optimize.h File Reference	185
5.19.1	Detailed Description	187
5.19.2	Function Documentation	187
5.19.2.1	optimize_best()	187
5.19.2.2	optimize_best_direction()	188
5.19.2.3	optimize_direction_sequential()	189
5.19.2.4	optimize_direction_thread()	189
5.19.2.5	optimize_estimate_direction_coordinates()	190
5.19.2.6	optimize_estimate_direction_random()	191
5.19.2.7	optimize_genetic_objective()	191
5.19.2.8	optimize_input()	192
5.19.2.9	optimize_merge()	193
5.19.2.10	optimize_norm_euclidian()	194
5.19.2.11	optimize_norm_maximum()	195
5.19.2.12	optimize_norm_p()	195
5.19.2.13	optimize_norm_taxicab()	196
5.19.2.14	optimize_parse()	197
5.19.2.15	optimize_save_variables()	198
5.19.2.16	optimize_step_direction()	199
5.19.2.17	optimize_thread()	200
5.20	optimize.h	201
5.21	utils.c File Reference	203
5.21.1	Detailed Description	204
5.21.2	Function Documentation	204
5.21.2.1	cores_number()	204
5.21.2.2	gtk_array_get_active()	205
5.21.2.3	json_object_get_float()	205

5.21.2.4	json_object_get_float_with_default()	206
5.21.2.5	json_object_get_int()	206
5.21.2.6	json_object_get_uint()	207
5.21.2.7	json_object_get_uint_with_default()	208
5.21.2.8	json_object_set_float()	208
5.21.2.9	json_object_set_int()	209
5.21.2.10	json_object_set_uint()	209
5.21.2.11	show_error()	210
5.21.2.12	show_message()	210
5.21.2.13	xml_node_get_float()	211
5.21.2.14	xml_node_get_float_with_default()	211
5.21.2.15	xml_node_get_int()	212
5.21.2.16	xml_node_get_uint()	213
5.21.2.17	xml_node_get_uint_with_default()	213
5.21.2.18	xml_node_set_float()	214
5.21.2.19	xml_node_set_int()	214
5.21.2.20	xml_node_set_uint()	215
5.22	utils.c	215
5.23	utils.h File Reference	219
5.23.1	Detailed Description	221
5.23.2	Function Documentation	221
5.23.2.1	cores_number()	221
5.23.2.2	gtk_array_get_active()	221
5.23.2.3	json_object_get_float()	222
5.23.2.4	json_object_get_float_with_default()	223
5.23.2.5	json_object_get_int()	223
5.23.2.6	json_object_get_uint()	224
5.23.2.7	json_object_get_uint_with_default()	225
5.23.2.8	json_object_set_float()	225
5.23.2.9	json_object_set_int()	226

5.23.2.10 json_object_set_uint()	226
5.23.2.11 show_error()	227
5.23.2.12 show_message()	227
5.23.2.13 xml_node_get_float()	228
5.23.2.14 xml_node_get_float_with_default()	228
5.23.2.15 xml_node_get_int()	229
5.23.2.16 xml_node_get_uint()	230
5.23.2.17 xml_node_get_uint_with_default()	230
5.23.2.18 xml_node_set_float()	231
5.23.2.19 xml_node_set_int()	231
5.23.2.20 xml_node_set_uint()	232
5.24 utils.h	232
5.25 variable.c File Reference	233
5.25.1 Detailed Description	234
5.25.2 Function Documentation	234
5.25.2.1 variable_error()	234
5.25.2.2 variable_free()	235
5.25.2.3 variable_new()	235
5.25.2.4 variable_open_json()	236
5.25.2.5 variable_open_xml()	238
5.25.3 Variable Documentation	240
5.25.3.1 format	240
5.25.3.2 precision	241
5.26 variable.c	241
5.27 variable.h File Reference	246
5.27.1 Detailed Description	246
5.27.2 Enumeration Type Documentation	247
5.27.2.1 Algorithm	247
5.27.3 Function Documentation	248
5.27.3.1 variable_error()	248
5.27.3.2 variable_free()	248
5.27.3.3 variable_new()	249
5.27.3.4 variable_open_json()	249
5.27.3.5 variable_open_xml()	251
5.28 variable.h	254

Chapter 1

MPCOTool

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

VERSIONS

- 3.0.4: Stable and recommended version.
- 3.1.4: 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:

- 3.0.4/configure.ac: configure generator.
- 3.0.4/Makefile.in: Makefile generator.
- 3.0.4/config.h.in: config header generator.
- 3.0.4/mpcotool.c: main source code.
- 3.0.4/mpcotool.h: main header code.
- 3.0.4/mpcotool.ico: icon file.
- 3.0.4/interface.h: interface header code.
- 3.0.4/build: script to build all.
- 3.0.4/logo.png: logo figure.
- 3.0.4/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 4.6

Dyson Illumos

FreeBSD 11.0

Linux Mint DE 2

OpenSUSE Linux Tumbleweed

Ubuntu Linux 16.10

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/3.0.4  
$ ln -s ../../genetic/2.0.1 genetic
```

4. Build doing on a terminal:

```
$ ./build
```

Fedora Linux 25

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.

Microsoft Windows 7

Microsoft Windows 8.1

Microsoft Windows 10

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
```

NetBSD 7.0

1. MPI does not work. Follow steps 1 to 3 of the previous Debian 8 section and do in the terminal:

```
$ CC=/usr/pkg/gcc5/bin/gcc ./build
```

OpenBSD 6.0

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.

OpenIndiana Hipster

1. In order to use OpenMPI compilation do in a terminal:

```
$ export PATH=/usr/lib/openmpi/gcc/bin:$PATH
```

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/3.0.4):

```
$ 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 ../3.0.4
$ 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 objective 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 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:

$$(\text{number of experiments}) \times (\text{variable 1 number of sweeps}) \times \dots \times (\text{variable n number of sweeps}) \times (\text{number of iterations})$$
- **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:

$$(\text{number of experiments}) \times (\text{number of simulations}) \times (\text{number of iterations})$$
- 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:

$$\times (\text{number of iterations})$$
- 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:

$$(\text{number of experiments}) \times (\text{number of iterations}) \times (\text{number of steps}) \times 2 \times (\text{number of variables})$$
 - * *random*: random method. It requires:
 - * *nestimates*: number of random checks to estimate the optimal direction.
It increases the total number of simulations by:

$$(\text{number of experiments}) \times (\text{number of iterations}) \times (\text{number of steps}) \times (\text{number of estimates})$$

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 norms 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:

```
{
  "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: $4 \times 5 \times 5 \times 5 = 2500$.
- The input file is:

```
<?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*:

```
{
  "towers" :
  [
    {
      "length"      : 50.11,
      "velocity"    : 0.02738,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"      : 50.11,
      "velocity"    : 0.02824,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"      : 50.11,
      "velocity"    : 0.03008,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    },
    {
      "length"      : 50.11,
      "velocity"    : 0.03753,
      "@variable1@" : @value1@,
      "@variable2@" : @value2@,
      "@variable3@" : @value3@,
      "@variable4@" : @value4@
    }
  ],
  "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:

```
{
  "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:

Experiment	Struct to define the experiment data	15
Input	Struct to define the optimization input file	15
Optimize	Struct to define the optimization ation data	17
Options	Struct to define the options dialog	20
ParallelData	Struct to pass to the GThreads parallelized function	20
Running	Struct to define the running dialog	21
Variable	Struct to define the variable data	21
Window	Struct to define the main window	22

Chapter 3

File Index

3.1 File List

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

config.h	Configuration header file	29
experiment.c	Source file to define the experiment data	30
experiment.h	Header file to define the experiment data	40
generate.c	??
input.c	Source file to define the input functions	47
input.h	Header file to define the input functions	72
interface.c	Source file to define the graphical interface functions	87
interface.h	Header file to define the graphical interface functions	134
main.c	Main source file	147
optimize.c	Source file to define the optimization functions	150
optimize.h	Header file to define the optimization functions	185
utils.c	Source file to define some useful functions	203
utils.h	Header file to define some useful functions	219
variable.c	Source file to define the variable data	233
variable.h	Header file to define the variable data	246

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](#)

- unsigned int [norm](#)
Number of best simulations.
- unsigned int [type](#)
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 thread_direction

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

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

Definition at line 80 of file [optimize.h](#).

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

- [optimize.h](#)

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

- GtkWidget * [window](#)
Main GtkWidget.
- GtkWidget * [grid](#)
Main GtkWidget.
- GtkWidget * [bar_buttons](#)
GtkToolbar to store the main buttons.
- GtkWidget * [button_open](#)
Open GtkWidget.
- GtkWidget * [button_save](#)
Save GtkWidget.
- GtkWidget * [button_run](#)
Run GtkWidget.
- GtkWidget * [button_options](#)
Options GtkWidget.
- GtkWidget * [button_help](#)
Help GtkWidget.
- GtkWidget * [button_about](#)
Help GtkWidget.
- GtkWidget * [button_exit](#)
Exit GtkWidget.
- GtkWidget * [grid_files](#)
Files GtkWidget.
- GtkWidget * [label_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [button_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [check_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [button_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [label_result](#)
Result file GtkWidget.
- GtkWidget * [entry_result](#)
Result file GtkWidget.
- GtkWidget * [label_variables](#)
Variables file GtkWidget.
- GtkWidget * [entry_variables](#)
Variables file GtkWidget.
- GtkWidget * [frame_norm](#)
GtkFrame to set the error norm.
- GtkWidget * [grid_norm](#)
GtkWidget to set the error norm.
- GtkWidget * [button_norm](#) [NNORMS]
Array of GtkWidget to set the error norm.
- GtkWidget * [label_p](#)
GtkWidget to set the p parameter.
- GtkWidget * [spin_p](#)
GtkSpinButton to set the p parameter.
- GtkWidget * [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_best`
GtkLabel to set the best number.
 - `GtkSpinButton * spin_best`
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
00033 #ifndef CONFIG__H
00034 #define CONFIG__H 1
00035
00036 // Array sizes
00037
00038 #define MAX_NINPUTS 8
00039 #define NALGORITHMS 3
00040 #define NDIRECTIONS 2
00041 #define NNORMS 4
00042 #define NPRECISIONS 15
00043
```

```

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"
00066
00067 #define LABEL_ADAPTATION "adaptation"
00068 #define LABEL_ALGORITHM "algorithm"
00069 #define LABEL_OPTIMIZE "optimize"
00070 #define LABEL_COORDINATES "coordinates"
00071 #define LABEL_DIRECTION "direction"
00072 #define LABEL_EUCLIDIAN "euclidian"
00073 #define LABEL_EVALUATOR "evaluator"
00074 #define LABEL_EXPERIMENT "experiment"
00075 #define LABEL_EXPERIMENTS "experiments"
00076 #define LABEL_GENETIC "genetic"
00077 #define LABEL_MINIMUM "minimum"
00078 #define LABEL_MAXIMUM "maximum"
00079 #define LABEL_MONTE_CARLO "Monte-Carlo"
00080 #define LABEL_MUTATION "mutation"
00081 #define LABEL_NAME "name"
00082 #define LABEL_NBEST "nbest"
00083 #define LABEL_NBITS "nbits"
00084 #define LABEL_NESTIMATES "nestimates"
00085 #define LABEL_NGENERATIONS "ngenerations"
00086 #define LABEL_NITERATIONS "niterations"
00087 #define LABEL_NORM "norm"
00088 #define LABEL_NPOPULATION "npopulation"
00089 #define LABEL_NSIMULATIONS "nsimulations"
00090 #define LABEL_NSTEPS "nsteps"
00091 #define LABEL_NSWEEPS "nsweeps"
00092 #define LABEL_P "p"
00093 #define LABEL_PRECISION "precision"
00094 #define LABEL_RANDOM "random"
00095 #define LABEL_RELAXATION "relaxation"
00096 #define LABEL_REPRODUCTION "reproduction"
00097 #define LABEL_RESULT_FILE "result_file"
00098 #define LABEL_SIMULATOR "simulator"
00099 #define LABEL_SEED "seed"
00100 #define LABEL_STEP "step"
00101 #define LABEL_SWEEP "sweep"
00102 #define LABEL_TAXICAB "taxicab"
00103 #define LABEL_TEMPLATE1 "template1"
00104 #define LABEL_TEMPLATE2 "template2"
00105 #define LABEL_TEMPLATE3 "template3"
00106 #define LABEL_TEMPLATE4 "template4"
00107 #define LABEL_TEMPLATE5 "template5"
00108 #define LABEL_TEMPLATE6 "template6"
00109 #define LABEL_TEMPLATE7 "template7"
00110 #define LABEL_TEMPLATE8 "template8"
00111 #define LABEL_THRESHOLD "threshold"
00112 #define LABEL_TOLERANCE "tolerance"
00113 #define LABEL_VARIABLE "variable"
00114 #define LABEL_VARIABLES "variables"
00115 #define LABEL_VARIABLES_FILE "variables_file"
00116 #define LABEL_WEIGHT "weight"
00117
00118 // Enumerations
00119
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

Copyright 2012-2016, all rights reserved.

Definition in file `experiment.c`.

5.3.2 Function Documentation

5.3.2.1 `experiment_error()`

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

Function to print a message error opening an `Experiment` struct.

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	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 `experiment_free()`

```

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

Function to free the memory of an [Experiment](#) struct.

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	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 `experiment_new()`

```

void experiment_new (
    Experiment * experiment )
```

Function to create a new [Experiment](#) struct.

Parameters

<i>experiment</i>	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 experiment_open_json()

```

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

Function to open the [Experiment](#) struct on a XML node.

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	JSON node.
<i>ninputs</i>	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,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291     #if DEBUG_EXPERIMENT
00292     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00293     #endif
00294     name = json_object_get_string_member (object, template[0]);
00295     if (name)
00296     {
00297         #if DEBUG_EXPERIMENT
00298         fprintf (stderr, "experiment_open_json: experiment=%s templatel=%s\n",
00299         name, template[0]);
00300         #endif
00301         ++experiment->ninputs;
00302     }
00303     else
00304     {
00305         experiment_error (experiment, gettext ("no template"));
00306         goto exit_on_error;
00307     }
00308     experiment->template[0] = g_strdup (name);
00309     for (i = 1; i < MAX_NINPUTS; ++i)
00310     {
00311         #if DEBUG_EXPERIMENT
00312         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00313         #endif
00314         if (json_object_get_member (object, template[i]))
00315         {
00316             if (ninputs && ninputs <= i)
00317             {
00318                 experiment_error (experiment, gettext ("bad templates number"));
00319                 goto exit_on_error;
00320             }
00321             name = json_object_get_string_member (object, template[i]);
00322             #if DEBUG_EXPERIMENT
00323             fprintf (stderr,
00324             "experiment_open_json: experiment=%s template%u=%s\n",
00325             experiment->nexperiments, name, template[i]);
00326             #endif
00327             experiment->template[i] = g_strdup (name);
00328             ++experiment->ninputs;
00329         }
00330         else if (ninputs && ninputs > i)
00331         {
00332             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00333             experiment_error (experiment, buffer);
00334             goto exit_on_error;
00335         }
00336         else
00337             break;
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 experiment_open_xml()

```
int experiment_open_xml (
```

```

Experiment * experiment,
xmlNode * node,
unsigned int ninputs )

```

Function to open the [Experiment](#) struct on a XML node.

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	XML node.
<i>ninputs</i>	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 template

```
const char* template[MAX_NINPUTS]
```

Initial value:

```

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

```

Array of xmlChar strings with template labels.

Definition at line 50 of file [experiment.c](#).

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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "experiment.h"
00041
00042 #define DEBUG_EXPERIMENT 0
00043
00044 const char *template[MAX_NINPUTS] = {
00045     LABEL_TEMPLATE1, LABEL_TEMPLATE2,
00046     LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00047     LABEL_TEMPLATE5, LABEL_TEMPLATE6,
00048     LABEL_TEMPLATE7, LABEL_TEMPLATE8
00049 };
00050
00051 void
00052 experiment_new (Experiment * experiment)
00053 {
00054     unsigned int i;
00055     #if DEBUG_EXPERIMENT
00056     fprintf (stderr, "experiment_new: start\n");
00057     #endif
00058     experiment->name = NULL;
00059     experiment->ninputs = 0;
00060     for (i = 0; i < MAX_NINPUTS; ++i)
00061         experiment->template[i] = NULL;
00062     #if DEBUG_EXPERIMENT
00063     fprintf (stderr, "input_new: end\n");
00064     #endif
00065 }
00066
00067 void
00068 experiment_free (Experiment * experiment, unsigned int type)
00069 {
00070     unsigned int i;
00071     #if DEBUG_EXPERIMENT
00072     fprintf (stderr, "experiment_free: start\n");
00073     #endif
00074     if (type == INPUT_TYPE_XML)
00075     {
00076         for (i = 0; i < experiment->ninputs; ++i)
00077             xmlFree (experiment->template[i]);
00078         xmlFree (experiment->name);
00079     }
00080     else
00081     {
00082         for (i = 0; i < experiment->ninputs; ++i)
00083             g_free (experiment->template[i]);
00084     }
00085 }

```

```

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 *)
00171     LABEL_WEIGHT, 1.,
00172                                     &error_code);
00173     if (error_code)
00174     {
00175         experiment_error (experiment, gettext ("bad weight"));
00176         goto exit_on_error;
00177     }
00178     #if DEBUG_EXPERIMENT
00179     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180     #endif
00181     experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183     if (experiment->template[0])
00184     {
00185         #if DEBUG_EXPERIMENT
00186         fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00187                 experiment->name, template[0]);
00188         #endif
00189         ++experiment->ninputs;
00190     }
00191     else
00192     {
00193         experiment_error (experiment, gettext ("no template"));
00194         goto exit_on_error;
00195     }
00196     for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198         #if DEBUG_EXPERIMENT
00199         fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200         #endif
00201         if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203             if (ninputs && ninputs <= i)
00204             {
00205                 experiment_error (experiment, gettext ("bad templates number"));
00206                 goto exit_on_error;
00207             }
00208             experiment->template[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210             #if DEBUG_EXPERIMENT

```



```

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,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291 #if DEBUG_EXPERIMENT
00292     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00293 #endif
00294     name = json_object_get_string_member (object, template[0]);
00295     if (name)
00296     {
00297         #if DEBUG_EXPERIMENT
00298             fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00299                     name, template[0]);
00300         #endif
00301         ++experiment->ninputs;
00302     }
00303     else
00304     {
00305         experiment_error (experiment, gettext ("no template"));
00306         goto exit_on_error;
00307     }
00308     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

Copyright 2012-2016, all rights reserved.

Definition in file [experiment.h](#).

5.5.2 Function Documentation

5.5.2.1 `experiment_error()`

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

Function to print a message error opening an [Experiment](#) struct.

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	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 `experiment_free()`

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

Function to free the memory of an [Experiment](#) struct.

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	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 [experiment_new\(\)](#)

```

void experiment_new (
    Experiment * experiment )

```

Function to create a new [Experiment](#) struct.

Parameters

<i>experiment</i>	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 [experiment_open_json\(\)](#)

```

int experiment_open_json (
    Experiment * experiment,

```

```

    JsonNode * node,
    unsigned int ninputs )

```

Function to open the [Experiment](#) struct on a XML node.

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	JSON node.
<i>ninputs</i>	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,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291     #if DEBUG_EXPERIMENT
00292     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00293     #endif
00294     name = json_object_get_string_member (object, template[0]);
00295     if (name)
00296     {
00297         #if DEBUG_EXPERIMENT
00298         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00299         name, template[0]);
00300         #endif
00301         ++experiment->ninputs;
00302     }
00303     else
00304     {
00305         experiment_error (experiment, gettext ("no template"));
00306         goto exit_on_error;
00307     }
00308     experiment->template[0] = g_strdup (name);
00309     for (i = 1; i < MAX_NINPUTS; ++i)
00310     {
00311         #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 experiment_open_xml()

```

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

```

Function to open the [Experiment](#) struct on a XML node.

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	XML node.
<i>ninputs</i>	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 *)
00171     LABEL_WEIGHT, 1.,
00172     &error_code);
00173     if (error_code)
00174     {
00175         experiment_error (experiment, gettext ("bad weight"));
00176         goto exit_on_error;
00177     }
00178     #if DEBUG_EXPERIMENT
00179     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00180     #endif
00181     experiment->template[0]
00182     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00183     if (experiment->template[0])
00184     {
00185         #if DEBUG_EXPERIMENT
00186         fprintf (stderr, "experiment_open_xml: experiment=%s templatel=%s\n",
00187         experiment->name, template[0]);
00188         #endif
00189         ++experiment->ninputs;
00190     }
00191     else
00192     {
00193         experiment_error (experiment, gettext ("no template"));
00194         goto exit_on_error;
00195     }
00196     for (i = 1; i < MAX_NINPUTS; ++i)
00197     {
00198         #if DEBUG_EXPERIMENT
00199         fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00200         #endif
00201         if (xmlHasProp (node, (const xmlChar *) template[i]))
00202         {
00203             if (ninputs && ninputs <= i)
00204             {
00205                 experiment_error (experiment, gettext ("bad templates number"));
00206                 goto exit_on_error;
00207             }
00208             experiment->template[i]
00209             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00210             #if DEBUG_EXPERIMENT
00211             fprintf (stderr, "experiment_open_xml: experiment=%s template%u=%s\n",
00212             experiment->nexperiments, experiment->name,
00213             experiment->template[i]);
00214             #endif
00215             ++experiment->ninputs;
00216         }
00217         else if (ninputs && ninputs > i)
00218         {
00219             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00220             experiment_error (experiment, buffer);
00221             goto exit_on_error;
00222         }
00223         else
00224             break;
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
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
00037     char *name;
00038     char *template[MAX_NINPUTS];
00039     double weight;
00040     unsigned int ninputs;
00041 } Experiment;
00042
00043 extern const char *template[MAX_NINPUTS];
00044
00045 // Public functions
00046 void experiment_new (Experiment * experiment);
00047 void experiment_free (Experiment * experiment, unsigned int type);
00048 void experiment_error (Experiment * experiment, char *message);
00049 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00050                         unsigned int ninputs);
00051 int experiment_open_json (Experiment * experiment, JsonNode * node,
00052                          unsigned int ninputs);
00053
00054 #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 input_error()

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

Function to print an error message opening an [Input](#) struct.

Parameters

<i>message</i>	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 input_open()

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

Function to open the input file.

Parameters

<i>filename</i>	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 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Parameters

<i>parser</i>	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,
00617 LABEL_SEED,
00618                                     DEFAULT_RANDOM_SEED, &error_code);
00619     if (error_code)
00620     {
00621         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00622         goto exit_on_error;
00623     }
00624
00625     // Opening algorithm
00626     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00627     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00628     {
00629         input->algorithm = ALGORITHM_MONTE_CARLO;
00630     }
00631     // Obtaining simulations number
00632     input->nsimulations
00633     = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code);
00634     if (error_code)
00635     {
00636         input_error (gettext ("Bad simulations number"));
00637         goto exit_on_error;
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,
00650 LABEL_NPOPULATION, &error_code);
00651         if (error_code || input->nsimulations < 3)
00652         {
00653             input_error (gettext ("Invalid population number"));
00654             goto exit_on_error;
00655         }
00656     }
00657     else
00658     {

```

```

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,
00667 LABEL_NGENERATIONS, &error_code);
00668         if (error_code || !input->niterations)
00669         {
00670             input_error (gettext ("Invalid generations number"));
00671             goto exit_on_error;
00672         }
00673     }
00674     else
00675     {
00676         input_error (gettext ("No generations number"));
00677         goto exit_on_error;
00678     }
00679
00680     // Obtaining mutation probability
00681     if (json_object_get_member (object, LABEL_MUTATION))
00682     {
00683         input->mutation_ratio
00684         = json_object_get_float (object, LABEL_MUTATION, &error_code
00685 );
00686         if (error_code || input->mutation_ratio < 0.
00687 || input->mutation_ratio >= 1.)
00688         {
00689             input_error (gettext ("Invalid mutation probability"));
00690             goto exit_on_error;
00691         }
00692     }
00693     else
00694     {
00695         input_error (gettext ("No mutation probability"));
00696         goto exit_on_error;
00697     }
00698
00699     // Obtaining reproduction probability
00700     if (json_object_get_member (object, LABEL_REPRODUCTION))
00701     {
00702         input->reproduction_ratio
00703         = json_object_get_float (object,
00704 LABEL_REPRODUCTION, &error_code);
00705         if (error_code || input->reproduction_ratio < 0.
00706 || input->reproduction_ratio >= 1.0)
00707         {
00708             input_error (gettext ("Invalid reproduction probability"));
00709             goto exit_on_error;
00710         }
00711     }
00712     else
00713     {
00714         input_error (gettext ("No reproduction probability"));
00715         goto exit_on_error;
00716     }
00717
00718     // Obtaining adaptation probability
00719     if (json_object_get_member (object, LABEL_ADAPTATION))
00720     {
00721         input->adaptation_ratio
00722         = json_object_get_float (object,
00723 LABEL_ADAPTATION, &error_code);
00724         if (error_code || input->adaptation_ratio < 0.
00725 || input->adaptation_ratio >= 1.)
00726         {
00727             input_error (gettext ("Invalid adaptation probability"));
00728             goto exit_on_error;
00729         }
00730     }
00731     else
00732     {
00733         input_error (gettext ("No adaptation probability"));
00734         goto exit_on_error;
00735     }
00736
00737     // Checking survivals
00738     i = input->mutation_ratio * input->nsimulations;
00739     i += input->reproduction_ratio * input->
00740 nsimulations;
00741     i += input->adaptation_ratio * input->
00742 nsimulations;
00743     if (i > input->nsimulations - 2)
00744     {

```

```

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     // Obtaining iterations number
00755     input->niterations
00756     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00757 );
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     // Obtaining best number
00766     input->nbest
00767     = json_object_get_uint_with_default (object,
00768 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     // Obtaining tolerance
00776     input->tolerance
00777     = json_object_get_float_with_default (object,
00778 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     // Getting direction search method parameters
00786     if (json_object_get_member (object, LABEL_NSTEPS))
00787     {
00788         input->nsteps
00789         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00790         if (error_code || !input->nsteps)
00791         {
00792             input_error (gettext ("Invalid steps number"));
00793             goto exit_on_error;
00794         }
00795         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00796         if (!strcmp (buffer, LABEL_COORDINATES))
00797             input->direction = DIRECTION_METHOD_COORDINATES;
00798         else if (!strcmp (buffer, LABEL_RANDOM))
00799         {
00800             input->direction = DIRECTION_METHOD_RANDOM;
00801             input->nestimates
00802             = json_object_get_uint (object,
00803 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     }
00811     else
00812     {
00813         input_error
00814         (gettext ("Unknown method to estimate the direction search"));
00815         goto exit_on_error;
00816     }
00817     input->relaxation
00818     = json_object_get_float_with_default (object,
00819 LABEL_RELAXATION,
00820                                         DEFAULT_RELAXATION,
00821                                         &error_code);
00822     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.,
                                &error_code);
00832
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,
00906 LABEL_P, &error_code);
00907         if (!error_code)
00908         {
00909             input_error (gettext ("Bad P parameter"));
00910             goto exit_on_error;
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 input_open_xml()

```

int input_open_xml (
    xmlDoc * doc )

```

Function to open the input file in XML format.

Parameters

<i>doc</i>	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,
                                DEFAULT_RANDOM_SEED, &error_code);
00199 if (error_code)
00200 {
00201     input_error (gettext ("Bad pseudo-random numbers generator seed"));
00202     goto exit_on_error;
00203 }
00204
00205 // Opening algorithm
00206 buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207 if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208 {
00209     input->algorithm = ALGORITHM_MONTE_CARLO;
00210 }
00211
00212 // Obtaining simulations number
00213 input->nsimulations
00214 = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                    &error_code);
00215 if (error_code)
00216 {
00217     input_error (gettext ("Bad simulations number"));
00218     goto exit_on_error;
00219 }
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
00229 // 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 *)
00251 LABEL_NGENERATIONS,
00252         &error_code);
00253     if (error_code || !input->niterations)
00254     {
00255         input_error (gettext ("Invalid generations number"));
00256         goto exit_on_error;
00257     }
00258 }
00259 else
00260 {
00261     input_error (gettext ("No generations number"));
00262     goto exit_on_error;
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 *)
00269 LABEL_MUTATION,
00270         &error_code);
00271     if (error_code || input->mutation_ratio < 0.
00272         || input->mutation_ratio >= 1.)
00273     {
00274         input_error (gettext ("Invalid mutation probability"));
00275         goto exit_on_error;
00276     }
00277 }
00278 else
00279 {
00280     input_error (gettext ("No mutation probability"));
00281     goto exit_on_error;
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 *)
00288 LABEL_REPRODUCTION,
00289         &error_code);
00290     if (error_code || input->reproduction_ratio < 0.
00291         || input->reproduction_ratio >= 1.0)
00292     {
00293         input_error (gettext ("Invalid reproduction probability"));
00294         goto exit_on_error;
00295     }
00296 }
00297 else
00298 {
00299     input_error (gettext ("No reproduction probability"));
00300     goto exit_on_error;
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 *)
00307 LABEL_ADAPTATION,
00308         &error_code);
00309     if (error_code || input->adaptation_ratio < 0.
00310         || input->adaptation_ratio >= 1.)
00311     {
00312         input_error (gettext ("Invalid adaptation probability"));
00313         goto exit_on_error;
00314     }
00315 }

```

```

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     // Obtaining iterations number
00345     input->niterations
00346     = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
00347                          &error_code);
00348     if (error_code == 1)
00349         input->niterations = 1;
00350     else if (error_code)
00351     {
00352         input_error (gettext ("Bad iterations number"));
00353         goto exit_on_error;
00354     }
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->
00418         relaxation > 2.)
00419     {
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 *)
00429     LABEL_THRESHOLD,
00430                                     0., &error_code);
00431 if (error_code)
00432 {
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 (
00448                 Experiment));
00449     if (!input->nexperiments)
00450     {
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 +
00456             input->nexperiments,
00457                                 child, input->experiment->
00458             ninputs))
00459             goto exit_on_error;
00460     }
00461     ++input->nexperiments;
00462 #if DEBUG_INPUT
00463     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00464             input->nexperiments);
00465 #endif
00466 }
00467 if (!input->nexperiments)
00468 {
00468     input_error (gettext ("No optimization experiments"));
00469     goto exit_on_error;
00470 }
00471 buffer = NULL;
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,
input->algorithm, input->nsteps))
00490         goto exit_on_error;
00491     ++input->nvariables;
00492 }
00493 if (!input->nvariables)
00494 {
00495     input_error (gettext ("No optimization variables"));
00496     goto exit_on_error;
00497 }
00498 buffer = NULL;
00499
00500 // Obtaining the error norm
00501 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00502 {
00503     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00504     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00505         input->norm = ERROR_NORM_EUCLIDIAN;
00506     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00507         input->norm = ERROR_NORM_MAXIMUM;
00508     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00509     {
00510         input->norm = ERROR_NORM_P;
00511         input->p
00512             = xml_node_get_float (node, (const xmlChar *)
LABEL_P, &error_code);
00513         if (!error_code)
00514         {
00515             input_error (gettext ("Bad P parameter"));
00516             goto exit_on_error;
00517         }
00518     }
00519     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00520         input->norm = ERROR_NORM_TAXICAB;
00521     else
00522     {
00523         input_error (gettext ("Unknown error norm"));
00524         goto exit_on_error;
00525     }
00526     xmlFree (buffer);
00527 }
00528 else
00529     input->norm = ERROR_NORM_EUCLIDIAN;
00530 // Closing the XML document
00531 xmlFreeDoc (doc);
00532
00533 #if DEBUG_INPUT
00534 fprintf (stderr, "input_open_xml: end\n");
00535 #endif
00536 return 1;
00537
00538 exit_on_error:
00539 xmlFree (buffer);
00540 xmlFreeDoc (doc);
00541 #if DEBUG_INPUT
00542 fprintf (stderr, "input_open_xml: end\n");
00543 #endif
00544 return 0;
00545 }

```

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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "utils.h"
00042 #include "experiment.h"
00043 #include "variable.h"
00044 #include "input.h"
00045
00046 #define DEBUG_INPUT 0
00047
00048 Input input[1];
00049
00050 const char *result_name = "result";
00051 const char *variables_name = "variables";
00052
00053 void
00054 input_new ()
00055 {
00056     #if DEBUG_INPUT
00057         fprintf (stderr, "input_new: start\n");
00058     #endif
00059     input->nvariables = input->nexperiments = input->nsteps = 0;
00060     input->simulator = input->evaluator = input->directory = input->
00061         name = NULL;
00062     input->experiment = NULL;
00063     input->variable = NULL;
00064     #if DEBUG_INPUT
00065         fprintf (stderr, "input_new: end\n");
00066     #endif
00067 }
00068
00069 void
00070 input_free ()
00071 {
00072     unsigned int i;
00073     #if DEBUG_INPUT
00074         fprintf (stderr, "input_free: start\n");
00075     #endif
00076     g_free (input->name);
00077     g_free (input->directory);
00078     for (i = 0; i < input->nexperiments; ++i)
00079         experiment_free (input->experiment + i, input->type);
00080     for (i = 0; i < input->nvariables; ++i)
00081         variable_free (input->variable + i, input->type);
00082     g_free (input->experiment);
00083     g_free (input->variable);
00084     if (input->type == INPUT_TYPE_XML)
00085     {
00086         xmlFree (input->evaluator);
00087         xmlFree (input->simulator);
00088         xmlFree (input->result);
00089         xmlFree (input->variables);
00090     }
00091 }

```

```

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
00117 void
00118 input_error (char *message)
00119 {
00120     char buffer[64];
00121     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00122     error_message = g_strdup (buffer);
00123 }
00124
00125 int
00126 input_open_xml (xmlDoc * doc)
00127 {
00128     char buffer2[64];
00129     xmlNode *node, *child;
00130     xmlChar *buffer;
00131     int error_code;
00132     unsigned int i;
00133
00134     #if DEBUG_INPUT
00135     fprintf (stderr, "input_open_xml: start\n");
00136     #endif
00137
00138     // Resetting input data
00139     buffer = NULL;
00140     input->type = INPUT_TYPE_XML;
00141
00142     // Getting the root node
00143     #if DEBUG_INPUT
00144     fprintf (stderr, "input_open_xml: getting the root node\n");
00145     #endif
00146     node = xmlDocGetRootElement (doc);
00147     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00148     {
00149         input_error (gettext ("Bad root XML node"));
00150         goto exit_on_error;
00151     }
00152
00153     // Getting result and variables file names
00154     if (!input->result)
00155     {
00156         input->result =
00157             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00158         if (!input->result)
00159             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00160     }
00161     if (!input->variables)
00162     {
00163         input->variables =
00164             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00165         if (!input->variables)
00166             input->variables =
00167                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00168     }
00169
00170     // Opening simulator program name
00171     input->simulator =
00172         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00173     if (!input->simulator)
00174     {
00175         input_error (gettext ("Bad simulator program"));
00176         goto exit_on_error;
00177     }
00178
00179     // Opening evaluator program name
00180     input->evaluator =
00181         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00182
00183     // Obtaining pseudo-random numbers generator seed
00184     input->seed
00185         = xml_node_get_uint_with_default (node, (const xmlChar *)
00186             LABEL_SEED,
00187             DEFAULT_RANDOM_SEED, &error_code);
00188     if (error_code)
00189     {
00190         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00191     }
00192 }

```



```

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.
|| input->mutation_ratio >= 1.)
00271         {
00272             input_error (gettext ("Invalid mutation probability"));
00273             goto exit_on_error;
00274         }
00275     }
00276     else
00277     {
00278         input_error (gettext ("No mutation probability"));
00279         goto exit_on_error;
00280     }
00281
00282     // Obtaining reproduction probability
00283     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00284     {
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 *)
00383                                LABEL_NSTEPS,
00384                                &error_code);
00385         if (error_code || !input->nsteps)
00386         {
00387             input_error (gettext ("Invalid steps number"));
00388             goto exit_on_error;
00389         }
00390         buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00391         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00392             input->direction = DIRECTION_METHOD_COORDINATES;
00393         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00394         {
00395             input->direction = DIRECTION_METHOD_RANDOM;
00396             input->nestimates
00397                 = xml_node_get_uint (node, (const xmlChar *)
00398                                        LABEL_NESTIMATES,
00399                                        &error_code);
00400             if (error_code || !input->nestimates)
00401             {
00402                 input_error (gettext ("Invalid estimates number"));
00403                 goto exit_on_error;
00404             }
00405         }
00406         else
00407         {
00408             input_error
00409                 (gettext ("Unknown method to estimate the direction search"));
00410             goto exit_on_error;
00411         }
00412         xmlFree (buffer);
00413         buffer = NULL;
00414         input->relaxation
00415             = xml_node_get_float_with_default (node,
00416                                                (const xmlChar *)
00417                                                LABEL_RELAXATION,
00418                                                DEFAULT_RELAXATION, &error_code);
00419         if (error_code || input->relaxation < 0. || input->
00420             relaxation > 2.)
00421         {
00422             input_error (gettext ("Invalid relaxation parameter"));
00423             goto exit_on_error;
00424         }
00425         else
00426             input->nsteps = 0;
00427     }
00428     // Obtaining the threshold
00429     input->threshold =
00430         xml_node_get_float_with_default (node, (const xmlChar *)
00431            LABEL_THRESHOLD,
00432            0., &error_code);
00433     if (error_code)
00434     {
00435         input_error (gettext ("Invalid threshold"));
00436         goto exit_on_error;
00437     }
00438
00439     // Reading the experimental data
00440     for (child = node->children; child; child = child->next)
00441     {
00442         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00443             break;
00444     }
00445     #if DEBUG_INPUT
00446     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00447             input->nexperiments);
00448     #endif
00449     input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                   (1 + input->nexperiments) * sizeof (Experiment));
00452     if (!input->nexperiments)
00453     {
00454         if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456     }

```

```

00452     }
00453     else
00454     {
00455         if (!experiment_open_xml (input->experiment + input->
n experiments,
00456                                     child, input->experiment->ninputs))
00457             goto exit_on_error;
00458     }
00459     ++input->n experiments;
00460 #if DEBUG_INPUT
00461     fprintf (stderr, "input_open_xml: n experiments=%u\n",
00462             input->n experiments);
00463 #endif
00464     }
00465     if (!input->n experiments)
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: n variables=%u\n", input->n variables);
00477 #endif
00478         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00479         {
00480             snprintf (buffer2, 64, "%s %u: %s",
00481                     gettext ("Variable"),
00482                     input->n variables + 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->n variables) * sizeof (Variable));
00489         if (!variable_open_xml (input->variable + input->
n variables, child,
00490                                 input->algorithm, input->n steps))
00491             goto exit_on_error;
00492         ++input->n variables;
00493     }
00494     if (!input->n variables)
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
00549 int
00550 input_open_json (JsonParser * parser)
00551 {
00552     JsonNode *node, *child;
00553     JsonObject *object;
00554     JsonArray *array;
00555     const char *buffer;
00556     int error_code;
00557     unsigned int i, n;
00558
00559 #if DEBUG_INPUT
00560     fprintf (stderr, "input_open_json: start\n");
00561 #endif
00562
00563     // Resetting input data
00564     input->type = INPUT_TYPE_JSON;
00565
00566     // Getting the root node
00567 #if DEBUG_INPUT
00568     fprintf (stderr, "input_open_json: getting the root node\n");
00569 #endif
00570     node = json_parser_get_root (parser);
00571     object = json_node_get_object (node);
00572
00573     // Getting result and variables file names
00574     if (!input->result)
00575     {
00576         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00577         if (!buffer)
00578             buffer = result_name;
00579         input->result = g_strdup (buffer);
00580     }
00581     else
00582         input->result = g_strdup (result_name);
00583     if (!input->variables)
00584     {
00585         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00586         if (!buffer)
00587             buffer = variables_name;
00588         input->variables = g_strdup (buffer);
00589     }
00590     else
00591         input->variables = g_strdup (variables_name);
00592
00593     // Opening simulator program name
00594     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00595     if (!buffer)
00596     {
00597         input_error (gettext ("Bad simulator program"));
00598         goto exit_on_error;
00599     }
00600     input->simulator = g_strdup (buffer);
00601
00602     // Opening evaluator program name
00603     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00604     if (!buffer)
00605         input->evaluator = g_strdup (buffer);
00606
00607     // Obtaining pseudo-random numbers generator seed
00608     input->seed
00609     = json_object_get_uint_with_default (object,
00610     LABEL_SEED,
00611     DEFAULT_RANDOM_SEED, &error_code);
00612     if (error_code)
00613     {
00614         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00615         goto exit_on_error;
00616     }
00617
00618     // Opening algorithm
00619     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00620     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00621     {
00622         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 // Obtaining generations
00662 if (json_object_get_member (object, LABEL_NGENERATIONS))
00663 {
00664     input->niterations
00665     = json_object_get_uint (object,
LABEL_NGENERATIONS, &error_code);
00666     if (error_code || !input->niterations)
00667     {
00668         input_error (gettext ("Invalid generations number"));
00669         goto exit_on_error;
00670     }
00671 }
00672 }
00673 else
00674 {
00675     input_error (gettext ("No generations number"));
00676     goto exit_on_error;
00677 }
00678 // Obtaining mutation probability
00679 if (json_object_get_member (object, LABEL_MUTATION))
00680 {
00681     input->mutation_ratio
00682     = json_object_get_float (object, LABEL_MUTATION, &error_code
);
00683     if (error_code || input->mutation_ratio < 0.
|| input->mutation_ratio >= 1.)
00684     {
00685         input_error (gettext ("Invalid mutation probability"));
00686         goto exit_on_error;
00687     }
00688 }
00689 }
00690 else
00691 {
00692     input_error (gettext ("No mutation probability"));
00693     goto exit_on_error;
00694 }
00695 }
00696 // Obtaining reproduction probability
00697 if (json_object_get_member (object, LABEL_REPRODUCTION))
00698 {
00699     input->reproduction_ratio
00700     = json_object_get_float (object,
LABEL_REPRODUCTION, &error_code);
00701     if (error_code || input->reproduction_ratio < 0.
|| input->reproduction_ratio >= 1.0)
00702     {
00703         input_error (gettext ("Invalid reproduction probability"));
00704         goto exit_on_error;
00705     }
00706 }
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,
00720 LABEL_ADAPTATION, &error_code);
00721         if (error_code || input->adaptation_ratio < 0.
00722             || input->adaptation_ratio >= 1.)
00723         {
00724             input_error (gettext ("Invalid adaptation probability"));
00725             goto exit_on_error;
00726         }
00727     }
00728     else
00729     {
00730         input_error (gettext ("No adaptation probability"));
00731         goto exit_on_error;
00732     }
00733
00734     // Checking survivals
00735     i = input->mutation_ratio * input->nsimulations;
00736     i += input->reproduction_ratio * input->nsimulations;
00737     i += input->adaptation_ratio * input->nsimulations;
00738     if (i > input->nsimulations - 2)
00739     {
00740         input_error
00741         (gettext
00742          ("No enough survival entities to reproduce the population"));
00743         goto exit_on_error;
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 );
00759         if (error_code == 1)
00760             input->niterations = 1;
00761         else if (error_code)
00762         {
00763             input_error (gettext ("Bad iterations number"));
00764             goto exit_on_error;
00765         }
00766
00767         // Obtaining best number
00768         input->nbest
00769         = json_object_get_uint_with_default (object,
00770 LABEL_NBEST, 1,
00771                                             &error_code);
00772         if (error_code || !input->nbest)
00773         {
00774             input_error (gettext ("Invalid best number"));
00775             goto exit_on_error;
00776         }
00777
00778         // Obtaining tolerance
00779         input->tolerance
00780         = json_object_get_float_with_default (object,
00781 LABEL_TOLERANCE, 0.,
00782                                             &error_code);
00783         if (error_code || input->tolerance < 0.)
00784         {
00785             input_error (gettext ("Invalid tolerance"));
00786             goto exit_on_error;
00787         }
00788
00789         // Getting direction search method parameters
00790         if (json_object_get_member (object, LABEL_NSTEPS))
00791         {
00792             input->nsteps
00793             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00794             if (error_code || !input->nsteps)
00795             {
00796                 input_error (gettext ("Invalid steps number"));
00797             }
00798         }
00799     }

```

```

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

```
enum DirectionMethod
```

Enum to define the methods to estimate the direction search.

Enumerator

DIRECTION_METHOD_COORDINATES	Coordinates descent method.
DIRECTION_METHOD_RANDOM	Random method.

Definition at line 45 of file [input.h](#).

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

5.9.2.2 ErrorNorm

enum [ErrorNorm](#)

Enum to define the error norm.

Enumerator

ERROR_NORM_EUCLIDIAN	Euclidian norm: $\sqrt{\sum_i (w_i x_i)^2}$.
ERROR_NORM_MAXIMUM	Maximum norm: $\max_i 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 input_error()**

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

Function to print an error message opening an [Input](#) struct.

Parameters

<i>message</i>	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 input_open()

```

int input_open (
    char * filename )

```

Function to open the input file.

Parameters

<i>filename</i>	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 input_open_json()

```

int input_open_json (
    JsonParser * parser )

```

Function to open the input file in JSON format.

Parameters

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

```

```

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

```



```

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 input_open_xml()

```

int input_open_xml (
    xmlDoc * doc )

```

Function to open the input file in XML format.

Parameters

<i>doc</i>	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 *)
00173 result_name);
00174     }
00175     if (!input->variables)
00176     {

```

```

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 *)
00199     LABEL_SEED,
00200                                         DEFAULT_RANDOM_SEED, &error_code);
00201     if (error_code)
00202     {
00203         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00204         goto exit_on_error;
00205     }
00206
00207     // Opening algorithm
00208     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00209     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00210     {
00211         input->algorithm = ALGORITHM_MONTE_CARLO;
00212
00213         // Obtaining simulations number
00214         input->nsimulations
00215             = xml_node_get_int (node, (const xmlChar *)
00216     LABEL_NSIMULATIONS,
00217                               &error_code);
00218         if (error_code)
00219         {
00220             input_error (gettext ("Bad simulations number"));
00221             goto exit_on_error;
00222         }
00223     }
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00225     {
00226         input->algorithm = ALGORITHM_SWEEP;
00227     }
00228     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00229     {
00230         input->algorithm = ALGORITHM_GENETIC;
00231     }
00232
00233     // Obtaining population
00234     if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00235     {
00236         input->nsimulations
00237             = xml_node_get_uint (node, (const xmlChar *)
00238     LABEL_NPOPULATION,
00239                               &error_code);
00240         if (error_code || input->nsimulations < 3)
00241         {
00242             input_error (gettext ("Invalid population number"));
00243             goto exit_on_error;
00244         }
00245     }
00246     else
00247     {
00248         input_error (gettext ("No population number"));
00249         goto exit_on_error;
00250     }
00251
00252     // Obtaining generations
00253     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00254     {
00255         input->niterations
00256             = xml_node_get_uint (node, (const xmlChar *)
00257     LABEL_NGENERATIONS,
00258                               &error_code);
00259         if (error_code || !input->niterations)
00260         {
00261             input_error (gettext ("Invalid generations number"));
00262             goto exit_on_error;
00263         }
00264     }
00265     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 *)
00269 LABEL_MUTATION,
00270                             &error_code);
00271         if (error_code || input->mutation_ratio < 0.
00272             || input->mutation_ratio >= 1.)
00273         {
00274             input_error (gettext ("Invalid mutation probability"));
00275             goto exit_on_error;
00276         }
00277     }
00278     else
00279     {
00280         input_error (gettext ("No mutation probability"));
00281         goto exit_on_error;
00282     }
00283
00284     // Obtaining reproduction probability
00285     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286     {
00287         input->reproduction_ratio
00288         = xml_node_get_float (node, (const xmlChar *)
00289 LABEL_REPRODUCTION,
00290                             &error_code);
00291         if (error_code || input->reproduction_ratio < 0.
00292             || input->reproduction_ratio >= 1.0)
00293         {
00294             input_error (gettext ("Invalid reproduction probability"));
00295             goto exit_on_error;
00296         }
00297     }
00298     else
00299     {
00300         input_error (gettext ("No reproduction probability"));
00301         goto exit_on_error;
00302     }
00303
00304     // Obtaining adaptation probability
00305     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00306     {
00307         input->adaptation_ratio
00308         = xml_node_get_float (node, (const xmlChar *)
00309 LABEL_ADAPTATION,
00310                             &error_code);
00311         if (error_code || input->adaptation_ratio < 0.
00312             || input->adaptation_ratio >= 1.)
00313         {
00314             input_error (gettext ("Invalid adaptation probability"));
00315             goto exit_on_error;
00316         }
00317     }
00318     else
00319     {
00320         input_error (gettext ("No adaptation probability"));
00321         goto exit_on_error;
00322     }
00323
00324     // Checking survivals
00325     i = input->mutation_ratio * input->nsimulations;
00326     i += input->reproduction_ratio * input->
00327 nsimulations;
00328     i += input->adaptation_ratio * input->
00329 nsimulations;
00330     if (i > input->nsimulations - 2)
00331     {
00332         input_error
00333         (gettext
00334          ("No enough survival entities to reproduce the population"));
00335         goto exit_on_error;
00336     }
00337     else
00338     {
00339         input_error (gettext ("Unknown algorithm"));
00340         goto exit_on_error;
00341     }
00342     xmlFree (buffer);
00343     buffer = NULL;
00344 }

```

```

00341     if (input->algorithm == ALGORITHM_MONTE_CARLO
00342         || input->algorithm == ALGORITHM_SWEEP)
00343     {
00344         // Obtaining iterations number
00345         input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
00347 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         // Obtaining best number
00357         input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
00359 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         // Obtaining tolerance
00367         input->tolerance
00368         = xml_node_get_float_with_default (node,
00369                                         (const xmlChar *) LABEL_TOLERANCE,
00370                                         0., &error_code);
00371         if (error_code || input->tolerance < 0.)
00372         {
00373             input_error (gettext ("Invalid tolerance"));
00374             goto exit_on_error;
00375         }
00376     }
00377     // Getting direction search method parameters
00378     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00379     {
00380         input->nsteps =
00381             xml_node_get_uint (node, (const xmlChar *)
00382 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 *)
00397 LABEL_NESTIMATES,
00398                             &error_code);
00399             if (error_code || !input->nestimates)
00400             {
00401                 input_error (gettext ("Invalid estimates number"));
00402                 goto exit_on_error;
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->
00418 relaxation > 2.)
00419             {
00420                 input_error (gettext ("Invalid relaxation parameter"));
00421                 goto exit_on_error;
00422             }
00423         }
00424     }

```

```

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 Burquete 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 <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 0`
Macro to debug interface functions.
- `#define INPUT_FILE "test-ga.xml"`
Macro to define the initial input file.

Functions

- void `input_save_direction_xml` (xmlNode *node)
Function to save the direction search method data in a XML node.
- void `input_save_direction_json` (JsonNode *node)
Function to save the direction search method data in a JSON node.
- void `input_save_xml` (xmlDoc *doc)
Function to save the input file in XML format.
- void `input_save_json` (JsonGenerator *generator)
Function to save the input file in JSON format.
- void `input_save` (char *filename)
Function to save the input file.
- void `options_new` ()
Function to open the options dialog.
- void `running_new` ()
Function to open the running dialog.
- unsigned int `window_get_algorithm` ()
Function to get the stochastic algorithm number.
- unsigned int `window_get_direction` ()
Function to get the direction search method number.
- unsigned int `window_get_norm` ()
Function to get the norm method number.
- void `window_save_direction` ()
Function to save the direction search method data in the input file.
- int `window_save` ()
Function to save the input file.
- void `window_run` ()
Function to run a optimization.
- void `window_help` ()
Function to show a help dialog.

- void [window_about](#) ()
Function to show an about dialog.
- void [window_update_direction](#) ()
Function to update direction search method widgets view in the main window.
- void [window_update](#) ()
Function to update the main window view.
- void [window_set_algorithm](#) ()
Function to avoid memory errors changing the algorithm.
- void [window_set_experiment](#) ()
Function to set the experiment data in the main window.
- void [window_remove_experiment](#) ()
Function to remove an experiment in the main window.
- void [window_add_experiment](#) ()
Function to add an experiment in the main window.
- void [window_name_experiment](#) ()
Function to set the experiment name in the main window.
- void [window_weight_experiment](#) ()
Function to update the experiment weight in the main window.
- void [window_inputs_experiment](#) ()
Function to update the experiment input templates number in the main window.
- void [window_template_experiment](#) (void *data)
Function to update the experiment i-th input template in the main window.
- void [window_set_variable](#) ()
Function to set the variable data in the main window.
- void [window_remove_variable](#) ()
Function to remove a variable in the main window.
- void [window_add_variable](#) ()
Function to add a variable in the main window.
- void [window_label_variable](#) ()
Function to set the variable label in the main window.
- void [window_precision_variable](#) ()
Function to update the variable precision in the main window.
- void [window_rangemin_variable](#) ()
Function to update the variable rangemin in the main window.
- void [window_rangemax_variable](#) ()
Function to update the variable rangemax in the main window.
- void [window_rangeminabs_variable](#) ()
Function to update the variable rangeminabs in the main window.
- void [window_rangemaxabs_variable](#) ()
Function to update the variable rangemaxabs in the main window.
- void [window_step_variable](#) ()
Function to update the variable step in the main window.
- void [window_update_variable](#) ()
Function to update the variable data in the main window.
- int [window_read](#) (char *filename)
Function to read the input data of a file.
- void [window_open](#) ()
Function to open the input data.
- void [window_new](#) ()
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

Copyright 2012-2016, all rights reserved.

Definition in file [interface.c](#).

5.11.2 Function Documentation

5.11.2.1 `input_save()`

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

Function to save the input file.

Parameters

<i>filename</i>	Input file name.
-----------------	------------------

Definition at line 575 of file [interface.c](#).

```
00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579
00580     #if DEBUG_INTERFACE
00581     fprintf (stderr, "input_save: start\n");
00582     #endif
00583
00584     // Getting the input file directory
00585     input->name = g_path_get_basename (filename);
00586     input->directory = g_path_get_dirname (filename);
00587 }
```

```

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

```

Here is the call graph for this function:

5.11.2.2 input_save_direction_json()

```

void input_save_direction_json (
    JsonNode * node )

```

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

Parameters

<i>node</i>	JSON node.
-------------	------------

Definition at line 207 of file [interface.c](#).

```

00208 {
00209     JsonObject *object;
00210     #if DEBUG_INTERFACE
00211     fprintf (stderr, "input_save_direction_json: start\n");
00212     #endif
00213     object = json_node_get_object (node);
00214     if (input->nsteps)
00215     {
00216         json_object_set_uint (object, LABEL_NSTEPS,
00217                               input->nsteps);
00218         if (input->relaxation != DEFAULT_RELAXATION)
00219             json_object_set_float (object, LABEL_RELAXATION,
00220                                    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,
00229                                                 LABEL_RANDOM);
00230         }
00231         json_object_set_uint (object, LABEL_NESTIMATES,
00232                               input->nestimates);
00233     }
00234 }

```

```

00230 #if DEBUG_INTERFACE
00231     fprintf (stderr, "input_save_direction_json: end\n");
00232 #endif
00233 }

```

Here is the call graph for this function:

5.11.2.3 input_save_direction_xml()

```

void input_save_direction_xml (
    xmlNode * node )

```

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

Parameters

<i>node</i>	XML node.
-------------	-----------

Definition at line 171 of file [interface.c](#).

```

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

```

Here is the call graph for this function:

5.11.2.4 input_save_json()

```

void input_save_json (
    JsonGenerator * generator )

```

Function to save the input file in JSON format.

Parameters

<i>generator</i>	JsonGenerator struct.
------------------	-----------------------

Definition at line 412 of file [interface.c](#).

```

00413 {
00414     unsigned int i, j;
00415     char *buffer;
00416     JsonNode *node, *child;
00417     JsonObject *object, *object2;
00418     JsonArray *array;
00419     GFile *file, *file2;
00420
00421     #if DEBUG_INTERFACE
00422         fprintf (stderr, "input_save_json: start\n");
00423     #endif
00424
00425     // Setting root JSON node
00426     node = json_node_new (JSON_NODE_OBJECT);
00427     object = json_node_get_object (node);
00428     json_generator_set_root (generator, node);
00429
00430     // Adding properties to the root JSON node
00431     if (strcmp (input->result, result_name))
00432         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00433     if (strcmp (input->variables, variables_name))
00434         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00435
00436     file = g_file_new_for_path (input->directory);
00437     file2 = g_file_new_for_path (input->simulator);
00438     buffer = g_file_get_relative_path (file, file2);
00439     g_object_unref (file2);
00440     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00441     g_free (buffer);
00442     if (input->evaluator)
00443     {
00444         file2 = g_file_new_for_path (input->evaluator);
00445         buffer = g_file_get_relative_path (file, file2);
00446         g_object_unref (file2);
00447         if (strlen (buffer))
00448             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00449         g_free (buffer);
00450     }
00451     if (input->seed != DEFAULT_RANDOM_SEED)
00452         json_object_set_uint (object, LABEL_SEED,
input->seed);
00453
00454     // Setting the algorithm
00455     buffer = (char *) g_slice_alloc (64);
00456     switch (input->algorithm)
00457     {
00458     case ALGORITHM_MONTE_CARLO:
00459         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00460         snprintf (buffer, 64, "%u", input->nsimulations);
00461         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00462         snprintf (buffer, 64, "%u", input->niterations);
00463         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00464         snprintf (buffer, 64, "%.3lg", input->tolerance);
00465         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00466         snprintf (buffer, 64, "%u", input->nbest);
00467         json_object_set_string_member (object, LABEL_NBEST, buffer);
00468         input_save_direction_json (node);
00469         break;
00470     case ALGORITHM_SWEEP:
00471         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00472         snprintf (buffer, 64, "%u", input->niterations);
00473         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00474         snprintf (buffer, 64, "%.3lg", input->tolerance);
00475         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00476         snprintf (buffer, 64, "%u", input->nbest);
00477         json_object_set_string_member (object, LABEL_NBEST, buffer);
00478         input_save_direction_json (node);
00479         break;
00480     default:
00481         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00482         snprintf (buffer, 64, "%u", input->nsimulations);
00483         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);

```

```

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

```



```

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

```

Here is the call graph for this function:

5.11.2.5 input_save_xml()

```

void input_save_xml (
    xmlDoc * doc )

```

Function to save the input file in XML format.

Parameters

<i>doc</i>	xmlDoc struct.
------------	----------------

Definition at line 242 of file [interface.c](#).

```

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

```

```

00294         (xmlChar *) buffer);
00295     snprintf (buffer, 64, "%u", input->niterations);
00296     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00297         (xmlChar *) buffer);
00298     snprintf (buffer, 64, "%.3lg", input->tolerance);
00299     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00300     snprintf (buffer, 64, "%u", input->nbest);
00301     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00302     input_save_direction_xml (node);
00303     break;
00304 case ALGORITHM_SWEEP:
00305     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306         (const xmlChar *) LABEL_SWEEP);
00307     snprintf (buffer, 64, "%u", input->niterations);
00308     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309         (xmlChar *) buffer);
00310     snprintf (buffer, 64, "%.3lg", input->tolerance);
00311     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312     snprintf (buffer, 64, "%u", input->nbest);
00313     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314     input_save_direction_xml (node);
00315     break;
00316 default:
00317     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318         (const xmlChar *) LABEL_GENETIC);
00319     snprintf (buffer, 64, "%u", input->nsimulations);
00320     xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321         (xmlChar *) buffer);
00322     snprintf (buffer, 64, "%u", input->niterations);
00323     xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324         (xmlChar *) buffer);
00325     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326     xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328     xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329         (xmlChar *) buffer);
00330     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332     break;
00333 }
00334 g_slice_free1 (64, buffer);
00335 if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
00337 LABEL_THRESHOLD,
00338         input->threshold);
00339 // Setting the experimental data
00340 for (i = 0; i < input->nexperiments; ++i)
00341 {
00342     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344         (xmlChar *) input->experiment[i].name);
00345     if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
00347 LABEL_WEIGHT,
00348             input->experiment[i].weight);
00349     for (j = 0; j < input->experiment->ninputs; ++j)
00350         xmlSetProp (child, (const xmlChar *) template[j],
00351             (xmlChar *) input->experiment[i].template[j]);
00352 }
00353 // Setting the variables data
00354 for (i = 0; i < input->nvariables; ++i)
00355 {
00356     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358         (xmlChar *) input->variable[i].name);
00359     xml_node_set_float (child, (const xmlChar *)
00360 LABEL_MINIMUM,
00361         input->variable[i].rangemin);
00362     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00363         xml_node_set_float (child, (const xmlChar *)
00364 LABEL_ABSOLUTE_MINIMUM,
00365             input->variable[i].rangeminabs);
00366     xml_node_set_float (child, (const xmlChar *)
00367 LABEL_MAXIMUM,
00368         input->variable[i].rangemax);
00369     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
00371 LABEL_ABSOLUTE_MAXIMUM,
00372             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
00374         DEFAULT_PRECISION)
00375         xml_node_set_uint (child, (const xmlChar *)
00376 LABEL_PRECISION,
00377             input->variable[i].precision);
00378     if (input->algorithm == ALGORITHM_SWEEP)

```

```

00373     xml_node_set_uint (child, (const xmlChar *)
00374     LABEL_NSWEEPS,
00375     input->variable[i].nsweeps);
00376     else if (input->algorithm == ALGORITHM_GENETIC)
00377         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00378         input->variable[i].nbits);
00379     if (input->nsteps)
00380         xml_node_set_float (child, (const xmlChar *)
00381         LABEL_STEP,
00382         input->variable[i].step);
00383     // Saving the error norm
00384     switch (input->norm)
00385     {
00386     case ERROR_NORM_MAXIMUM:
00387         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00388         (const xmlChar *) LABEL_MAXIMUM);
00389         break;
00390     case ERROR_NORM_P:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392         (const xmlChar *) LABEL_P);
00393         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00394         input->p);
00395         break;
00396     case ERROR_NORM_TAXICAB:
00397         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00398         (const xmlChar *) LABEL_TAXICAB);
00399     }
00400 #if DEBUG_INTERFACE
00401     fprintf (stderr, "input_save: end\n");
00402 #endif
00403 }

```

Here is the call graph for this function:

5.11.2.6 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 726 of file [interface.c](#).

```

00727 {
00728     unsigned int i;
00729     #if DEBUG_INTERFACE
00730     fprintf (stderr, "window_get_algorithm: start\n");
00731     #endif
00732     i = gtk_array_get_active (window->button_algorithm,
00733     NALGORITHMS);
00734     #if DEBUG_INTERFACE
00735     fprintf (stderr, "window_get_algorithm: %u\n", i);
00736     fprintf (stderr, "window_get_algorithm: end\n");
00737     #endif
00738     return i;
00739 }

```

Here is the call graph for this function:

5.11.2.7 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 746 of file [interface.c](#).

```
00747 {
00748     unsigned int i;
00749     #if DEBUG_INTERFACE
00750     fprintf (stderr, "window_get_direction: start\n");
00751     #endif
00752     i = gtk_array_get_active (window->button_direction,
00753                             NDIRECTIONS);
00754     #if DEBUG_INTERFACE
00755     fprintf (stderr, "window_get_direction: %u\n", i);
00756     fprintf (stderr, "window_get_direction: end\n");
00757     #endif
00758     return i;
00759 }
```

Here is the call graph for this function:

5.11.2.8 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

Definition at line 766 of file [interface.c](#).

```
00767 {
00768     unsigned int i;
00769     #if DEBUG_INTERFACE
00770     fprintf (stderr, "window_get_norm: start\n");
00771     #endif
00772     i = gtk_array_get_active (window->button_norm,
00773                             NNORMS);
00774     #if DEBUG_INTERFACE
00775     fprintf (stderr, "window_get_norm: %u\n", i);
00776     fprintf (stderr, "window_get_norm: end\n");
00777     #endif
00778     return i;
00779 }
```

Here is the call graph for this function:

5.11.2.9 window_read()

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

Function to read the input data of a file.

Parameters

<i>filename</i>	File name.
-----------------	------------

Returns

1 on succes, 0 on error.

Definition at line 1874 of file [interface.c](#).

```

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

```

```

01937     }
01938     break;
01939     default:
01940         gtk_spin_button_set_value (window->spin_population,
01941                                   (gdouble) input->nsimulations);
01942         gtk_spin_button_set_value (window->spin_generations,
01943                                   (gdouble) input->niterations);
01944         gtk_spin_button_set_value (window->spin_mutation, input->
01945                                   mutation_ratio);
01946         gtk_spin_button_set_value (window->spin_reproduction,
01947                                   input->reproduction_ratio);
01948         gtk_spin_button_set_value (window->spin_adaptation,
01949                                   input->adaptation_ratio);
01950     }
01951     gtk_toggle_button_set_active
01952     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01953     gtk_spin_button_set_value (window->spin_p, input->p);
01954     gtk_spin_button_set_value (window->spin_threshold, input->
01955                               threshold);
01956     g_signal_handler_block (window->combo_experiment, window->
01957                             id_experiment);
01958     g_signal_handler_block (window->button_experiment,
01959                             window->id_experiment_name);
01960     gtk_combo_box_text_remove_all (window->combo_experiment);
01961     for (i = 0; i < input->nexperiments; ++i)
01962         gtk_combo_box_text_append_text (window->combo_experiment,
01963                                         input->experiment[i].name);
01964     g_signal_handler_unblock
01965     (window->button_experiment, window->
01966       id_experiment_name);
01967     g_signal_handler_unblock (window->combo_experiment,
01968                               window->id_experiment);
01969     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01970     g_signal_handler_block (window->combo_variable, window->
01971                             id_variable);
01972     g_signal_handler_block (window->entry_variable, window->
01973                             id_variable_label);
01974     gtk_combo_box_text_remove_all (window->combo_variable);
01975     for (i = 0; i < input->nvariables; ++i)
01976         gtk_combo_box_text_append_text (window->combo_variable,
01977                                         input->variable[i].name);
01978     g_signal_handler_unblock (window->entry_variable, window->
01979                             id_variable_label);
01980     g_signal_handler_unblock (window->combo_variable, window->
01981                             id_variable);
01982     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01983     window_set_variable ();
01984     window_update ();
01985 }
01986
01987 #if DEBUG_INTERFACE
01988 fprintf (stderr, "window_read: end\n");
01989 #endif
01990 return 1;
01991 }

```

Here is the call graph for this function:

5.11.2.10 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 819 of file [interface.c](#).

```

00820 {
00821     GtkFileChooserDialog *dlg;
00822     GtkFileFilter *filter1, *filter2;
00823     char *buffer;
00824
00825     #if DEBUG_INTERFACE
00826         fprintf (stderr, "window_save: start\n");
00827     #endif
00828
00829     // Opening the saving dialog
00830     dlg = (GtkFileChooserDialog *)
00831         gtk_file_chooser_dialog_new (gettext ("Save file"),
00832                                     window->window,
00833                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00834                                     gettext ("Cancel"),
00835                                     GTK_RESPONSE_CANCEL,
00836                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
00837     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00838     buffer = g_build_filename (input->directory, input->name, NULL);
00839     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00840     g_free (buffer);
00841
00842     // Adding XML filter
00843     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00844     gtk_file_filter_set_name (filter1, "XML");
00845     gtk_file_filter_add_pattern (filter1, "*.xml");
00846     gtk_file_filter_add_pattern (filter1, "*.XML");
00847     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00848
00849     // Adding JSON filter
00850     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter2, "JSON");
00852     gtk_file_filter_add_pattern (filter2, "*.json");
00853     gtk_file_filter_add_pattern (filter2, "*.JSON");
00854     gtk_file_filter_add_pattern (filter2, "*.js");
00855     gtk_file_filter_add_pattern (filter2, "*.JS");
00856     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00857
00858     if (input->type == INPUT_TYPE_XML)
00859         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00860     else
00861         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863     // If OK response then saving
00864     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00865     {
00866         // Setting input file type
00867         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00868         buffer = (char *) gtk_file_filter_get_name (filter1);
00869         if (!strcmp (buffer, "XML"))
00870             input->type = INPUT_TYPE_XML;
00871         else
00872             input->type = INPUT_TYPE_JSON;
00873
00874         // Adding properties to the root XML node
00875         input->simulator = gtk_file_chooser_get_filename
00876             (GTK_FILE_CHOOSER (window->button_simulator));
00877         if (gtk_toggle_button_get_active
00878             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00879             input->evaluator = gtk_file_chooser_get_filename
00880                 (GTK_FILE_CHOOSER (window->button_evaluator));
00881         else
00882             input->evaluator = NULL;
00883         if (input->type == INPUT_TYPE_XML)
00884         {
00885             input->result
00886                 = (char *) xmlStrdup ((const xmlChar *)
00887                                         gtk_entry_get_text (window->entry_result));
00888             input->variables
00889                 = (char *) xmlStrdup ((const xmlChar *)
00890                                         gtk_entry_get_text (window->
00891 entry_variables));
00892         }
00893         else
00894         {
00895             input->result = g_strdup (gtk_entry_get_text (window->
00896 entry_result));
00897             input->variables
00898                 = g_strdup (gtk_entry_get_text (window->entry_variables));
00899         }
00900
00901         // Setting the algorithm
00902         switch (window_get_algorithm ())
00903         {
00904             case ALGORITHM_MONTE_CARLO:
00905                 input->algorithm = ALGORITHM_MONTE_CARLO;
00906                 input->nsimulations

```

```

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

```

Here is the call graph for this function:

5.11.2.11 window_template_experiment()

```

void window_template_experiment (
    void * data )

```

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

Parameters

<i>data</i>	Callback data (i-th input template).
-------------	--------------------------------------

Definition at line 1518 of file [interface.c](#).


```

01519 {
01520     unsigned int i, j;
01521     char *buffer;
01522     GFile *file1, *file2;
01523     #if DEBUG_INTERFACE
01524     fprintf (stderr, "window_template_experiment: start\n");
01525     #endif
01526     i = (size_t) data;
01527     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01528     file1
01529     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01530     file2 = g_file_new_for_path (input->directory);
01531     buffer = g_file_get_relative_path (file2, file1);
01532     if (input->type == INPUT_TYPE_XML)
01533         input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01534     else
01535         input->experiment[j].template[i] = g_strdup (buffer);
01536     g_free (buffer);
01537     g_object_unref (file2);
01538     g_object_unref (file1);
01539     #if DEBUG_INTERFACE
01540     fprintf (stderr, "window_template_experiment: end\n");
01541     #endif
01542 }

```

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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #include <gio/gio.h>
00051 #include <gtk/gtk.h>
00052 #include "genetic/genetic.h"
00053 #include "utils.h"
00054 #include "experiment.h"
00055 #include "variable.h"

```

```

00062 #include "input.h"
00063 #include "optimize.h"
00064 #include "interface.h"
00065
00066 #define DEBUG_INTERFACE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define INPUT_FILE "test-ga-win.xml"
00074 #else
00075 #define INPUT_FILE "test-ga.xml"
00076 #endif
00077
00078 const char *logo[] = {
00079     "32 32 3 1",
00080     "    c None",
00081     ".    c #0000FF",
00082 "+    c #FF0000",
00083     "                                ",
00084     "                                ",
00085     "                                ",
00086     "    .    .    .    .    ",
00087     "    .    .    .    .    ",
00088     "    .    .    .    .    ",
00089     "    .    .    .    .    ",
00090     "    .    .    +++    .    ",
00091     "    .    .    +++++    .    ",
00092     "    .    .    +++++    .    ",
00093     "    .    .    +++++    .    ",
00094     "    +++    .    +++    +++    ",
00095     "    +++++    .    +++++    ",
00096     "    +++++    .    +++++    ",
00097     "    +++++    .    +++++    ",
00098     "    +++    .    +++    ",
00099     "    .    .    .    .    ",
00100     "    .    +++    .    .    ",
00101     "    .    +++++    .    .    ",
00102     "    .    +++++    .    .    ",
00103     "    .    +++++    .    .    ",
00104     "    .    +++    .    .    ",
00105     "    .    .    .    .    ",
00106     "    .    .    .    .    ",
00107     "    .    .    .    .    ",
00108     "    .    .    .    .    ",
00109     "    .    .    .    .    ",
00110     "    .    .    .    .    ",
00111     "    .    .    .    .    ",
00112     "                                ",
00113     "                                ",
00114     "                                ",
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119     "32 32 3 1",
00120     "    c #FFFFFFFFFFFF",
00121     ".    c #00000000FFFF",
00122     "X    c #FFFF00000000",
00123     "                                ",
00124     "                                ",
00125     "                                ",
00126     "    .    .    .    .    ",
00127     "    .    .    .    .    ",
00128     "    .    .    .    .    ",
00129     "    .    .    .    .    ",
00130     "    .    .    XXX    .    ",
00131     "    .    .    XXXXX    .    ",
00132     "    .    .    XXXXX    .    ",
00133     "    .    .    XXXXX    .    ",
00134     "    XXX    .    XXX    XXX    ",
00135     "    XXXXX    .    XXXXX    ",
00136     "    XXXXX    .    XXXXX    ",
00137     "    XXXXX    .    XXXXX    ",
00138     "    XXX    .    XXX    ",
00139     "    .    .    .    .    ",
00140     "    .    XXX    .    .    ",
00141     "    .    XXXXX    .    .    ",
00142     "    .    XXXXX    .    .    ",
00143     "    .    XXXXX    .    .    ",
00144     "    .    XXX    .    .    ",
00145     "    .    .    .    .    ",
00146     "    .    .    .    .    ",
00147     "    .    .    .    .    ",
00148     "    .    .    .    .    ",
00149     "    .    .    .    .    ",
00150     "    .    .    .    .    ",
00151     "    .    .    .    .    ",

```

```

00152 "                                ",
00153 "                                ",
00154 "                                ";
00155 */
00156
00157 Options options[1];
00159 Running running[1];
00161 Window window[1];
00163
00170 void
00171 input_save_direction_xml (xmlNode * node)
00172 {
00173     #if DEBUG_INTERFACE
00174         fprintf (stderr, "input_save_direction_xml: start\n");
00175     #endif
00176     if (input->nsteps)
00177     {
00178         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00179 input->nsteps);
00179         if (input->relaxation != DEFAULT_RELAXATION)
00180             xml_node_set_float (node, (const xmlChar *)
00181 LABEL_RELAXATION,
00182                               input->relaxation);
00182         switch (input->direction)
00183         {
00184             case DIRECTION_METHOD_COORDINATES:
00185                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00186 (const xmlChar *) LABEL_COORDINATES);
00187                 break;
00188             default:
00189                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00190 (const xmlChar *) LABEL_RANDOM);
00191                 xml_node_set_uint (node, (const xmlChar *)
00192 LABEL_NESTIMATES,
00193                               input->nestimates);
00194         }
00195     }
00196     #if DEBUG_INTERFACE
00197         fprintf (stderr, "input_save_direction_xml: end\n");
00198     #endif
00199 }
00206 void
00207 input_save_direction_json (JsonNode * node)
00208 {
00209     JsonObject *object;
00210     #if DEBUG_INTERFACE
00211         fprintf (stderr, "input_save_direction_json: start\n");
00212     #endif
00213     object = json_node_get_object (node);
00214     if (input->nsteps)
00215     {
00216         json_object_set_uint (object, LABEL_NSTEPS,
00217 input->nsteps);
00217         if (input->relaxation != DEFAULT_RELAXATION)
00218             json_object_set_float (object, LABEL_RELAXATION,
00219 input->relaxation);
00219         switch (input->direction)
00220         {
00221             case DIRECTION_METHOD_COORDINATES:
00222                 json_object_set_string_member (object, LABEL_DIRECTION,
00223 LABEL_COORDINATES);
00224                 break;
00225             default:
00226                 json_object_set_string_member (object, LABEL_DIRECTION,
00227 LABEL_RANDOM);
00227                 json_object_set_uint (object, LABEL_NESTIMATES,
00228 input->nestimates);
00229         }
00230     }
00231     #if DEBUG_INTERFACE
00232         fprintf (stderr, "input_save_direction_json: end\n");
00233     #endif
00234 }
00241 void
00242 input_save_xml (xmlDoc * doc)
00243 {
00244     unsigned int i, j;
00245     char *buffer;
00246     xmlNode *node, *child;
00247     GFile *file, *file2;
00248
00249     #if DEBUG_INTERFACE
00250         fprintf (stderr, "input_save_xml: start\n");
00251     #endif
00252

```

```

00253 // Setting root XML node
00254 node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255 xmlDocSetRootElement (doc, node);
00256
00257 // Adding properties to the root XML node
00258 if (xmlStrcmp
00259     ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260     xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                 (xmlChar *) input->result);
00262 if (xmlStrcmp
00263     ((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
00264     xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00265                 (xmlChar *) input->variables);
00266 file = g_file_new_for_path (input->directory);
00267 file2 = g_file_new_for_path (input->simulator);
00268 buffer = g_file_get_relative_path (file, file2);
00269 g_object_unref (file2);
00270 xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00271 g_free (buffer);
00272 if (input->evaluator)
00273 {
00274     file2 = g_file_new_for_path (input->evaluator);
00275     buffer = g_file_get_relative_path (file, file2);
00276     g_object_unref (file2);
00277     if (xmlStrlen ((xmlChar *) buffer))
00278         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00279                     (xmlChar *) buffer);
00280     g_free (buffer);
00281 }
00282 if (input->seed != DEFAULT_RANDOM_SEED)
00283     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00284
00285 // Setting the algorithm
00286 buffer = (char *) g_slice_alloc (64);
00287 switch (input->algorithm)
00288 {
00289     case ALGORITHM_MONTE_CARLO:
00290         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00291                     (const xmlChar *) LABEL_MONTE_CARLO);
00292         snprintf (buffer, 64, "%u", input->nsimulations);
00293         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00294                     (xmlChar *) buffer);
00295         snprintf (buffer, 64, "%u", input->niterations);
00296         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00297                     (xmlChar *) buffer);
00298         snprintf (buffer, 64, "%.3lg", input->tolerance);
00299         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%u", input->nbest);
00301         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00302         input_save_direction_xml (node);
00303         break;
00304     case ALGORITHM_SWEEP:
00305         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00306                     (const xmlChar *) LABEL_SWEEP);
00307         snprintf (buffer, 64, "%u", input->niterations);
00308         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00309                     (xmlChar *) buffer);
00310         snprintf (buffer, 64, "%.3lg", input->tolerance);
00311         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%u", input->nbest);
00313         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00314         input_save_direction_xml (node);
00315         break;
00316     default:
00317         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00318                     (const xmlChar *) LABEL_GENETIC);
00319         snprintf (buffer, 64, "%u", input->nsimulations);
00320         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00321                     (xmlChar *) buffer);
00322         snprintf (buffer, 64, "%u", input->niterations);
00323         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00324                     (xmlChar *) buffer);
00325         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00326         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00328         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00329                     (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00332         break;
00333 }
00334 g_slice_free1 (64, buffer);
00335 if (input->threshold != 0.)
00336     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,

```

```

00337         input->threshold);
00338
00339 // Setting the experimental data
00340 for (i = 0; i < input->nexperiments; ++i)
00341 {
00342     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00343     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00344                 (xmlChar *) input->experiment[i].name);
00345     if (input->experiment[i].weight != 1.)
00346         xml_node_set_float (child, (const xmlChar *)
00347                             LABEL_WEIGHT,
00348                             input->experiment[i].weight);
00349     for (j = 0; j < input->experiment->ninputs; ++j)
00350         xmlSetProp (child, (const xmlChar *) template[j],
00351                     (xmlChar *) input->experiment[i].template[j]);
00352 }
00353 // Setting the variables data
00354 for (i = 0; i < input->nvariables; ++i)
00355 {
00356     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00357     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00358                 (xmlChar *) input->variable[i].name);
00359     xml_node_set_float (child, (const xmlChar *)
00360                         LABEL_MINIMUM,
00361                         input->variable[i].rangemin);
00362     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00363         xml_node_set_float (child, (const xmlChar *)
00364                             LABEL_ABSOLUTE_MINIMUM,
00365                             input->variable[i].rangeminabs);
00366     xml_node_set_float (child, (const xmlChar *)
00367                         LABEL_MAXIMUM,
00368                         input->variable[i].rangemax);
00369     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00370         xml_node_set_float (child, (const xmlChar *)
00371                             LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
00374         DEFAULT_PRECISION)
00375         xml_node_set_uint (child, (const xmlChar *)
00376                             LABEL_PRECISION,
00377                             input->variable[i].precision);
00378     if (input->algorithm == ALGORITHM_SWEEP)
00379         xml_node_set_uint (child, (const xmlChar *)
00380                             LABEL_NSWEEPS,
00381                             input->variable[i].nsweeps);
00382     else if (input->algorithm == ALGORITHM_GENETIC)
00383         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00384                             input->variable[i].nbits);
00385     if (input->nsteps)
00386         xml_node_set_float (child, (const xmlChar *)
00387                             LABEL_STEP,
00388                             input->variable[i].step);
00389 }
00390
00391 // Saving the error norm
00392 switch (input->norm)
00393 {
00394     case ERROR_NORM_MAXIMUM:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                     (const xmlChar *) LABEL_MAXIMUM);
00397         break;
00398     case ERROR_NORM_P:
00399         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00400                     (const xmlChar *) LABEL_P);
00401         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00402                             input->p);
00403         break;
00404     case ERROR_NORM_TAXICAB:
00405         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00406                     (const xmlChar *) LABEL_TAXICAB);
00407         break;
00408 }
00409
00410 #if DEBUG_INTERFACE
00411 fprintf (stderr, "input_save: end\n");
00412 #endif
00413 }
00414
00415 void
00416 input_save_json (JsonGenerator * generator)
00417 {
00418     unsigned int i, j;
00419     char *buffer;
00420     XmlNode *node, *child;
00421     JsonObject *object, *object2;
00422     JsonArray *array;
00423     GFile *file, *file2;

```

```

00420
00421 #if DEBUG_INTERFACE
00422     fprintf (stderr, "input_save_json: start\n");
00423 #endif
00424
00425     // Setting root JSON node
00426     node = json_node_new (JSON_NODE_OBJECT);
00427     object = json_node_get_object (node);
00428     json_generator_set_root (generator, node);
00429
00430     // Adding properties to the root JSON node
00431     if (strcmp (input->result, result_name))
00432         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00433     if (strcmp (input->variables, variables_name))
00434         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00435     file = g_file_new_for_path (input->directory);
00436     file2 = g_file_new_for_path (input->simulator);
00437     buffer = g_file_get_relative_path (file, file2);
00438     g_object_unref (file2);
00439     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00440     g_free (buffer);
00441     if (input->evaluator)
00442     {
00443         file2 = g_file_new_for_path (input->evaluator);
00444         buffer = g_file_get_relative_path (file, file2);
00445         g_object_unref (file2);
00446         if (strlen (buffer))
00447             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00448         g_free (buffer);
00449     }
00450     if (input->seed != DEFAULT_RANDOM_SEED)
00451         json_object_set_uint (object, LABEL_SEED,
input->seed);
00452
00453     // Setting the algorithm
00454     buffer = (char *) g_slice_alloc (64);
00455     switch (input->algorithm)
00456     {
00457     case ALGORITHM_MONTE_CARLO:
00458         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00459         snprintf (buffer, 64, "%u", input->nsimulations);
00460         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00461         snprintf (buffer, 64, "%u", input->niterations);
00462         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00463         snprintf (buffer, 64, "%.3lg", input->tolerance);
00464         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00465         snprintf (buffer, 64, "%u", input->nbest);
00466         json_object_set_string_member (object, LABEL_NBEST, buffer);
00467         input_save_direction_json (node);
00468         break;
00469     case ALGORITHM_SWEEP:
00470         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00471         snprintf (buffer, 64, "%u", input->niterations);
00472         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00473         snprintf (buffer, 64, "%.3lg", input->tolerance);
00474         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00475         snprintf (buffer, 64, "%u", input->nbest);
00476         json_object_set_string_member (object, LABEL_NBEST, buffer);
00477         input_save_direction_json (node);
00478         break;
00479     default:
00480         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00481         snprintf (buffer, 64, "%u", input->nsimulations);
00482         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00483         snprintf (buffer, 64, "%u", input->niterations);
00484         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00485         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00486         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00487         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00488         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00489         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00490         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00491         break;
00492     }
00493     g_slice_free1 (64, buffer);
00494     if (input->threshold != 0.)
00495         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00496
00497     // Setting the experimental data
00498     array = json_array_new ();
00499     for (i = 0; i < input->nexperiments; ++i)

```

```

00502     {
00503         child = json_node_new (JSON_NODE_OBJECT);
00504         object = json_node_get_object (child);
00505         json_object_set_string_member (object2, LABEL_NAME,
00506                                     input->experiment[i].name);
00507         if (input->experiment[i].weight != 1.)
00508             json_object_set_float (object2, LABEL_WEIGHT,
00509                                   input->experiment[i].weight);
00510         for (j = 0; j < input->experiment->ninputs; ++j)
00511             json_object_set_string_member (object2, template[j],
00512                                           input->experiment[i].
00513 template[j]);
00514         json_array_add_element (array, child);
00515     }
00516     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00517     // Setting the variables data
00518     array = json_array_new ();
00519     for (i = 0; i < input->nvariables; ++i)
00520     {
00521         child = json_node_new (JSON_NODE_OBJECT);
00522         object = json_node_get_object (child);
00523         json_object_set_string_member (object2, LABEL_NAME,
00524                                     input->variable[i].name);
00525         json_object_set_float (object2, LABEL_MINIMUM,
00526                               input->variable[i].rangemin);
00527         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00528             json_object_set_float (object2,
00529 LABEL_ABSOLUTE_MINIMUM,
00530                                   input->variable[i].rangeminabs);
00531         json_object_set_float (object2, LABEL_MAXIMUM,
00532                               input->variable[i].rangemax);
00533         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00534             json_object_set_float (object2,
00535 LABEL_ABSOLUTE_MAXIMUM,
00536                                   input->variable[i].rangemaxabs);
00537         if (input->variable[i].precision !=
00538 DEFAULT_PRECISION)
00539             json_object_set_uint (object2, LABEL_PRECISION,
00540                                  input->variable[i].precision);
00541         if (input->algorithm == ALGORITHM_SWEEP)
00542             json_object_set_uint (object2, LABEL_NSWEEPS,
00543                                  input->variable[i].nsweeps);
00544         else if (input->algorithm == ALGORITHM_GENETIC)
00545             json_object_set_uint (object2, LABEL_NBITS,
00546                                  input->variable[i].nbits);
00547         if (input->nsteps)
00548             json_object_set_float (object, LABEL_STEP,
00549                                   input->variable[i].step);
00550         json_array_add_element (array, child);
00551     }
00552     json_object_set_array_member (object, LABEL_VARIABLES, array);
00553     // Saving the error norm
00554     switch (input->norm)
00555     {
00556     case ERROR_NORM_MAXIMUM:
00557         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00558         break;
00559     case ERROR_NORM_P:
00560         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00561         json_object_set_float (object, LABEL_P, input->
00562 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 void
00573 input_save (char *filename)
00574 {
00575     xmlDoc *doc;
00576     JsonGenerator *generator;
00577     #if DEBUG_INTERFACE
00578     fprintf (stderr, "input_save: start\n");
00579     #endif
00580     // Getting the input file directory
00581     input->name = g_path_get_basename (filename);
00582     input->directory = g_path_get_dirname (filename);
00583 }

```

```

00588     if (input->type == INPUT_TYPE_XML)
00589     {
00590         // Opening the input file
00591         doc = xmlNewDoc ((const xmlChar *) "1.0");
00592         input_save_xml (doc);
00593
00594         // Saving the XML file
00595         xmlSaveFormatFile (filename, doc, 1);
00596
00597         // Freeing memory
00598         xmlFreeDoc (doc);
00599     }
00600     else
00601     {
00602         // Opening the input file
00603         generator = json_generator_new ();
00604         json_generator_set_pretty (generator, TRUE);
00605         input_save_json (generator);
00606
00607         // Saving the JSON file
00608         json_generator_to_file (generator, filename, NULL);
00609
00610         // Freeing memory
00611         g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615     fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
00618
00623 void
00624 options_new ()
00625 {
00626 #if DEBUG_INTERFACE
00627     fprintf (stderr, "options_new: start\n");
00628 #endif
00629     options->label_seed = (GtkLabel *)
00630         gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00631     options->spin_seed = (GtkSpinButton *)
00632         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00633     gtk_widget_set_tooltip_text
00634         (GTK_WIDGET (options->spin_seed),
00635          gettext ("Seed to init the pseudo-random numbers generator"));
00636     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00637     options->label_threads = (GtkLabel *)
00638         gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00639     options->spin_threads
00640         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00641     gtk_widget_set_tooltip_text
00642         (GTK_WIDGET (options->spin_threads),
00643          gettext ("Number of threads to perform the calibration/optimization for "
00644                  "the stochastic algorithm"));
00645     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00646     options->label_direction = (GtkLabel *)
00647         gtk_label_new (gettext ("Threads number for the direction search method"));
00648     options->spin_direction
00649         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650     gtk_widget_set_tooltip_text
00651         (GTK_WIDGET (options->spin_direction),
00652          gettext ("Number of threads to perform the calibration/optimization for "
00653                  "the direction search method"));
00654     gtk_spin_button_set_value (options->spin_direction,
00655                                (gdouble) nthreads_direction);
00656     options->grid = (GtkGrid *) gtk_grid_new ();
00657     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00658     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00659     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00660                      0, 1, 1, 1);
00661     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00662                      1, 1, 1, 1);
00663     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00664                      0, 2, 1, 1);
00665     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00666                      1, 2, 1, 1);
00667     gtk_widget_show_all (GTK_WIDGET (options->grid));
00668     options->dialog = (GtkDialog *)
00669         gtk_dialog_new_with_buttons (gettext ("Options"),
00670                                     window->window,
00671                                     GTK_DIALOG_MODAL,
00672                                     gettext ("_OK"), GTK_RESPONSE_OK,
00673                                     gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
00674                                     NULL);
00675     gtk_container_add
00676         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),

```



```

00677     GTK_WIDGET (options->grid));
00678     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00679     {
00680         input->seed
00681         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00682         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00683         nthreads_direction
00684         = gtk_spin_button_get_value_as_int (options->spin_direction);
00685     }
00686     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00687     #if DEBUG_INTERFACE
00688     fprintf (stderr, "options_new: end\n");
00689     #endif
00690 }
00691
00692 void
00693 running_new ()
00694 {
00695     #if DEBUG_INTERFACE
00696     fprintf (stderr, "running_new: start\n");
00697     #endif
00698     running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00699     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00700     running->grid = (GtkGrid *) gtk_grid_new ();
00701     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00702     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00703     running->dialog = (GtkDialog *)
00704     gtk_dialog_new_with_buttons (gettext ("Calculating"),
00705     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00706     gtk_container_add
00707     (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00708     GTK_WIDGET (running->grid));
00709     gtk_spinner_start (running->spinner);
00710     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00711     #if DEBUG_INTERFACE
00712     fprintf (stderr, "running_new: end\n");
00713     #endif
00714 }
00715
00716 unsigned int
00717 window_get_algorithm ()
00718 {
00719     unsigned int i;
00720     #if DEBUG_INTERFACE
00721     fprintf (stderr, "window_get_algorithm: start\n");
00722     #endif
00723     i = gtk_array_get_active (window->button_algorithm,
00724     NALGORITHMS);
00725     #if DEBUG_INTERFACE
00726     fprintf (stderr, "window_get_algorithm: %u\n", i);
00727     fprintf (stderr, "window_get_algorithm: end\n");
00728     #endif
00729     return i;
00730 }
00731
00732 unsigned int
00733 window_get_direction ()
00734 {
00735     unsigned int i;
00736     #if DEBUG_INTERFACE
00737     fprintf (stderr, "window_get_direction: start\n");
00738     #endif
00739     i = gtk_array_get_active (window->button_direction,
00740     NDIRECTIONS);
00741     #if DEBUG_INTERFACE
00742     fprintf (stderr, "window_get_direction: %u\n", i);
00743     fprintf (stderr, "window_get_direction: end\n");
00744     #endif
00745     return i;
00746 }
00747
00748 unsigned int
00749 window_get_norm ()
00750 {
00751     unsigned int i;
00752     #if DEBUG_INTERFACE
00753     fprintf (stderr, "window_get_norm: start\n");
00754     #endif
00755     i = gtk_array_get_active (window->button_norm,
00756     NNORMS);
00757     #if DEBUG_INTERFACE
00758     fprintf (stderr, "window_get_norm: %u\n", i);
00759     fprintf (stderr, "window_get_norm: end\n");
00760     #endif
00761     return i;
00762 }
00763
00764 }
00765
00766

```

```

00784 void
00785 window_save_direction ()
00786 {
00787     #if DEBUG_INTERFACE
00788     fprintf (stderr, "window_save_direction: start\n");
00789     #endif
00790     if (gtk_toggle_button_get_active
00791         (GTK_TOGGLE_BUTTON (window->check_direction)))
00792     {
00793         input->nsteps = gtk_spin_button_get_value_as_int (window->
spin_steps);
00794         input->relaxation = gtk_spin_button_get_value (window->
spin_relaxation);
00795         switch (window_get_direction ())
00796         {
00797             case DIRECTION_METHOD_COORDINATES:
00798                 input->direction = DIRECTION_METHOD_COORDINATES;
00799                 break;
00800             default:
00801                 input->direction = DIRECTION_METHOD_RANDOM;
00802                 input->nestimates
00803                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00804         }
00805     }
00806     else
00807         input->nsteps = 0;
00808     #if DEBUG_INTERFACE
00809     fprintf (stderr, "window_save_direction: end\n");
00810     #endif
00811 }
00812
00818 int
00819 window_save ()
00820 {
00821     GtkFileChooserDialog *dlg;
00822     GtkFileFilter *filter1, *filter2;
00823     char *buffer;
00824
00825     #if DEBUG_INTERFACE
00826     fprintf (stderr, "window_save: start\n");
00827     #endif
00828
00829     // Opening the saving dialog
00830     dlg = (GtkFileChooserDialog *)
00831         gtk_file_chooser_dialog_new (gettext ("Save file"),
00832                                     window->window,
00833                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00834                                     gettext ("Cancel"),
00835                                     GTK_RESPONSE_CANCEL,
00836                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
00837     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00838     buffer = g_build_filename (input->directory, input->name, NULL);
00839     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00840     g_free (buffer);
00841
00842     // Adding XML filter
00843     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00844     gtk_file_filter_set_name (filter1, "XML");
00845     gtk_file_filter_add_pattern (filter1, "*.xml");
00846     gtk_file_filter_add_pattern (filter1, "*.XML");
00847     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00848
00849     // Adding JSON filter
00850     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter2, "JSON");
00852     gtk_file_filter_add_pattern (filter2, "*.json");
00853     gtk_file_filter_add_pattern (filter2, "*.JSON");
00854     gtk_file_filter_add_pattern (filter2, "*.js");
00855     gtk_file_filter_add_pattern (filter2, "*.JS");
00856     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00857
00858     if (input->type == INPUT_TYPE_XML)
00859         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00860     else
00861         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863     // If OK response then saving
00864     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00865     {
00866         // Setting input file type
00867         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00868         buffer = (char *) gtk_file_filter_get_name (filter1);
00869         if (!strcmp (buffer, "XML"))
00870             input->type = INPUT_TYPE_XML;
00871         else
00872             input->type = INPUT_TYPE_JSON;
00873     }

```

```

00874     // Adding properties to the root XML node
00875     input->simulator = gtk_file_chooser_get_filename
00876     (GTK_FILE_CHOOSER (window->button_simulator));
00877     if (gtk_toggle_button_get_active
00878     (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00879         input->evaluator = gtk_file_chooser_get_filename
00880         (GTK_FILE_CHOOSER (window->button_evaluator));
00881     else
00882         input->evaluator = NULL;
00883     if (input->type == INPUT_TYPE_XML)
00884     {
00885         input->result
00886         = (char *) xmlStrdup ((const xmlChar *)
00887         gtk_entry_get_text (window->entry_result));
00888         input->variables
00889         = (char *) xmlStrdup ((const xmlChar *)
00890         gtk_entry_get_text (window->entry_variables));
00891     }
00892     else
00893     {
00894         input->result = g_strdup (gtk_entry_get_text (window->
00895         entry_result));
00896         input->variables
00897         = g_strdup (gtk_entry_get_text (window->entry_variables));
00898     }
00899     // Setting the algorithm
00900     switch (window_get_algorithm ())
00901     {
00902         case ALGORITHM_MONTE_CARLO:
00903             input->algorithm = ALGORITHM_MONTE_CARLO;
00904             input->nsimulations
00905             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00906             input->niterations
00907             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00908             input->tolerance = gtk_spin_button_get_value (window->
00909             spin_tolerance);
00910             input->nbest = gtk_spin_button_get_value_as_int (window->
00911             spin_bests);
00912             window_save_direction ();
00913             break;
00914         case ALGORITHM_SWEEP:
00915             input->algorithm = ALGORITHM_SWEEP;
00916             input->niterations
00917             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00918             input->tolerance = gtk_spin_button_get_value (window->
00919             spin_tolerance);
00920             input->nbest = gtk_spin_button_get_value_as_int (window->
00921             spin_bests);
00922             window_save_direction ();
00923             break;
00924         default:
00925             input->algorithm = ALGORITHM_GENETIC;
00926             input->nsimulations
00927             = gtk_spin_button_get_value_as_int (window->spin_population);
00928             input->niterations
00929             = gtk_spin_button_get_value_as_int (window->spin_generations);
00930             input->mutation_ratio
00931             = gtk_spin_button_get_value (window->spin_mutation);
00932             input->reproduction_ratio
00933             = gtk_spin_button_get_value (window->spin_reproduction);
00934             input->adaptation_ratio
00935             = gtk_spin_button_get_value (window->spin_adaptation);
00936             break;
00937     }
00938     input->norm = window_get_norm ();
00939     input->p = gtk_spin_button_get_value (window->spin_p);
00940     input->threshold = gtk_spin_button_get_value (window->
00941     spin_threshold);
00942     // Saving the XML file
00943     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00944     input_save (buffer);
00945     // Closing and freeing memory
00946     g_free (buffer);
00947     gtk_widget_destroy (GTK_WIDGET (dlg));
00948     #if DEBUG_INTERFACE
00949     fprintf (stderr, "window_save: end\n");
00950     #endif
00951     return 1;
00952 }
00953 // Closing and freeing memory
00954 gtk_widget_destroy (GTK_WIDGET (dlg));
00955 #if DEBUG_INTERFACE
00956 fprintf (stderr, "window_save: end\n");

```

```

00955 #endif
00956     return 0;
00957 }
00958
00963 void
00964 window_run ()
00965 {
00966     unsigned int i;
00967     char *msg, *msg2, buffer[64], buffer2[64];
00968     #if DEBUG_INTERFACE
00969     fprintf (stderr, "window_run: start\n");
00970     #endif
00971     if (!window_save ())
00972     {
00973         #if DEBUG_INTERFACE
00974         fprintf (stderr, "window_run: end\n");
00975         #endif
00976         return;
00977     }
00978     running_new ();
00979     while (gtk_events_pending ())
00980         gtk_main_iteration ();
00981     optimize_open ();
00982     #if DEBUG_INTERFACE
00983     fprintf (stderr, "window_run: closing running dialog\n");
00984     #endif
00985     gtk_spinner_stop (running->spinner);
00986     gtk_widget_destroy (GTK_WIDGET (running->dialog));
00987     #if DEBUG_INTERFACE
00988     fprintf (stderr, "window_run: displaying results\n");
00989     #endif
00990     snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00991     msg2 = g_strdup (buffer);
00992     for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00993     {
00994         snprintf (buffer, 64, "%s = %s\n",
00995             input->variable[i].name, format[input->
00996             variable[i].precision]);
00997         snprintf (buffer2, 64, buffer, optimize->value_old[i]);
00998         msg = g_strconcat (msg2, buffer2, NULL);
00999         g_free (msg2);
01000     }
01001     snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
01002         optimize->calculation_time);
01003     msg = g_strconcat (msg2, buffer, NULL);
01004     g_free (msg2);
01005     show_message (gettext ("Best result"), msg, INFO_TYPE);
01006     g_free (msg);
01007     #if DEBUG_INTERFACE
01008     fprintf (stderr, "window_run: freeing memory\n");
01009     #endif
01010     optimize_free ();
01011     #if DEBUG_INTERFACE
01012     fprintf (stderr, "window_run: end\n");
01013     #endif
01014 }
01015
01019 void
01020 window_help ()
01021 {
01022     char *buffer, *buffer2;
01023     #if DEBUG_INTERFACE
01024     fprintf (stderr, "window_help: start\n");
01025     #endif
01026     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01027         gettext ("user-manual.pdf"), NULL);
01028     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01029     g_free (buffer2);
01030     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01031     #if DEBUG_INTERFACE
01032     fprintf (stderr, "window_help: uri=%s\n", buffer);
01033     #endif
01034     g_free (buffer);
01035     #if DEBUG_INTERFACE
01036     fprintf (stderr, "window_help: end\n");
01037     #endif
01038 }
01039
01044 void
01045 window_about ()
01046 {
01047     static const gchar *authors[] = {
01048         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01049         "Borja Latorre Garcés <borja.latorre@csic.es>",
01050         NULL
01051     };
01052     #if DEBUG_INTERFACE

```

```

01053     fprintf (stderr, "window_about: start\n");
01054 #endif
01055     gtk_show_about_dialog
01056     (window->window,
01057      "program_name", "MPCOTool",
01058      "comments",
01059      gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
01060              "A software to perform calibrations or optimizations of "
01061              "empirical parameters"),
01062      "authors", authors,
01063      "translator-credits",
01064      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01065      "(english, french and spanish)\n"
01066      "Uğur Çayoğlu (german)",
01067      "version", "3.0.4",
01068      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01069      "logo", window->logo,
01070      "website", "https://github.com/jburguete/mpcotool",
01071      "license-type", GTK_LICENSE_BSD, NULL);
01072 #if DEBUG_INTERFACE
01073     fprintf (stderr, "window_about: end\n");
01074 #endif
01075 }
01076
01082 void
01083 window_update_direction ()
01084 {
01085 #if DEBUG_INTERFACE
01086     fprintf (stderr, "window_update_direction: start\n");
01087 #endif
01088     gtk_widget_show (GTK_WIDGET (window->check_direction));
01089     if (gtk_toggle_button_get_active
01090         (GTK_TOGGLE_BUTTON (window->check_direction)))
01091     {
01092         gtk_widget_show (GTK_WIDGET (window->grid_direction));
01093         gtk_widget_show (GTK_WIDGET (window->label_step));
01094         gtk_widget_show (GTK_WIDGET (window->spin_step));
01095     }
01096     switch (window_get_direction ())
01097     {
01098     case DIRECTION_METHOD_COORDINATES:
01099         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01100         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01101         break;
01102     default:
01103         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01104         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01105     }
01106 #if DEBUG_INTERFACE
01107     fprintf (stderr, "window_update_direction: end\n");
01108 #endif
01109 }
01110
01115 void
01116 window_update ()
01117 {
01118     unsigned int i;
01119 #if DEBUG_INTERFACE
01120     fprintf (stderr, "window_update: start\n");
01121 #endif
01122     gtk_widget_set_sensitive
01123     (GTK_WIDGET (window->button_evaluator),
01124      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01125                                   (window->check_evaluator)));
01126     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01127     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01128     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01129     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01130     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01131     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01132     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01133     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01134     gtk_widget_hide (GTK_WIDGET (window->label_population));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01136     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01138     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01140     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01142     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01144     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01146     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01148     gtk_widget_hide (GTK_WIDGET (window->check_direction));

```

```

01149 gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01150 gtk_widget_hide (GTK_WIDGET (window->label_step));
01151 gtk_widget_hide (GTK_WIDGET (window->spin_step));
01152 gtk_widget_hide (GTK_WIDGET (window->label_p));
01153 gtk_widget_hide (GTK_WIDGET (window->spin_p));
01154 i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01155 switch (window_get_algorithm ())
01156 {
01157     case ALGORITHM_MONTE_CARLO:
01158         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01159         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01160         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01161         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01162         if (i > 1)
01163         {
01164             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01165             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01166             gtk_widget_show (GTK_WIDGET (window->label_bests));
01167             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01168         }
01169         window_update_direction ();
01170         break;
01171     case ALGORITHM_SWEEP:
01172         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01173         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01174         if (i > 1)
01175         {
01176             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01177             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01178             gtk_widget_show (GTK_WIDGET (window->label_bests));
01179             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01180         }
01181         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01182         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01183         gtk_widget_show (GTK_WIDGET (window->check_direction));
01184         window_update_direction ();
01185         break;
01186     default:
01187         gtk_widget_show (GTK_WIDGET (window->label_population));
01188         gtk_widget_show (GTK_WIDGET (window->spin_population));
01189         gtk_widget_show (GTK_WIDGET (window->label_generations));
01190         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01191         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01192         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01193         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01194         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01195         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01196         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01197         gtk_widget_show (GTK_WIDGET (window->label_bits));
01198         gtk_widget_show (GTK_WIDGET (window->spin_bits));
01199     }
01200 gtk_widget_set_sensitive
01201 (GTK_WIDGET (window->button_remove_experiment),
01202  input->nexperiments > 1);
01203 gtk_widget_set_sensitive
01204 (GTK_WIDGET (window->button_remove_variable), input->
01205  nvariables > 1);
01204 for (i = 0; i < input->experiment->ninputs; ++i)
01205 {
01206     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01207     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01208     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01209     gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01210     g_signal_handler_block
01211     (window->check_template[i], window->id_template[i]);
01212     g_signal_handler_block (window->button_template[i], window->
01213     id_input[i]);
01214     gtk_toggle_button_set_active
01215     (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
01216     g_signal_handler_unblock
01217     (window->button_template[i], window->id_input[i]);
01218     g_signal_handler_unblock
01219     (window->check_template[i], window->id_template[i]);
01220 }
01221 if (i > 0)
01222 {
01223     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01224     gtk_widget_set_sensitive
01225     (GTK_WIDGET (window->button_template[i - 1]),
01226      gtk_toggle_button_get_active
01227      (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
01228 }
01229 if (i < MAX_NINPUTS)
01230 {
01231     gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01232     gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01233     gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);

```

```

01233     gtk_widget_set_sensitive
01234     (GTK_WIDGET (window->button_template[i]),
01235     gtk_toggle_button_get_active
01236     GTK_TOGGLE_BUTTON (window->check_template[i]));
01237     g_signal_handler_block
01238     (window->check_template[i], window->id_template[i]);
01239     g_signal_handler_block (window->button_template[i], window->
id_input[i]);
01240     gtk_toggle_button_set_active
01241     (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
01242     g_signal_handler_unblock
01243     (window->button_template[i], window->id_input[i]);
01244     g_signal_handler_unblock
01245     (window->check_template[i], window->id_template[i]);
01246     }
01247     while (++i < MAX_NINPUTS)
01248     {
01249         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01250         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01251     }
01252     gtk_widget_set_sensitive
01253     (GTK_WIDGET (window->spin_minabs),
01254     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01255     gtk_widget_set_sensitive
01256     (GTK_WIDGET (window->spin_maxabs),
01257     gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01258     if (window_get_norm () == ERROR_NORM_P)
01259     {
01260         gtk_widget_show (GTK_WIDGET (window->label_p));
01261         gtk_widget_show (GTK_WIDGET (window->spin_p));
01262     }
01263     #if DEBUG_INTERFACE
01264     fprintf (stderr, "window_update: end\n");
01265     #endif
01266     }
01267
01272 void
01273 window_set_algorithm ()
01274 {
01275     int i;
01276     #if DEBUG_INTERFACE
01277     fprintf (stderr, "window_set_algorithm: start\n");
01278     #endif
01279     i = window_get_algorithm ();
01280     switch (i)
01281     {
01282         case ALGORITHM_SWEEP:
01283             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01284             if (i < 0)
01285                 i = 0;
01286             gtk_spin_button_set_value (window->spin_sweeps,
(gdouble) input->variable[i].
nsweeps);
01288             break;
01289         case ALGORITHM_GENETIC:
01290             i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01291             if (i < 0)
01292                 i = 0;
01293             gtk_spin_button_set_value (window->spin_bits,
(gdouble) input->variable[i].nbits);
01294             }
01295     }
01296     window_update ();
01297     #if DEBUG_INTERFACE
01298     fprintf (stderr, "window_set_algorithm: end\n");
01299     #endif
01300     }
01301
01306 void
01307 window_set_experiment ()
01308 {
01309     unsigned int i, j;
01310     char *buffer1, *buffer2;
01311     #if DEBUG_INTERFACE
01312     fprintf (stderr, "window_set_experiment: start\n");
01313     #endif
01314     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01315     gtk_spin_button_set_value (window->spin_weight, input->
experiment[i].weight);
01316     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01317     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01318     g_free (buffer1);
01319     g_signal_handler_block
01320     (window->button_experiment, window->id_experiment_name);
01321     gtk_file_chooser_set_filename
01322     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01323     g_signal_handler_unblock
01324     (window->button_experiment, window->id_experiment_name);

```

```

01325     g_free (buffer2);
01326     for (j = 0; j < input->experiment->ninputs; ++j)
01327     {
01328         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01329         buffer2 = g_build_filename (input->directory,
                                input->experiment[i].template[j], NULL);
01330         gtk_file_chooser_set_filename
01331             (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01332         g_free (buffer2);
01333         g_signal_handler_unblock
01334             (window->button_template[j], window->id_input[j]);
01335     }
01336 #if DEBUG_INTERFACE
01337     fprintf (stderr, "window_set_experiment: end\n");
01338 #endif
01339 }
01340
01341 void
01342 window_remove_experiment ()
01343 {
01344     unsigned int i, j;
01345 #if DEBUG_INTERFACE
01346     fprintf (stderr, "window_remove_experiment: start\n");
01347 #endif
01348     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01349     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01350     gtk_combo_box_text_remove (window->combo_experiment, i);
01351     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01352     experiment_free (input->experiment + i, input->
type);
01353     --input->nexperiments;
01354     for (j = i; j < input->nexperiments; ++j)
01355         memcpy (input->experiment + j, input->experiment + j + 1,
01356             sizeof (Experiment));
01357     j = input->nexperiments - 1;
01358     if (i > j)
01359         i = j;
01360     for (j = 0; j < input->experiment->ninputs; ++j)
01361         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01362     g_signal_handler_block
01363         (window->button_experiment, window->id_experiment_name);
01364     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01365     g_signal_handler_unblock
01366         (window->button_experiment, window->id_experiment_name);
01367     for (j = 0; j < input->experiment->ninputs; ++j)
01368         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01369     window_update ();
01370 #if DEBUG_INTERFACE
01371     fprintf (stderr, "window_remove_experiment: end\n");
01372 #endif
01373 }
01374
01375 void
01376 window_add_experiment ()
01377 {
01378     unsigned int i, j;
01379 #if DEBUG_INTERFACE
01380     fprintf (stderr, "window_add_experiment: start\n");
01381 #endif
01382     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01383     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01384     gtk_combo_box_text_insert_text
01385         (window->combo_experiment, i, input->experiment[i].
name);
01386     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01387     input->experiment = (Experiment *) g_realloc
01388         (input->experiment, (input->nexperiments + 1) * sizeof (
Experiment));
01389     for (j = input->nexperiments - 1; j > i; --j)
01390         memcpy (input->experiment + j + 1, input->experiment + j,
01391             sizeof (Experiment));
01392     input->experiment[j + 1].weight = input->experiment[j].
weight;
01393     input->experiment[j + 1].ninputs = input->
experiment[j].ninputs;
01394     if (input->type == INPUT_TYPE_XML)
01395     {
01396         input->experiment[j + 1].name
01397             = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
name);

```



```

01407         for (j = 0; j < input->experiment->ninputs; ++j)
01408             input->experiment[i + 1].template[j]
01409             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
template[j]);
01410     }
01411     else
01412     {
01413         input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
01414         for (j = 0; j < input->experiment->ninputs; ++j)
01415             input->experiment[i + 1].template[j]
01416             = g_strdup (input->experiment[i].template[j]);
01417     }
01418     ++input->nexperiments;
01419     for (j = 0; j < input->experiment->ninputs; ++j)
01420         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01421     g_signal_handler_block
01422     (window->button_experiment, window->id_experiment_name);
01423     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01424     g_signal_handler_unblock
01425     (window->button_experiment, window->id_experiment_name);
01426     for (j = 0; j < input->experiment->ninputs; ++j)
01427         g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01428     window_update ();
01429     #if DEBUG_INTERFACE
01430     fprintf (stderr, "window_add_experiment: end\n");
01431     #endif
01432 }
01433
01438 void
01439 window_name_experiment ()
01440 {
01441     unsigned int i;
01442     char *buffer;
01443     GFile *file1, *file2;
01444     #if DEBUG_INTERFACE
01445     fprintf (stderr, "window_name_experiment: start\n");
01446     #endif
01447     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01448     file1
01449     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01450     file2 = g_file_new_for_path (input->directory);
01451     buffer = g_file_get_relative_path (file2, file1);
01452     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01453     gtk_combo_box_text_remove (window->combo_experiment, i);
01454     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01455     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01456     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01457     g_free (buffer);
01458     g_object_unref (file2);
01459     g_object_unref (file1);
01460     #if DEBUG_INTERFACE
01461     fprintf (stderr, "window_name_experiment: end\n");
01462     #endif
01463 }
01464
01469 void
01470 window_weight_experiment ()
01471 {
01472     unsigned int i;
01473     #if DEBUG_INTERFACE
01474     fprintf (stderr, "window_weight_experiment: start\n");
01475     #endif
01476     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01477     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01478     #if DEBUG_INTERFACE
01479     fprintf (stderr, "window_weight_experiment: end\n");
01480     #endif
01481 }
01482
01488 void
01489 window_inputs_experiment ()
01490 {
01491     unsigned int j;
01492     #if DEBUG_INTERFACE
01493     fprintf (stderr, "window_inputs_experiment: start\n");
01494     #endif
01495     j = input->experiment->ninputs - 1;
01496     if (j
01497         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01498         (window->check_template[j])))
01499         --input->experiment->ninputs;

```

```

01500     if (input->experiment->ninputs < MAX_NINPUTS
01501         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01502             (window->check_template[j])))
01503         ++input->experiment->ninputs;
01504     window_update ();
01505 #if DEBUG_INTERFACE
01506     fprintf (stderr, "window_inputs_experiment: end\n");
01507 #endif
01508 }
01509
01510 void
01511 window_template_experiment (void *data)
01512 {
01513     unsigned int i, j;
01514     char *buffer;
01515     GFile *file1, *file2;
01516 #if DEBUG_INTERFACE
01517     fprintf (stderr, "window_template_experiment: start\n");
01518 #endif
01519     i = (size_t) data;
01520     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01521     file1
01522         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01523     file2 = g_file_new_for_path (input->directory);
01524     buffer = g_file_get_relative_path (file2, file1);
01525     if (input->type == INPUT_TYPE_XML)
01526         input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01527     else
01528         input->experiment[j].template[i] = g_strdup (buffer);
01529     g_free (buffer);
01530     g_object_unref (file2);
01531     g_object_unref (file1);
01532 #if DEBUG_INTERFACE
01533     fprintf (stderr, "window_template_experiment: end\n");
01534 #endif
01535 }
01536
01537 void
01538 window_set_variable ()
01539 {
01540     unsigned int i;
01541 #if DEBUG_INTERFACE
01542     fprintf (stderr, "window_set_variable: start\n");
01543 #endif
01544     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01545     g_signal_handler_block (window->entry_variable, window->
01546         id_variable_label);
01547     gtk_entry_set_text (window->entry_variable, input->variable[i].
01548         name);
01549     g_signal_handler_unblock (window->entry_variable, window->
01550         id_variable_label);
01551     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01552         rangemin);
01553     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01554         rangemax);
01555     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01556     {
01557         gtk_spin_button_set_value (window->spin_minabs,
01558             input->variable[i].rangeminabs);
01559         gtk_toggle_button_set_active
01560             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01561     }
01562     else
01563     {
01564         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01565         gtk_toggle_button_set_active
01566             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01567     }
01568     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01569     {
01570         gtk_spin_button_set_value (window->spin_maxabs,
01571             input->variable[i].rangemaxabs);
01572         gtk_toggle_button_set_active
01573             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01574     }
01575     else
01576     {
01577         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01578         gtk_toggle_button_set_active
01579             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01580     }
01581     gtk_spin_button_set_value (window->spin_precision,
01582         input->variable[i].precision);
01583     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01584         nsteps);
01585     if (input->nsteps)
01586         gtk_spin_button_set_value (window->spin_step, input->variable[i].

```

```

        step);
01592 #if DEBUG_INTERFACE
01593     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01594             input->variable[i].precision);
01595 #endif
01596     switch (window_get_algorithm ())
01597     {
01598     case ALGORITHM_SWEEP:
01599         gtk_spin_button_set_value (window->spin_sweeps,
01600                                   (gdouble) input->variable[i].
01601                                   nsweeps);
01602 #if DEBUG_INTERFACE
01603     fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01604             input->variable[i].nsweeps);
01605 #endif
01606         break;
01607     case ALGORITHM_GENETIC:
01608         gtk_spin_button_set_value (window->spin_bits,
01609                                   (gdouble) input->variable[i].nbits);
01610 #if DEBUG_INTERFACE
01611     fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01612             input->variable[i].nbits);
01613 #endif
01614         break;
01615     }
01616     window_update ();
01617 #if DEBUG_INTERFACE
01618     fprintf (stderr, "window_set_variable: end\n");
01619 #endif
01620 }
01621 void
01622 window_remove_variable ()
01623 {
01624     unsigned int i, j;
01625 #if DEBUG_INTERFACE
01626     fprintf (stderr, "window_remove_variable: start\n");
01627 #endif
01628     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01629     g_signal_handler_block (window->combo_variable, window->
01630                             id_variable);
01631     gtk_combo_box_text_remove (window->combo_variable, i);
01632     g_signal_handler_unblock (window->combo_variable, window->
01633                              id_variable);
01634     xmlFree (input->variable[i].name);
01635     --input->nvariables;
01636     for (j = i; j < input->nvariables; ++j)
01637         memcpy (input->variable + j, input->variable + j + 1, sizeof (
01638                 Variable));
01639     j = input->nvariables - 1;
01640     if (i > j)
01641         i = j;
01642     g_signal_handler_block (window->entry_variable, window->
01643                             id_variable_label);
01644     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01645     g_signal_handler_unblock (window->entry_variable, window->
01646                              id_variable_label);
01647     window_update ();
01648 #if DEBUG_INTERFACE
01649     fprintf (stderr, "window_remove_variable: end\n");
01650 #endif
01651 }
01652 void
01653 window_add_variable ()
01654 {
01655     unsigned int i, j;
01656 #if DEBUG_INTERFACE
01657     fprintf (stderr, "window_add_variable: start\n");
01658 #endif
01659     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01660     g_signal_handler_block (window->combo_variable, window->
01661                             id_variable);
01662     gtk_combo_box_text_insert_text (window->combo_variable, i,
01663                                    input->variable[i].name);
01664     g_signal_handler_unblock (window->combo_variable, window->
01665                              id_variable);
01666     input->variable = (Variable *) g_realloc
01667         (input->variable, (input->nvariables + 1) * sizeof (
01668         Variable));
01669     for (j = input->nvariables - 1; j > i; --j)
01670         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01671                 Variable));
01672     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01673             Variable));
01674     if (input->type == INPUT_TYPE_XML)
01675         input->variable[j + 1].name

```

```

01675     = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01676     else
01677         input->variable[j + 1].name = g_strdup (input->
variable[j].name);
01678     ++input->nvariables;
01679     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01680     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01681     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01682     window_update ();
01683     #if DEBUG_INTERFACE
01684     fprintf (stderr, "window_add_variable: end\n");
01685     #endif
01686 }
01687
01692 void
01693 window_label_variable ()
01694 {
01695     unsigned int i;
01696     const char *buffer;
01697     #if DEBUG_INTERFACE
01698     fprintf (stderr, "window_label_variable: start\n");
01699     #endif
01700     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01701     buffer = gtk_entry_get_text (window->entry_variable);
01702     g_signal_handler_block (window->combo_variable, window->
id_variable);
01703     gtk_combo_box_text_remove (window->combo_variable, i);
01704     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01705     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01706     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01707     #if DEBUG_INTERFACE
01708     fprintf (stderr, "window_label_variable: end\n");
01709     #endif
01710 }
01711
01716 void
01717 window_precision_variable ()
01718 {
01719     unsigned int i;
01720     #if DEBUG_INTERFACE
01721     fprintf (stderr, "window_precision_variable: start\n");
01722     #endif
01723     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01724     input->variable[i].precision
01725     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01726     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
precision);
01727     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
precision);
01728     gtk_spin_button_set_digits (window->spin_minabs,
input->variable[i].precision);
01729     gtk_spin_button_set_digits (window->spin_maxabs,
input->variable[i].precision);
01730     #if DEBUG_INTERFACE
01731     fprintf (stderr, "window_precision_variable: end\n");
01732     #endif
01733 }
01734
01741 void
01742 window_rangemin_variable ()
01743 {
01744     unsigned int i;
01745     #if DEBUG_INTERFACE
01746     fprintf (stderr, "window_rangemin_variable: start\n");
01747     #endif
01748     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01749     input->variable[i].rangemin = gtk_spin_button_get_value (window->
spin_min);
01750     #if DEBUG_INTERFACE
01751     fprintf (stderr, "window_rangemin_variable: end\n");
01752     #endif
01753 }
01754
01759 void
01760 window_rangemax_variable ()
01761 {
01762     unsigned int i;
01763     #if DEBUG_INTERFACE
01764     fprintf (stderr, "window_rangemax_variable: start\n");
01765     #endif
01766     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01767     input->variable[i].rangemax = gtk_spin_button_get_value (window->
spin_max);
01768     #if DEBUG_INTERFACE

```

```

01769     fprintf (stderr, "window_rangemax_variable: end\n");
01770 #endif
01771 }
01772
01773 void
01774 window_rangeminabs_variable ()
01775 {
01776     unsigned int i;
01777     #if DEBUG_INTERFACE
01778     fprintf (stderr, "window_rangeminabs_variable: start\n");
01779     #endif
01780     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01781     input->variable[i].rangeminabs
01782     = gtk_spin_button_get_value (window->spin_minabs);
01783     #if DEBUG_INTERFACE
01784     fprintf (stderr, "window_rangeminabs_variable: end\n");
01785     #endif
01786 }
01787
01788 void
01789 window_rangemaxabs_variable ()
01790 {
01791     unsigned int i;
01792     #if DEBUG_INTERFACE
01793     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01794     #endif
01795     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01796     input->variable[i].rangemaxabs
01797     = gtk_spin_button_get_value (window->spin_maxabs);
01798     #if DEBUG_INTERFACE
01799     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01800     #endif
01801 }
01802
01803 void
01804 window_step_variable ()
01805 {
01806     unsigned int i;
01807     #if DEBUG_INTERFACE
01808     fprintf (stderr, "window_step_variable: start\n");
01809     #endif
01810     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01811     input->variable[i].step = gtk_spin_button_get_value (window->
01812     spin_step);
01813     #if DEBUG_INTERFACE
01814     fprintf (stderr, "window_step_variable: end\n");
01815     #endif
01816 }
01817
01818 void
01819 window_update_variable ()
01820 {
01821     int i;
01822     #if DEBUG_INTERFACE
01823     fprintf (stderr, "window_update_variable: start\n");
01824     #endif
01825     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01826     if (i < 0)
01827         i = 0;
01828     switch (window_get_algorithm ())
01829     {
01830     case ALGORITHM_SWEEP:
01831         input->variable[i].nsweeps
01832         = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01833         #if DEBUG_INTERFACE
01834         fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01835         input->variable[i].nsweeps);
01836         #endif
01837         break;
01838     case ALGORITHM_GENETIC:
01839         input->variable[i].nbits
01840         = gtk_spin_button_get_value_as_int (window->spin_bits);
01841         #if DEBUG_INTERFACE
01842         fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01843         input->variable[i].nbits);
01844         #endif
01845     }
01846     #if DEBUG_INTERFACE
01847     fprintf (stderr, "window_update_variable: end\n");
01848     #endif
01849 }
01850
01851 int
01852 window_read (char *filename)
01853 {
01854     unsigned int i;
01855     char *buffer;

```

```

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

```

```

01956         window->id_experiment_name);
01957     gtk_combo_box_text_remove_all (window->combo_experiment);
01958     for (i = 0; i < input->nexperiments; ++i)
01959         gtk_combo_box_text_append_text (window->combo_experiment,
01960                                         input->experiment[i].name);
01961     g_signal_handler_unblock
01962         (window->button_experiment, window->id_experiment_name);
01963     g_signal_handler_unblock (window->combo_experiment, window->
01964                             id_experiment);
01964     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01965     g_signal_handler_block (window->combo_variable, window->
01966                             id_variable);
01966     g_signal_handler_block (window->entry_variable, window->
01967                             id_variable_label);
01967     gtk_combo_box_text_remove_all (window->combo_variable);
01968     for (i = 0; i < input->nvariables; ++i)
01969         gtk_combo_box_text_append_text (window->combo_variable,
01970                                         input->variable[i].name);
01970     g_signal_handler_unblock (window->entry_variable, window->
01971                             id_variable_label);
01972     g_signal_handler_unblock (window->combo_variable, window->
01973                             id_variable);
01973     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01974     window_set_variable ();
01975     window_update ();
01976
01977 #if DEBUG_INTERFACE
01978     fprintf (stderr, "window_read: end\n");
01979 #endif
01980     return 1;
01981 }
01982
01987 void
01988 window_open ()
01989 {
01990     GtkFileChooserDialog *dlg;
01991     GtkFileFilter *filter;
01992     char *buffer, *directory, *name;
01993
01994 #if DEBUG_INTERFACE
01995     fprintf (stderr, "window_open: start\n");
01996 #endif
01997
01998     // Saving a backup of the current input file
01999     directory = g_strdup (input->directory);
02000     name = g_strdup (input->name);
02001
02002     // Opening dialog
02003     dlg = (GtkFileChooserDialog *)
02004         gtk_file_chooser_dialog_new (gettext ("Open input file"),
02005                                     window->window,
02006                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02007                                     gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02008                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02009
02010     // Adding XML filter
02011     filter = (GtkFileFilter *) gtk_file_filter_new ();
02012     gtk_file_filter_set_name (filter, "XML");
02013     gtk_file_filter_add_pattern (filter, "*.xml");
02014     gtk_file_filter_add_pattern (filter, "*.XML");
02015     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02016
02017     // Adding JSON filter
02018     filter = (GtkFileFilter *) gtk_file_filter_new ();
02019     gtk_file_filter_set_name (filter, "JSON");
02020     gtk_file_filter_add_pattern (filter, "*.json");
02021     gtk_file_filter_add_pattern (filter, "*.JSON");
02022     gtk_file_filter_add_pattern (filter, "*.js");
02023     gtk_file_filter_add_pattern (filter, "*.JS");
02024     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02025
02026     // If OK saving
02027     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02028     {
02029         // Trying to open the input file
02030         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02031         if (!window_read (buffer))
02032         {
02033             #if DEBUG_INTERFACE
02034                 fprintf (stderr, "window_open: error reading input file\n");
02035             #endif
02036             g_free (buffer);
02037
02038             // Reading backup file on error
02039             buffer = g_build_filename (directory, name, NULL);
02040             if (!input_open (buffer))

```

```

02042         {
02043
02044             // Closing on backup file reading error
02045             #if DEBUG_INTERFACE
02046                 fprintf (stderr, "window_read: error reading backup file\n");
02047             #endif
02048             g_free (buffer);
02049             break;
02050         }
02051         g_free (buffer);
02052     }
02053     else
02054     {
02055         g_free (buffer);
02056         break;
02057     }
02058 }
02059
02060 // Freeing and closing
02061 g_free (name);
02062 g_free (directory);
02063 gtk_widget_destroy (GTK_WIDGET (dlg));
02064 #if DEBUG_INTERFACE
02065     fprintf (stderr, "window_open: end\n");
02066 #endif
02067 }
02068
02073 void
02074 window_new ()
02075 {
02076     unsigned int i;
02077     char *buffer, *buffer2, buffer3[64];
02078     char *label_algorithm[NALGORITHMS] = {
02079         "_Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
02080     };
02081     char *tip_algorithm[NALGORITHMS] = {
02082         gettext ("Monte-Carlo brute force algorithm"),
02083         gettext ("Sweep brute force algorithm"),
02084         gettext ("Genetic algorithm")
02085     };
02086     char *label_direction[N DIRECTIONS] = {
02087         gettext ("_Coordinates descent"), gettext ("_Random")
02088     };
02089     char *tip_direction[N DIRECTIONS] = {
02090         gettext ("Coordinates direction estimate method"),
02091         gettext ("Random direction estimate method")
02092     };
02093     char *label_norm[N NORMS] = { "L2", "L", "Lp", "L1" };
02094     char *tip_norm[N NORMS] = {
02095         gettext ("Euclidean error norm (L2)"),
02096         gettext ("Maximum error norm (L)"),
02097         gettext ("P error norm (Lp)"),
02098         gettext ("Taxicab error norm (L1)")
02099     };
02100
02101     #if DEBUG_INTERFACE
02102         fprintf (stderr, "window_new: start\n");
02103     #endif
02104
02105     // Creating the window
02106     window->window = main_window
02107         = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
02108
02109     // Finish when closing the window
02110     g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
02111
02112     // Setting the window title
02113     gtk_window_set_title (window->window, "MPCOTool");
02114
02115     // Creating the open button
02116     window->button_open = (GtkToolButton *) gtk_tool_button_new
02117         (gtk_image_new_from_icon_name ("document-open",
02118             GTK_ICON_SIZE_LARGE_TOOLBAR),
02119         gettext ("Open"));
02120     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02121
02122     // Creating the save button
02123     window->button_save = (GtkToolButton *) gtk_tool_button_new
02124         (gtk_image_new_from_icon_name ("document-save",
02125             GTK_ICON_SIZE_LARGE_TOOLBAR),
02126         gettext ("Save"));
02127     g_signal_connect (window->button_save, "clicked", (void (*)(
02128         window_save,
02129         NULL));
02130
02131     // Creating the run button
02132     window->button_run = (GtkToolButton *) gtk_tool_button_new

```



```

02132     (gtk_image_new_from_icon_name ("system-run",
02133                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02134     gettext ("Run"));
02135     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02136
02137     // Creating the options button
02138     window->button_options = (GtkToolButton *) gtk_tool_button_new
02139     (gtk_image_new_from_icon_name ("preferences-system",
02140                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02141     gettext ("Options"));
02142     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02143
02144     // Creating the help button
02145     window->button_help = (GtkToolButton *) gtk_tool_button_new
02146     (gtk_image_new_from_icon_name ("help-browser",
02147                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02148     gettext ("Help"));
02149     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02150
02151     // Creating the about button
02152     window->button_about = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("help-about",
02154                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02155     gettext ("About"));
02156     g_signal_connect (window->button_about, "clicked", window_about, NULL);
02157
02158     // Creating the exit button
02159     window->button_exit = (GtkToolButton *) gtk_tool_button_new
02160     (gtk_image_new_from_icon_name ("application-exit",
02161                                   GTK_ICON_SIZE_LARGE_TOOLBAR),
02162     gettext ("Exit"));
02163     g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
02164
02165     // Creating the buttons bar
02166     window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02167     gtk_toolbar_insert
02168     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02169     gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02171     gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02173     gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02175     gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02177     gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02179     gtk_toolbar_insert
02180     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02181     gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02182
02183     // Creating the simulator program label and entry
02184     window->label_simulator
02185     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
02186     window->button_simulator = (GtkFileChooserButton *)
02187     gtk_file_chooser_button_new (gettext ("Simulator program"),
02188                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02189     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02190                                   gettext ("Simulator program executable file"));
02191     gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02192
02193     // Creating the evaluator program label and entry
02194     window->check_evaluator = (GtkCheckButton *)
02195     gtk_check_button_new_with_mnemonic (gettext ("Evaluator program"));
02196     g_signal_connect (window->check_evaluator, "toggled",
02197 window_update, NULL);
02197     window->button_evaluator = (GtkFileChooserButton *)
02198     gtk_file_chooser_button_new (gettext ("Evaluator program"),
02199                                   GTK_FILE_CHOOSER_ACTION_OPEN);
02200     gtk_widget_set_tooltip_text
02201     (GTK_WIDGET (window->button_evaluator),
02202     gettext ("Optional evaluator program executable file"));
02203
02204     // Creating the results files labels and entries
02205     window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
02206     window->entry_result = (GtkEntry *) gtk_entry_new ();
02207     gtk_widget_set_tooltip_text
02208     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
02209     window->label_variables
02210     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
02211     window->entry_variables = (GtkEntry *) gtk_entry_new ();
02212     gtk_widget_set_tooltip_text
02213     (GTK_WIDGET (window->entry_variables),
02214     gettext ("All simulated results file"));
02215
02216     // Creating the files grid and attaching widgets
02217     window->grid_files = (GtkGrid *) gtk_grid_new ();

```

```

02218   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02219                       0, 0, 1, 1);
02220   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02221                       1, 0, 1, 1);
02222   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02223                       0, 1, 1, 1);
02224   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02225                       1, 1, 1, 1);
02226   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02227                       0, 2, 1, 1);
02228   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02229                       1, 2, 1, 1);
02230   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02231                       0, 3, 1, 1);
02232   gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02233                       1, 3, 1, 1);
02234
02235   // Creating the algorithm properties
02236   window->label_simulations = (GtkLabel *) gtk_label_new
(gettext ("Simulations number"));
02237   window->spin_simulations
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02238   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_simulations),
02239       gettext ("Number of simulations to perform for each iteration"));
02240   gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02241   window->label_iterations = (GtkLabel *)
02242   gtk_label_new (gettext ("Iterations number"));
02243   window->spin_iterations
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02244   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
02245   g_signal_connect
(window->spin_iterations, "value-changed", window_update, NULL);
02246   gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02247   window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
02248   window->spin_tolerance
= (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02249   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_tolerance),
02250       gettext ("Tolerance to set the variable interval on the next iteration"));
02251   window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
02252   window->spin_bests
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02253   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_bests),
02254       gettext ("Number of best simulations used to set the variable interval "
02255           "on the next iteration"));
02256   window->label_population
= (GtkLabel *) gtk_label_new (gettext ("Population number"));
02257   window->spin_population
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02258   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_population),
02259       gettext ("Number of population for the genetic algorithm"));
02260   gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_population), TRUE);
02261   window->label_generations
= (GtkLabel *) gtk_label_new (gettext ("Generations number"));
02262   window->spin_generations
= (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02263   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_generations),
02264       gettext ("Number of generations for the genetic algorithm"));
02265   window->label_mutation
= (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
02266   window->spin_mutation
= (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02267   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_mutation),
02268       gettext ("Ratio of mutation for the genetic algorithm"));
02269   window->label_reproduction
= (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
02270   window->spin_reproduction
= (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02271   gtk_widget_set_tooltip_text
(GTK_WIDGET (window->spin_reproduction),
02272       gettext ("Ratio of reproduction for the genetic algorithm"));
02273   window->label_adaptation
= (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
02274

```

```

02297 window->spin_adaptation
02298 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02299 gtk_widget_set_tooltip_text
02300 (GTK_WIDGET (window->spin_adaptation),
02301  gettext ("Ratio of adaptation for the genetic algorithm"));
02302 window->label_threshold = (GtkLabel *) gtk_label_new (gettext ("Threshold"));
02303 window->spin_threshold = (GtkSpinButton *) gtk_spin_button_new_with_range
02304 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02305 gtk_widget_set_tooltip_text
02306 (GTK_WIDGET (window->spin_threshold),
02307  gettext ("Threshold in the objective function to finish the simulations"));
02308 window->scrolled_threshold
02309 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02310 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02311  GTK_WIDGET (window->spin_threshold));
02312 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02313 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02314 //  GTK_ALIGN_FILL);
02315
02316 // Creating the direction search method properties
02317 window->check_direction = (GtkCheckButton *)
02318  gtk_check_button_new_with_mnemonic (gettext ("_Direction search method"));
02319 g_signal_connect (window->check_direction, "clicked",
02320  window_update, NULL);
02321 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02322 window->button_direction[0] = (GtkRadioButton *)
02323  gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02324 gtk_grid_attach (window->grid_direction,
02325  GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02326 g_signal_connect (window->button_direction[0], "clicked",
02327  window_update, NULL);
02328 for (i = 0; ++i < NDIRECTIONS;)
02329 {
02330  window->button_direction[i] = (GtkRadioButton *)
02331  gtk_radio_button_new_with_mnemonic
02332  (gtk_radio_button_get_group (window->button_direction[0]),
02333   label_direction[i]);
02334  gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02335   tip_direction[i]);
02336  gtk_grid_attach (window->grid_direction,
02337   GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02338  g_signal_connect (window->button_direction[i], "clicked",
02339   window_update, NULL);
02340 }
02341 window->label_steps = (GtkLabel *) gtk_label_new (gettext ("Steps number"));
02342 window->spin_steps = (GtkSpinButton *)
02343  gtk_spin_button_new_with_range (1., 1.e12, 1.);
02344 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02345 window->label_estimates
02346 = (GtkLabel *) gtk_label_new (gettext ("Direction estimates number"));
02347 window->spin_estimates = (GtkSpinButton *)
02348  gtk_spin_button_new_with_range (1., 1.e3, 1.);
02349 window->label_relaxation
02350 = (GtkLabel *) gtk_label_new (gettext ("Relaxation parameter"));
02351 window->spin_relaxation = (GtkSpinButton *)
02352  gtk_spin_button_new_with_range (0., 2., 0.001);
02353 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02354  label_steps),
02355  0, NDIRECTIONS, 1, 1);
02356 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02357  spin_steps),
02358  1, NDIRECTIONS, 1, 1);
02359 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02360  label_estimates),
02361  0, NDIRECTIONS + 1, 1, 1);
02362 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02363  spin_estimates),
02364  1, NDIRECTIONS + 1, 1, 1);
02365 gtk_grid_attach (window->grid_direction,
02366  GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2, 1,
02367  1);
02368 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02369  spin_relaxation),
02370  1, NDIRECTIONS + 2, 1, 1);
02371
02372 // Creating the array of algorithms
02373 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02374 window->button_algorithm[0] = (GtkRadioButton *)
02375  gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02376 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02377  tip_algorithm[0]);
02378 gtk_grid_attach (window->grid_algorithm,
02379  GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02380 g_signal_connect (window->button_algorithm[0], "clicked",
02381  window_set_algorithm, NULL);
02382 for (i = 0; ++i < NALGORITHMS;)

```

```

02377     {
02378         window->button_algorithm[i] = (GtkRadioButton *)
02379             gtk_radio_button_new_with_mnemonic
02380             (gtk_radio_button_get_group (window->button_algorithm[0]),
02381              label_algorithm[i]);
02382         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02383                                     tip_algorithm[i]);
02384         gtk_grid_attach (window->grid_algorithm,
02385                         GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02386         g_signal_connect (window->button_algorithm[i], "clicked",
02387                         window_set_algorithm, NULL);
02388     }
02389     gtk_grid_attach (window->grid_algorithm,
02390                     GTK_WIDGET (window->label_simulations), 0,
02391                     NALGORITHMS, 1, 1);
02392     gtk_grid_attach (window->grid_algorithm,
02393                     GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1, 1);
02394     gtk_grid_attach (window->grid_algorithm,
02395                     GTK_WIDGET (window->label_iterations), 0,
02396                     NALGORITHMS + 1, 1, 1);
02397     gtk_grid_attach (window->grid_algorithm,
02398                     GTK_WIDGET (window->spin_iterations), 1,
02399                     NALGORITHMS + 1, 1, 1);
02400     gtk_grid_attach (window->grid_algorithm,
02401                     GTK_WIDGET (window->label_tolerance), 0,
02402                     NALGORITHMS + 2, 1, 1);
02403     gtk_grid_attach (window->grid_algorithm,
02404                     GTK_WIDGET (window->spin_tolerance), 1,
02405                     NALGORITHMS + 2, 1, 1);
02406     gtk_grid_attach (window->grid_algorithm,
02407                     GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02408     gtk_grid_attach (window->grid_algorithm,
02409                     GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02410     gtk_grid_attach (window->grid_algorithm,
02411                     GTK_WIDGET (window->label_population), 0,
02412                     NALGORITHMS + 4, 1, 1);
02413     gtk_grid_attach (window->grid_algorithm,
02414                     GTK_WIDGET (window->spin_population), 1,
02415                     NALGORITHMS + 4, 1, 1);
02416     gtk_grid_attach (window->grid_algorithm,
02417                     GTK_WIDGET (window->label_generations), 0,
02418                     NALGORITHMS + 5, 1, 1);
02419     gtk_grid_attach (window->grid_algorithm,
02420                     GTK_WIDGET (window->spin_generations), 1,
02421                     NALGORITHMS + 5, 1, 1);
02422     gtk_grid_attach (window->grid_algorithm,
02423                     GTK_WIDGET (window->label_mutation), 0,
02424                     NALGORITHMS + 6, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm,
02426                     GTK_WIDGET (window->spin_mutation), 1,
02427                     NALGORITHMS + 6, 1, 1);
02428     gtk_grid_attach (window->grid_algorithm,
02429                     GTK_WIDGET (window->label_reproduction), 0,
02430                     NALGORITHMS + 7, 1, 1);
02431     gtk_grid_attach (window->grid_algorithm,
02432                     GTK_WIDGET (window->spin_reproduction), 1,
02433                     NALGORITHMS + 7, 1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435                     GTK_WIDGET (window->label_adaptation), 0,
02436                     NALGORITHMS + 8, 1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438                     GTK_WIDGET (window->spin_adaptation), 1,
02439                     NALGORITHMS + 8, 1, 1);
02440     gtk_grid_attach (window->grid_algorithm,
02441                     GTK_WIDGET (window->check_direction), 0,
02442                     NALGORITHMS + 9, 2, 1);
02443     gtk_grid_attach (window->grid_algorithm,
02444                     GTK_WIDGET (window->grid_direction), 0,
02445                     NALGORITHMS + 10, 2, 1);
02446     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_threshold),
02447                     0, NALGORITHMS + 11, 1, 1);
02448     gtk_grid_attach (window->grid_algorithm,
02449                     GTK_WIDGET (window->scrolled_threshold), 1,
02450                     NALGORITHMS + 11, 1, 1);
02451     window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02452     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02453                     GTK_WIDGET (window->grid_algorithm));
02454
02455     // Creating the variable widgets
02456     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02457     gtk_widget_set_tooltip_text
02458         (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02459     window->id_variable = g_signal_connect
02460         (window->combo_variable, "changed", window_set_variable, NULL);
02461     window->button_add_variable
02462         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",

```

```

02463                                     GTK_ICON_SIZE_BUTTON);
02464     g_signal_connect
02465     (window->button_add_variable, "clicked",
window_add_variable, NULL);
02466     gtk_widget_set_tooltip_text
02467     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02468     window->button_remove_variable
02469     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02470                                     GTK_ICON_SIZE_BUTTON);
02471     g_signal_connect
02472     (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
02473     gtk_widget_set_tooltip_text
02474     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02475     window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02476     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02477     gtk_widget_set_tooltip_text
02478     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02479     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02480     window->id_variable_label = g_signal_connect
02481     (window->entry_variable, "changed", window_label_variable, NULL);
02482     window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02483     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02484     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02485     gtk_widget_set_tooltip_text
02486     (GTK_WIDGET (window->spin_min),
02487     gettext ("Minimum initial value of the variable"));
02488     window->scrolled_min
02489     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02490     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02491     GTK_WIDGET (window->spin_min));
02492     g_signal_connect (window->spin_min, "value-changed",
02493     window_rangemin_variable, NULL);
02494     window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02495     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02496     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02497     gtk_widget_set_tooltip_text
02498     (GTK_WIDGET (window->spin_max),
02499     gettext ("Maximum initial value of the variable"));
02500     window->scrolled_max
02501     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02502     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02503     GTK_WIDGET (window->spin_max));
02504     g_signal_connect (window->spin_max, "value-changed",
02505     window_rangemax_variable, NULL);
02506     window->check_minabs = (GtkCheckButton *)
02507     gtk_check_button_new_with_mnemonic (gettext ("Absolute minimum"));
02508     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02509     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02510     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02511     gtk_widget_set_tooltip_text
02512     (GTK_WIDGET (window->spin_minabs),
02513     gettext ("Minimum allowed value of the variable"));
02514     window->scrolled_minabs
02515     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02516     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02517     GTK_WIDGET (window->spin_minabs));
02518     g_signal_connect (window->spin_minabs, "value-changed",
02519     window_rangeminabs_variable, NULL);
02520     window->check_maxabs = (GtkCheckButton *)
02521     gtk_check_button_new_with_mnemonic (gettext ("Absolute maximum"));
02522     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02523     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02524     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02525     gtk_widget_set_tooltip_text
02526     (GTK_WIDGET (window->spin_maxabs),
02527     gettext ("Maximum allowed value of the variable"));
02528     window->scrolled_maxabs
02529     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02530     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02531     GTK_WIDGET (window->spin_maxabs));
02532     g_signal_connect (window->spin_maxabs, "value-changed",
02533     window_rangemaxabs_variable, NULL);
02534     window->label_precision
02535     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02536     window->spin_precision = (GtkSpinButton *)
02537     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02538     gtk_widget_set_tooltip_text
02539     (GTK_WIDGET (window->spin_precision),
02540     gettext ("Number of precision floating point digits\n"
02541     "0 is for integer numbers"));
02542     g_signal_connect (window->spin_precision, "value-changed",
02543     window_precision_variable, NULL);
02544     window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02545     window->spin_sweeps
02546     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02547     gtk_widget_set_tooltip_text

```

```

02548     (GTK_WIDGET (window->spin_sweeps),
02549     gettext ("Number of steps sweeping the variable"));
02550 g_signal_connect
02551 (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02552 window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02553 window->spin_bits
02554 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02555 gtk_widget_set_tooltip_text
02556 (GTK_WIDGET (window->spin_bits),
02557 gettext ("Number of bits to encode the variable"));
02558 g_signal_connect
02559 (window->spin_bits, "value-changed", window_update_variable, NULL);
02560 window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02561 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02562 (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02563 gtk_widget_set_tooltip_text
02564 (GTK_WIDGET (window->spin_step),
02565 gettext ("Initial step size for the direction search method"));
02566 window->scrolled_step
02567 = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02568 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02569 GTK_WIDGET (window->spin_step));
02570 g_signal_connect
02571 (window->spin_step, "value-changed", window_step_variable, NULL);
02572 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02573 gtk_grid_attach (window->grid_variable,
02574 GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02575 gtk_grid_attach (window->grid_variable,
02576 GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02577 gtk_grid_attach (window->grid_variable,
02578 GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02579 gtk_grid_attach (window->grid_variable,
02580 GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02581 gtk_grid_attach (window->grid_variable,
02582 GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02583 gtk_grid_attach (window->grid_variable,
02584 GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02585 gtk_grid_attach (window->grid_variable,
02586 GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02587 gtk_grid_attach (window->grid_variable,
02588 GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02589 gtk_grid_attach (window->grid_variable,
02590 GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02591 gtk_grid_attach (window->grid_variable,
02592 GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02593 gtk_grid_attach (window->grid_variable,
02594 GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02595 gtk_grid_attach (window->grid_variable,
02596 GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02597 gtk_grid_attach (window->grid_variable,
02598 GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02599 gtk_grid_attach (window->grid_variable,
02600 GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02601 gtk_grid_attach (window->grid_variable,
02602 GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02603 gtk_grid_attach (window->grid_variable,
02604 GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02605 gtk_grid_attach (window->grid_variable,
02606 GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02607 gtk_grid_attach (window->grid_variable,
02608 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02609 gtk_grid_attach (window->grid_variable,
02610 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02611 gtk_grid_attach (window->grid_variable,
02612 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02613 gtk_grid_attach (window->grid_variable,
02614 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02615 window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02616 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02617 GTK_WIDGET (window->grid_variable));
02618
02619 // Creating the experiment widgets
02620 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02621 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02622 gettext ("Experiment selector"));
02623 window->id_experiment = g_signal_connect
02624 (window->combo_experiment, "changed", window_set_experiment, NULL)
;
02625 window->button_add_experiment
02626 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02627 GTK_ICON_SIZE_BUTTON);
02628 g_signal_connect
02629 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02630 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02631 gettext ("Add experiment"));
02632 window->button_remove_experiment

```



```

02633     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02634                                                    GTK_ICON_SIZE_BUTTON);
02635 g_signal_connect (window->button_remove_experiment, "clicked",
02636                  window_remove_experiment, NULL);
02637 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02638                              gettext ("Remove experiment"));
02639 window->label_experiment
02640     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02641 window->button_experiment = (GtkFileChooserButton *)
02642     gtk_file_chooser_button_new (gettext ("Experimental data file"),
02643                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02644 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02645                              gettext ("Experimental data file"));
02646 window->id_experiment_name
02647     = g_signal_connect (window->button_experiment, "selection-changed",
02648                        window_name_experiment, NULL);
02649 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_experiment), TRUE);
02650 window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02651 window->spin_weight
02652     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02653 gtk_widget_set_tooltip_text
02654     (GTK_WIDGET (window->spin_weight),
02655      gettext ("Weight factor to build the objective function"));
02656 g_signal_connect
02657     (window->spin_weight, "value-changed", window_weight_experiment,
02658      NULL);
02659 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02660 gtk_grid_attach (window->grid_experiment,
02661                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02662 gtk_grid_attach (window->grid_experiment,
02663                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02664 gtk_grid_attach (window->grid_experiment,
02665                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02666 gtk_grid_attach (window->grid_experiment,
02667                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02668 gtk_grid_attach (window->grid_experiment,
02669                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02670 gtk_grid_attach (window->grid_experiment,
02671                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02672 gtk_grid_attach (window->grid_experiment,
02673                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02674 for (i = 0; i < MAX_NINPUTS; ++i)
02675 {
02676     snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02677     window->check_template[i] = (GtkCheckButton *)
02678     gtk_check_button_new_with_label (buffer3);
02679     window->id_template[i]
02680     = g_signal_connect (window->check_template[i], "toggled",
02681                        window_inputs_experiment, NULL);
02682     gtk_grid_attach (window->grid_experiment,
02683                     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02684     window->button_template[i] = (GtkFileChooserButton *)
02685     gtk_file_chooser_button_new (gettext ("Input template"),
02686                                  GTK_FILE_CHOOSER_ACTION_OPEN);
02687     gtk_widget_set_tooltip_text
02688     (GTK_WIDGET (window->button_template[i]),
02689      gettext ("Experimental input template file"));
02690     window->id_input[i]
02691     = g_signal_connect_swapped (window->button_template[i],
02692                                "selection-changed",
02693                                (void (*)(void *)) window_template_experiment,
02694                                (void *) (size_t) i);
02695     gtk_grid_attach (window->grid_experiment,
02696                     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02697 }
02698 window->frame_experiment
02699     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02700 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02701                   GTK_WIDGET (window->grid_experiment));
02702 // Creating the error norm widgets
02703 window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02704 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02705 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02706                   GTK_WIDGET (window->grid_norm));
02707 window->button_norm[0] = (GtkRadioButton *)
02708     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02709 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02710                             tip_norm[0]);
02711 gtk_grid_attach (window->grid_norm,
02712                 GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02713 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02714 for (i = 0; ++i < NNORMS;)
02715 {
02716     window->button_norm[i] = (GtkRadioButton *)
02717     gtk_radio_button_new_with_mnemonic
02718     (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);

```

```

02719     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02720                                 tip_norm[i]);
02721     gtk_grid_attach (window->grid_norm,
02722                     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02723     g_signal_connect (window->button_norm[i], "clicked",
window_update, NULL);
02724 }
02725 window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02726 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02727 window->spin_p = (GtkSpinButton *)
02728     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02729 gtk_widget_set_tooltip_text
02730     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02731 window->scrolled_p
02732     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02733 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02734                   GTK_WIDGET (window->spin_p));
02735 gtk_widget_set_expand (GTK_WIDGET (window->scrolled_p), TRUE);
02736 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02737 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02738                 1, 2, 1, 2);
02739
02740 // Creating the grid and attaching the widgets to the grid
02741 window->grid = (GtkGrid *) gtk_grid_new ();
02742 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02743 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02744 gtk_grid_attach (window->grid,
02745                 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02746 gtk_grid_attach (window->grid,
02747                 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02748 gtk_grid_attach (window->grid,
02749                 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02750 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02751 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));
02752
02753 // Setting the window logo
02754 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02755 gtk_window_set_icon (window->window, window->logo);
02756
02757 // Showing the window
02758 gtk_widget_show_all (GTK_WIDGET (window->window));
02759
02760 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02761 #if GTK_MINOR_VERSION >= 16
02762     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02763     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02764     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02765     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02766     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02767     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02768     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02769 #endif
02770
02771 // Reading initial example
02772 input_new ();
02773 buffer2 = g_get_current_dir ();
02774 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02775 g_free (buffer2);
02776 window_read (buffer);
02777 g_free (buffer);
02778
02779 #if DEBUG_INTERFACE
02780     fprintf (stderr, "window_new: start\n");
02781 #endif
02782 }

```

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 GtkWidget * [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

Copyright 2012-2016, all rights reserved.

Definition in file [interface.h](#).

5.13.2 Function Documentation

5.13.2.1 gtk_array_get_active()

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

Function to get the active GtkRadioButton.

Parameters

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line [565](#) of file [utils.c](#).

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

5.13.2.2 input_save()

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

Function to save the input file.

Parameters

<i>filename</i>	Input file name.
-----------------	------------------

Definition at line 575 of file [interface.c](#).

```

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

```

Here is the call graph for this function:

5.13.2.3 window_get_algorithm()

```
unsigned int window_get_algorithm ( )
```

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 726 of file [interface.c](#).

```

00727 {
00728     unsigned int i;
00729     #if DEBUG_INTERFACE
00730         fprintf (stderr, "window_get_algorithm: start\n");
00731     #endif
00732     i = gtk_array_get_active (window->button_algorithm,
00733                             NALGORITHMS);
00734     #if DEBUG_INTERFACE
00735         fprintf (stderr, "window_get_algorithm: %u\n", i);
00736         fprintf (stderr, "window_get_algorithm: end\n");
00737     #endif
00738     return i;
00739 }

```

Here is the call graph for this function:

5.13.2.4 window_get_direction()

```
unsigned int window_get_direction ( )
```

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 746 of file [interface.c](#).

```
00747 {
00748     unsigned int i;
00749     #if DEBUG_INTERFACE
00750     fprintf (stderr, "window_get_direction: start\n");
00751     #endif
00752     i = gtk_array_get_active (window->button_direction,
00753                             NDIRECTIONS);
00754     #if DEBUG_INTERFACE
00755     fprintf (stderr, "window_get_direction: %u\n", i);
00756     fprintf (stderr, "window_get_direction: end\n");
00757     #endif
00758     return i;
00759 }
```

Here is the call graph for this function:

5.13.2.5 window_get_norm()

```
unsigned int window_get_norm ( )
```

Function to get the norm method number.

Returns

Norm method number.

Definition at line 766 of file [interface.c](#).

```
00767 {
00768     unsigned int i;
00769     #if DEBUG_INTERFACE
00770     fprintf (stderr, "window_get_norm: start\n");
00771     #endif
00772     i = gtk_array_get_active (window->button_norm,
00773                             NNORMS);
00774     #if DEBUG_INTERFACE
00775     fprintf (stderr, "window_get_norm: %u\n", i);
00776     fprintf (stderr, "window_get_norm: end\n");
00777     #endif
00778     return i;
00779 }
```

Here is the call graph for this function:

5.13.2.6 window_read()

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

Function to read the input data of a file.

Parameters

<i>filename</i>	File name.
-----------------	------------

Returns

1 on succes, 0 on error.

Definition at line 1874 of file [interface.c](#).

```

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

```

```

01937     }
01938     break;
01939     default:
01940         gtk_spin_button_set_value (window->spin_population,
01941                                   (gdouble) input->nsimulations);
01942         gtk_spin_button_set_value (window->spin_generations,
01943                                   (gdouble) input->niterations);
01944         gtk_spin_button_set_value (window->spin_mutation, input->
01945                                   mutation_ratio);
01946         gtk_spin_button_set_value (window->spin_reproduction,
01947                                   input->reproduction_ratio);
01948         gtk_spin_button_set_value (window->spin_adaptation,
01949                                   input->adaptation_ratio);
01950     }
01951     gtk_toggle_button_set_active
01952     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01953     gtk_spin_button_set_value (window->spin_p, input->p);
01954     gtk_spin_button_set_value (window->spin_threshold, input->
01955                               threshold);
01956     g_signal_handler_block (window->combo_experiment, window->
01957                             id_experiment);
01958     g_signal_handler_block (window->button_experiment,
01959                             window->id_experiment_name);
01960     gtk_combo_box_text_remove_all (window->combo_experiment);
01961     for (i = 0; i < input->nexperiments; ++i)
01962         gtk_combo_box_text_append_text (window->combo_experiment,
01963                                         input->experiment[i].name);
01964     g_signal_handler_unblock
01965     (window->button_experiment, window->
01966         id_experiment_name);
01967     g_signal_handler_unblock (window->combo_experiment,
01968                               window->id_experiment);
01969     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01970     g_signal_handler_block (window->combo_variable, window->
01971                             id_variable);
01972     g_signal_handler_block (window->entry_variable, window->
01973                             id_variable_label);
01974     gtk_combo_box_text_remove_all (window->combo_variable);
01975     for (i = 0; i < input->nvariables; ++i)
01976         gtk_combo_box_text_append_text (window->combo_variable,
01977                                         input->variable[i].name);
01978     g_signal_handler_unblock (window->entry_variable, window->
01979                             id_variable_label);
01980     g_signal_handler_unblock (window->combo_variable, window->
01981                             id_variable);
01982     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01983     window_set_variable ();
01984     window_update ();
01985 }
01986
01987 #if DEBUG_INTERFACE
01988 fprintf (stderr, "window_read: end\n");
01989 #endif
01990 return 1;
01991 }

```

Here is the call graph for this function:

5.13.2.7 window_save()

```
int window_save ( )
```

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 819 of file [interface.c](#).

```

00820 {
00821     GtkFileChooserDialog *dlg;
00822     GtkFileFilter *filter1, *filter2;
00823     char *buffer;
00824
00825     #if DEBUG_INTERFACE
00826         fprintf (stderr, "window_save: start\n");
00827     #endif
00828
00829     // Opening the saving dialog
00830     dlg = (GtkFileChooserDialog *)
00831         gtk_file_chooser_dialog_new (gettext ("Save file"),
00832                                     window->window,
00833                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00834                                     gettext ("Cancel"),
00835                                     GTK_RESPONSE_CANCEL,
00836                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
00837     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00838     buffer = g_build_filename (input->directory, input->name, NULL);
00839     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00840     g_free (buffer);
00841
00842     // Adding XML filter
00843     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00844     gtk_file_filter_set_name (filter1, "XML");
00845     gtk_file_filter_add_pattern (filter1, "*.xml");
00846     gtk_file_filter_add_pattern (filter1, "*.XML");
00847     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00848
00849     // Adding JSON filter
00850     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter2, "JSON");
00852     gtk_file_filter_add_pattern (filter2, "*.json");
00853     gtk_file_filter_add_pattern (filter2, "*.JSON");
00854     gtk_file_filter_add_pattern (filter2, "*.js");
00855     gtk_file_filter_add_pattern (filter2, "*.JS");
00856     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00857
00858     if (input->type == INPUT_TYPE_XML)
00859         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00860     else
00861         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00862
00863     // If OK response then saving
00864     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00865     {
00866         // Setting input file type
00867         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00868         buffer = (char *) gtk_file_filter_get_name (filter1);
00869         if (!strcmp (buffer, "XML"))
00870             input->type = INPUT_TYPE_XML;
00871         else
00872             input->type = INPUT_TYPE_JSON;
00873
00874         // Adding properties to the root XML node
00875         input->simulator = gtk_file_chooser_get_filename
00876             (GTK_FILE_CHOOSER (window->button_simulator));
00877         if (gtk_toggle_button_get_active
00878             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00879             input->evaluator = gtk_file_chooser_get_filename
00880                 (GTK_FILE_CHOOSER (window->button_evaluator));
00881         else
00882             input->evaluator = NULL;
00883         if (input->type == INPUT_TYPE_XML)
00884         {
00885             input->result
00886                 = (char *) xmlStrdup ((const xmlChar *)
00887                                         gtk_entry_get_text (window->entry_result));
00888             input->variables
00889                 = (char *) xmlStrdup ((const xmlChar *)
00890                                         gtk_entry_get_text (window->
00891 entry_variables));
00892         }
00893         else
00894         {
00895             input->result = g_strdup (gtk_entry_get_text (window->
00896 entry_result));
00897             input->variables
00898                 = g_strdup (gtk_entry_get_text (window->entry_variables));
00899         }
00900
00901         // Setting the algorithm
00902         switch (window_get_algorithm ())
00903         {
00904             case ALGORITHM_MONTE_CARLO:
00905                 input->algorithm = ALGORITHM_MONTE_CARLO;
00906                 input->nsimulations

```



```

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

```

Here is the call graph for this function:

5.13.2.8 window_template_experiment()

```

void window_template_experiment (
    void * data )

```

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

Parameters

<i>data</i>	Callback data (i-th input template).
-------------	--------------------------------------

Definition at line 1518 of file [interface.c](#).

```

01519 {
01520     unsigned int i, j;
01521     char *buffer;
01522     GFile *file1, *file2;
01523     #if DEBUG_INTERFACE
01524     fprintf (stderr, "window_template_experiment: start\n");
01525     #endif
01526     i = (size_t) data;
01527     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01528     file1
01529     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01530     file2 = g_file_new_for_path (input->directory);
01531     buffer = g_file_get_relative_path (file2, file1);
01532     if (input->type == INPUT_TYPE_XML)
01533         input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01534     else
01535         input->experiment[j].template[i] = g_strdup (buffer);
01536     g_free (buffer);
01537     g_object_unref (file2);
01538     g_object_unref (file1);
01539     #if DEBUG_INTERFACE
01540     fprintf (stderr, "window_template_experiment: end\n");
01541     #endif
01542 }

```

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
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkDialog *dialog;
00040     GtkGrid *grid;
00041     GtkLabel *label_seed;
00042     GtkSpinButton *spin_seed;
00043     GtkLabel *label_threads;
00044     GtkSpinButton *spin_threads;
00045     GtkLabel *label_direction;
00046     GtkSpinButton *spin_direction;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkDialog *dialog;
00052     GtkLabel *label;
00053     GtkSpinner *spinner;
00054     GtkGrid *grid;
00055 } Running;

```

```
00074
00079 typedef struct
00080 {
00081     GtkWidget *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   GtkWidget *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 n_experiments;
00213   unsigned int n_variables;
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 GtkWidget *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227     GtkWidget *button;
00228     GtkWidget *image;
00229     button = (GtkWidget *) gtk_button_new ();
00230     image = (GtkWidget *) 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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #endif
00054 #include "genetic/genetic.h"
00055 #include "utils.h"
00056 #include "experiment.h"
00057 #include "variable.h"
00058 #include "input.h"
00059 #include "optimize.h"
00060 #if HAVE_GTK
00061 #include "interface.h"
00062 #endif
00063
00064 #define DEBUG_MAIN 0
00065
00066 int
00067 main (int argn, char **argc)
00068 {
00069     #if HAVE_GTK
00070     char *buffer;
00071     #endif
00072
00073     // Starting pseudo-random numbers generator
00074     #if DEBUG_MAIN
00075     fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00076     #endif
00077     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00078
00079     // Allowing spaces in the XML data file
00080     #if DEBUG_MAIN
00081     fprintf (stderr, "main: allowing spaces in the XML data file\n");
00082     #endif
00083     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,
00130 LOCALE_DIR, NULL);
00131 bindtextdomain (PROGRAM_INTERFACE, buffer);
00132 bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00133 textdomain (PROGRAM_INTERFACE);
00134
00135 // Initing GTK+
00136 gtk_disable_setlocale ();
00137 gtk_init (&argn, &argc);
00138
00139 // Opening the main window
00140 window_new ();
00141 gtk_main ();
00142
00143 // Freeing memory
00144 input_free ();
00145 g_free (buffer);
00146 gtk_widget_destroy (GTK_WIDGET (window->window));
00147 g_free (window->application_directory);
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 <sys/param.h>
#include <gsl/gsl_rng.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>

```



```
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include <alloca.h>
#include <mpi.h>
#include "genetic/genetic.h"
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"
#include "optimize.h"
```

Include dependency graph for optimize.c:

Macros

- `#define _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

Copyright 2012-2016, all rights reserved.

Definition in file [optimize.c](#).

5.17.2 Function Documentation

5.17.2.1 optimize_best()

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

Function to save the best simulations.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 463 of file [optimize.c](#).

```
00464 {
00465     unsigned int i, j;
00466     double e;
00467     #if DEBUG_OPTIMIZE
00468         fprintf (stderr, "optimize_best: start\n");
00469         fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00470                 optimize->nsaveds, optimize->nbest);
00471     #endif
00472     if (optimize->nsaveds < optimize->nbest
00473         || value < optimize->error_best[optimize->nsaveds - 1])
00474     {
00475         if (optimize->nsaveds < optimize->nbest)
00476             ++optimize->nsaveds;
00477         optimize->error_best[optimize->nsaveds - 1] = value;
00478         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00479         for (i = optimize->nsaveds; --i;)
00480         {
00481             if (optimize->error_best[i] < optimize->
00482                 error_best[i - 1])
00483             {
00484                 j = optimize->simulation_best[i];
00485                 e = optimize->error_best[i];
00486                 optimize->simulation_best[i] = optimize->
00487                     simulation_best[i - 1];
00488                 optimize->error_best[i] = optimize->
00489                     error_best[i - 1];
00490                 optimize->simulation_best[i - 1] = j;
00491                 optimize->error_best[i - 1] = e;
00492             }
00493         }
00494     }
```

```

00490         else
00491             break;
00492     }
00493 }
00494 #if DEBUG_OPTIMIZE
00495 fprintf (stderr, "optimize_best: end\n");
00496 #endif
00497 }

```

5.17.2.2 optimize_best_direction()

```

void optimize_best_direction (
    unsigned int simulation,
    double value )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 788 of file [optimize.c](#).

```

00789 {
00790 #if DEBUG_OPTIMIZE
00791     fprintf (stderr, "optimize_best_direction: start\n");
00792     fprintf (stderr,
00793         "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00794         simulation, value, optimize->error_best[0]);
00795 #endif
00796     if (value < optimize->error_best[0])
00797     {
00798         optimize->error_best[0] = value;
00799         optimize->simulation_best[0] = simulation;
00800 #if DEBUG_OPTIMIZE
00801         fprintf (stderr,
00802             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00803             simulation, value);
00804 #endif
00805     }
00806 #if DEBUG_OPTIMIZE
00807     fprintf (stderr, "optimize_best_direction: end\n");
00808 #endif
00809 }

```

5.17.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 818 of file [optimize.c](#).

```

00819 {
00820     unsigned int i, j;
00821     double e;
00822     #if DEBUG_OPTIMIZE
00823         fprintf (stderr, "optimize_direction_sequential: start\n");
00824         fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00825                 "nend_direction=%u\n",
00826                 optimize->nstart_direction, optimize->
nend_direction);
00827     #endif
00828     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00829     {
00830         j = simulation + i;
00831         e = optimize_norm (j);
00832         optimize_best_direction (j, e);
00833         optimize_save_variables (j, e);
00834         if (e < optimize->threshold)
00835         {
00836             optimize->stop = 1;
00837             break;
00838         }
00839     #if DEBUG_OPTIMIZE
00840         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00841     #endif
00842     }
00843     #if DEBUG_OPTIMIZE
00844         fprintf (stderr, "optimize_direction_sequential: end\n");
00845     #endif
00846 }

```

Here is the call graph for this function:

5.17.2.4 optimize_direction_thread()

```

void * optimize_direction_thread (
    ParallelData * data )

```

Function to estimate the direction search on a thread.

Parameters

<i>data</i>	Function data.
-------------	----------------

Returns

NULL

Definition at line 856 of file [optimize.c](#).

```

00857 {
00858     unsigned int i, thread;
00859     double e;
00860     #if DEBUG_OPTIMIZE
00861         fprintf (stderr, "optimize_direction_thread: start\n");
00862     #endif
00863     thread = data->thread;
00864     #if DEBUG_OPTIMIZE
00865         fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00866                 thread,
00867                 optimize->thread_direction[thread],
00868                 optimize->thread_direction[thread + 1]);
00869     #endif
00870     for (i = optimize->thread_direction[thread];
00871          i < optimize->thread_direction[thread + 1]; ++i)
00872     {
00873         e = optimize_norm (i);
00874         g_mutex_lock (mutex);
00875         optimize_best_direction (i, e);
00876         optimize_save_variables (i, e);

```

```

00877     if (e < optimize->threshold)
00878         optimize->stop = 1;
00879     g_mutex_unlock (mutex);
00880     if (optimize->stop)
00881         break;
00882 #if DEBUG_OPTIMIZE
00883     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00884 #endif
00885 }
00886 #if DEBUG_OPTIMIZE
00887     fprintf (stderr, "optimize_direction_thread: end\n");
00888 #endif
00889     g_thread_exit (NULL);
00890     return NULL;
00891 }

```

Here is the call graph for this function:

5.17.2.5 optimize_estimate_direction_coordinates()

```

double optimize_estimate_direction_coordinates (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the direction search vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 930 of file [optimize.c](#).

```

00932 {
00933     double x;
00934 #if DEBUG_OPTIMIZE
00935     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00936 #endif
00937     x = optimize->direction[variable];
00938     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00939     {
00940         if (estimate & 1)
00941             x += optimize->step[variable];
00942         else
00943             x -= optimize->step[variable];
00944     }
00945 #if DEBUG_OPTIMIZE
00946     fprintf (stderr,
00947             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00948             variable, x);
00949     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00950 #endif
00951     return x;
00952 }

```

5.17.2.6 optimize_estimate_direction_random()

```

double optimize_estimate_direction_random (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the direction search vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 903 of file [optimize.c](#).

```

00905 {
00906     double x;
00907     #if DEBUG_OPTIMIZE
00908     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00909     #endif
00910     x = optimize->direction[variable]
00911         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00912         step[variable];
00913     #if DEBUG_OPTIMIZE
00914     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00915             variable, x);
00916     #endif
00917     return x;
00918 }
```

5.17.2.7 optimize_genetic_objective()

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

Function to calculate the objective function of an entity.

Parameters

<i>entity</i>	entity data.
---------------	--------------

Returns

objective function value.

Definition at line 1097 of file [optimize.c](#).

```

01098 {
01099     unsigned int j;
01100     double objective;
01101     char buffer[64];
01102     #if DEBUG_OPTIMIZE
01103     fprintf (stderr, "optimize_genetic_objective: start\n");
01104     #endif
01105     for (j = 0; j < optimize->nvariables; ++j)
01106     {
01107         optimize->value[entity->id * optimize->nvariables + j]
01108             = genetic_get_variable (entity, optimize->genetic_variable + j);
01109     }
01110     objective = optimize_norm (entity->id);
01111     g_mutex_lock (mutex);
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01115         fprintf (optimize->file_variables, buffer,
01116                 genetic_get_variable (entity, optimize->genetic_variable + j));
01117     }
01118     fprintf (optimize->file_variables, "%.14le\n", objective);
01119     g_mutex_unlock (mutex);
01120     #if DEBUG_OPTIMIZE
01121     fprintf (stderr, "optimize_genetic_objective: end\n");
01122     #endif
01123     return objective;
01124 }
```

5.17.2.8 optimize_input()

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

Function to write the simulation input file.

Parameters

<i>simulation</i>	Simulation number.
<i>input</i>	Input file name.
<i>template</i>	Template of the input file name.

Definition at line 104 of file [optimize.c](#).

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



```

00162                                     0, NULL);
00163     g_free (buffer2);
00164     g_regex_unref (regex);
00165 }
00166
00167 // Saving input file
00168 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00169 g_free (buffer3);
00170 fclose (file);
00171
00172 optimize_input_end:
00173 #if DEBUG_OPTIMIZE
00174     fprintf (stderr, "optimize_input: end\n");
00175 #endif
00176     return;
00177 }

```

5.17.2.9 optimize_merge()

```

void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best )

```

Function to merge the 2 optimization results.

Parameters

<i>nsaveds</i>	Number of saved results.
<i>simulation_best</i>	Array of best simulation numbers.
<i>error_best</i>	Array of best objective function values.

Definition at line 586 of file [optimize.c](#).

```

00588 {
00589     unsigned int i, j, k, s[optimize->nbest];
00590     double e[optimize->nbest];
00591     #if DEBUG_OPTIMIZE
00592     fprintf (stderr, "optimize_merge: start\n");
00593     #endif
00594     i = j = k = 0;
00595     do
00596     {
00597         if (i == optimize->nsaveds)
00598         {
00599             s[k] = simulation_best[j];
00600             e[k] = error_best[j];
00601             ++j;
00602             ++k;
00603             if (j == nsaveds)
00604                 break;
00605         }
00606         else if (j == nsaveds)
00607         {
00608             s[k] = optimize->simulation_best[i];
00609             e[k] = optimize->error_best[i];
00610             ++i;
00611             ++k;
00612             if (i == optimize->nsaveds)
00613                 break;
00614         }
00615         else if (optimize->error_best[i] > error_best[j])
00616         {
00617             s[k] = simulation_best[j];
00618             e[k] = error_best[j];
00619             ++j;
00620             ++k;
00621         }
00622         else
00623         {

```

```

00624         s[k] = optimize->simulation_best[i];
00625         e[k] = optimize->error_best[i];
00626         ++i;
00627         ++k;
00628     }
00629 }
00630 while (k < optimize->nbest);
00631 optimize->nsaveds = k;
00632 memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00633 memcpy (optimize->error_best, e, k * sizeof (double));
00634 #if DEBUG_OPTIMIZE
00635     fprintf (stderr, "optimize_merge: end\n");
00636 #endif
00637 }

```

5.17.2.10 optimize_norm_euclidian()

```

double optimize_norm_euclidian (
    unsigned int simulation )

```

Function to calculate the Euclidian error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Euclidian error norm.

Definition at line 296 of file [optimize.c](#).

```

00297 {
00298     double e, ei;
00299     unsigned int i;
00300     #if DEBUG_OPTIMIZE
00301         fprintf (stderr, "optimize_norm_euclidian: start\n");
00302     #endif
00303     e = 0.;
00304     for (i = 0; i < optimize->nexperiments; ++i)
00305     {
00306         ei = optimize_parse (simulation, i);
00307         e += ei * ei;
00308     }
00309     e = sqrt (e);
00310     #if DEBUG_OPTIMIZE
00311         fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00312         fprintf (stderr, "optimize_norm_euclidian: end\n");
00313     #endif
00314     return e;
00315 }

```

Here is the call graph for this function:

5.17.2.11 optimize_norm_maximum()

```

double optimize_norm_maximum (
    unsigned int simulation )

```

Function to calculate the maximum error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Maximum error norm.

Definition at line 325 of file [optimize.c](#).

```

00326 {
00327     double e, ei;
00328     unsigned int i;
00329     #if DEBUG_OPTIMIZE
00330     fprintf (stderr, "optimize_norm_maximum: start\n");
00331     #endif
00332     e = 0.;
00333     for (i = 0; i < optimize->nexperiments; ++i)
00334     {
00335         ei = fabs (optimize_parse (simulation, i));
00336         e = fmax (e, ei);
00337     }
00338     #if DEBUG_OPTIMIZE
00339     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00340     fprintf (stderr, "optimize_norm_maximum: end\n");
00341     #endif
00342     return e;
00343 }
```

Here is the call graph for this function:

5.17.2.12 optimize_norm_p()

```

double optimize_norm_p (
    unsigned int simulation )
```

Function to calculate the P error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

P error norm.

Definition at line 353 of file [optimize.c](#).

```

00354 {
00355     double e, ei;
00356     unsigned int i;
00357     #if DEBUG_OPTIMIZE
00358     fprintf (stderr, "optimize_norm_p: start\n");
00359     #endif
00360     e = 0.;
00361     for (i = 0; i < optimize->nexperiments; ++i)
00362     {
00363         ei = fabs (optimize_parse (simulation, i));
00364         e += pow (ei, optimize->p);
00365     }
```

```

00366     e = pow (e, 1. / optimize->p);
00367     #if DEBUG_OPTIMIZE
00368     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00369     fprintf (stderr, "optimize_norm_p: end\n");
00370     #endif
00371     return e;
00372 }

```

Here is the call graph for this function:

5.17.2.13 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )

```

Function to calculate the taxicab error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Taxicab error norm.

Definition at line 382 of file [optimize.c](#).

```

00383 {
00384     double e;
00385     unsigned int i;
00386     #if DEBUG_OPTIMIZE
00387     fprintf (stderr, "optimize_norm_taxicab: start\n");
00388     #endif
00389     e = 0.;
00390     for (i = 0; i < optimize->nexperiments; ++i)
00391         e += fabs (optimize_parse (simulation, i));
00392     #if DEBUG_OPTIMIZE
00393     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00394     fprintf (stderr, "optimize_norm_taxicab: end\n");
00395     #endif
00396     return e;
00397 }

```

Here is the call graph for this function:

5.17.2.14 optimize_parse()

```

double optimize_parse (
    unsigned int simulation,
    unsigned int experiment )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>experiment</i>	Experiment number.

Returns

Objective function value.

Definition at line 190 of file [optimize.c](#).

```

00191 {
00192     unsigned int i;
00193     double e;
00194     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00195         *buffer3, *buffer4;
00196     FILE *file_result;
00197
00198     #if DEBUG_OPTIMIZE
00199         fprintf (stderr, "optimize_parse: start\n");
00200         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00201             experiment);
00202     #endif
00203
00204     // Opening input files
00205     for (i = 0; i < optimize->ninputs; ++i)
00206     {
00207         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00208     #if DEBUG_OPTIMIZE
00209         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00210     #endif
00211         optimize_input (simulation, &input[i][0], optimize->
00212             file[i][experiment]);
00212     }
00213     for (; i < MAX_NINPUTS; ++i)
00214         strcpy (&input[i][0], "");
00215     #if DEBUG_OPTIMIZE
00216         fprintf (stderr, "optimize_parse: parsing end\n");
00217     #endif
00218
00219     // Performing the simulation
00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225         buffer, input[0], input[1], input[2], input[3], input[4], input[5],
00226         input[6], input[7], output);
00227     g_free (buffer4);
00228     g_free (buffer3);
00229     g_free (buffer2);
00230     #if DEBUG_OPTIMIZE
00231         fprintf (stderr, "optimize_parse: %s\n", buffer);
00232     #endif
00233     system (buffer);
00234
00235     // Checking the objective value function
00236     if (optimize->evaluator)
00237     {
00238         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00239         buffer2 = g_path_get_dirname (optimize->evaluator);
00240         buffer3 = g_path_get_basename (optimize->evaluator);
00241         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00242         snprintf (buffer, 512, "\"%s\" %s %s %s",
00243             buffer4, output, optimize->experiment[experiment], result);
00244         g_free (buffer4);
00245         g_free (buffer3);
00246         g_free (buffer2);
00247     #if DEBUG_OPTIMIZE
00248         fprintf (stderr, "optimize_parse: %s\n", buffer);
00249     #endif
00250         system (buffer);
00251         file_result = g_fopen (result, "r");
00252         e = atof (fgets (buffer, 512, file_result));
00253         fclose (file_result);
00254     }
00255     else
00256     {
00257         strcpy (result, "");
00258         file_result = g_fopen (output, "r");
00259         e = atof (fgets (buffer, 512, file_result));
00260         fclose (file_result);
00261     }
00262
00263     // Removing files
00264     #if !DEBUG_OPTIMIZE
00265     for (i = 0; i < optimize->ninputs; ++i)
00266     {
00267         if (optimize->file[i][0])

```

```

00268     {
00269         snprintf (buffer, 512, RM " %s", &input[i][0]);
00270         system (buffer);
00271     }
00272 }
00273 snprintf (buffer, 512, RM " %s %s", output, result);
00274 system (buffer);
00275 #endif
00276
00277 // Processing pending events
00278 show_pending ();
00279
00280 #if DEBUG_OPTIMIZE
00281 fprintf (stderr, "optimize_parse: end\n");
00282 #endif
00283
00284 // Returning the objective function
00285 return e * optimize->weight[experiment];
00286 }

```

Here is the call graph for this function:

5.17.2.15 optimize_save_variables()

```

void optimize_save_variables (
    unsigned int simulation,
    double error )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>error</i>	Error value.

Definition at line 435 of file [optimize.c](#).

```

00436 {
00437     unsigned int i;
00438     char buffer[64];
00439     #if DEBUG_OPTIMIZE
00440     fprintf (stderr, "optimize_save_variables: start\n");
00441     #endif
00442     for (i = 0; i < optimize->nvariables; ++i)
00443     {
00444         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00445         fprintf (optimize->file_variables, buffer,
00446             optimize->value[simulation * optimize->
00447                 nvariables + i]);
00448     }
00449     fprintf (optimize->file_variables, "%.14le\n", error);
00450     #if DEBUG_OPTIMIZE
00451     fprintf (stderr, "optimize_save_variables: end\n");
00452     #endif
00453 }

```

5.17.2.16 optimize_step_direction()

```

void optimize_step_direction (
    unsigned int simulation )

```

Function to do a step of the direction search method.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 961 of file `optimize.c`.

```

00962 {
00963     GThread *thread[nthreads_direction];
00964     ParallelData data[nthreads_direction];
00965     unsigned int i, j, k, b;
00966     #if DEBUG_OPTIMIZE
00967     fprintf (stderr, "optimize_step_direction: start\n");
00968     #endif
00969     for (i = 0; i < optimize->nestimates; ++i)
00970     {
00971         k = (simulation + i) * optimize->nvariables;
00972         b = optimize->simulation_best[0] * optimize->
nvariables;
00973         #if DEBUG_OPTIMIZE
00974         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00975                 simulation + i, optimize->simulation_best[0]);
00976         #endif
00977         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00978         {
00979             #if DEBUG_OPTIMIZE
00980             fprintf (stderr,
00981                     "optimize_step_direction: estimate=%u best%u=%.14le\n",
00982                     i, j, optimize->value[b]);
00983             #endif
00984             optimize->value[k]
00985             = optimize->value[b] + optimize_estimate_direction (j,
i);
00986             optimize->value[k] = fmin (fmax (optimize->value[k],
00987                                             optimize->rangeminabs[j]),
00988                                       optimize->rangemaxabs[j]);
00989             #if DEBUG_OPTIMIZE
00990             fprintf (stderr,
00991                     "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00992                     i, j, optimize->value[k]);
00993             #endif
00994         }
00995     }
00996     if (nthreads_direction == 1)
00997         optimize_direction_sequential (simulation);
00998     else
00999     {
01000         for (i = 0; i <= nthreads_direction; ++i)
01001         {
01002             optimize->thread_direction[i]
01003             = simulation + optimize->nstart_direction
01004             + i * (optimize->nend_direction - optimize->
nstart_direction)
01005             / nthreads_direction;
01006             #if DEBUG_OPTIMIZE
01007             fprintf (stderr,
01008                     "optimize_step_direction: i=%u thread_direction=%u\n",
01009                     i, optimize->thread_direction[i]);
01010             #endif
01011         }
01012         for (i = 0; i < nthreads_direction; ++i)
01013         {
01014             data[i].thread = i;
01015             thread[i] = g_thread_new
01016             (NULL, (void (*)(void *)) optimize_direction_thread, &data[i]);
01017         }
01018         for (i = 0; i < nthreads_direction; ++i)
01019             g_thread_join (thread[i]);
01020     }
01021     #if DEBUG_OPTIMIZE
01022     fprintf (stderr, "optimize_step_direction: end\n");
01023     #endif
01024 }

```

Here is the call graph for this function:

5.17.2.17 optimize_thread()

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

Function to optimize on a thread.

Parameters

<i>data</i>	Function data.
-------------	----------------

Returns

NULL

Definition at line 540 of file [optimize.c](#).

```

00541 {
00542     unsigned int i, thread;
00543     double e;
00544     #if DEBUG_OPTIMIZE
00545     fprintf (stderr, "optimize_thread: start\n");
00546     #endif
00547     thread = data->thread;
00548     #if DEBUG_OPTIMIZE
00549     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00550             optimize->thread[thread], optimize->thread[thread + 1]);
00551     #endif
00552     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00553     {
00554         e = optimize_norm (i);
00555         g_mutex_lock (mutex);
00556         optimize_best (i, e);
00557         optimize_save_variables (i, e);
00558         if (e < optimize->threshold)
00559             optimize->stop = 1;
00560         g_mutex_unlock (mutex);
00561         if (optimize->stop)
00562             break;
00563     #if DEBUG_OPTIMIZE
00564     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00565     #endif
00566     }
00567     #if DEBUG_OPTIMIZE
00568     fprintf (stderr, "optimize_thread: end\n");
00569     #endif
00570     g_thread_exit (NULL);
00571     return NULL;
00572 }
```

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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <sys/param.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <glib/gstdio.h>
00044 #include <json-glib/json-glib.h>
00045 #ifdef G_OS_WIN32
00046 #include <windows.h>
00047 #elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
00048 #include <alloca.h>
00049 #endif
00050 #if HAVE_MPI
00051 #include <mpi.h>
00052 #endif
00053 #include "genetic/genetic.h"
00054 #include "utils.h"
00055 #include "experiment.h"
00056 #include "variable.h"
00057 #include "input.h"
00058 #include "optimize.h"
00059
00060 #define DEBUG_OPTIMIZE 0
00061
00062 #ifdef G_OS_WIN32
00063 #define RM "del"
00064 #else
00065 #define RM "rm"
00066 #endif
00067
00068 int ntasks;
00069 unsigned int nthreads;
00070 unsigned int nthreads_direction;
00071 GMutex mutex[1];
00072 void (*optimize_algorithm) ();
00073 double (*optimize_estimate_direction) (unsigned int variable,
00074                                       unsigned int estimate);
00075 double (*optimize_norm) (unsigned int simulation);
00076 Optimize optimize[1];
00077
00078 void
00079 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00080 {
00081     unsigned int i;
00082     char buffer[32], value[32], *buffer2, *buffer3, *content;
00083     FILE *file;
00084     gsize length;
00085     GRegex *regex;
00086
00087     #if DEBUG_OPTIMIZE
00088     fprintf (stderr, "optimize_input: start\n");
00089     #endif
00090
00091     // Checking the file
00092     if (!template)
00093         goto optimize_input_end;
00094
00095     // Opening template
00096     content = g_mapped_file_get_contents (template);
00097     length = g_mapped_file_get_length (template);
00098     #if DEBUG_OPTIMIZE
00099     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00100     #endif
00101     file = g_fopen (input, "w");
00102
00103     // Parsing template
00104     for (i = 0; i < optimize->nvariables; ++i)
00105     {
00106         #if DEBUG_OPTIMIZE
00107         fprintf (stderr, "optimize_input: variable=%u\n", i);
00108         #endif
00109         snprintf (buffer, 32, "@variable%u@", i + 1);
00110         regex = g_regex_new (buffer, 0, 0, NULL);

```

```

00136         if (i == 0)
00137         {
00138             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00139                                                optimize->label[i], 0, NULL);
00140 #if DEBUG_OPTIMIZE
00141             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00142 #endif
00143         }
00144         else
00145         {
00146             length = strlen (buffer3);
00147             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00148                                                optimize->label[i], 0, NULL);
00149             g_free (buffer3);
00150         }
00151         g_regex_unref (regex);
00152         length = strlen (buffer2);
00153         snprintf (buffer, 32, "@value%u@", i + 1);
00154         regex = g_regex_new (buffer, 0, 0, NULL);
00155         snprintf (value, 32, format[optimize->precision[i]],
00156                  optimize->value[simulation * optimize->nvariables + i]);
00157 #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
00167 // Saving input file
00168 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00169 g_free (buffer3);
00170 fclose (file);
00171
00172 optimize_input_end:
00173 #if DEBUG_OPTIMIZE
00174     fprintf (stderr, "optimize_input: end\n");
00175 #endif
00176     return;
00177 }
00178
00179 double
00180 optimize_parse (unsigned int simulation, unsigned int experiment)
00181 {
00182     unsigned int i;
00183     double e;
00184     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00185          *buffer3, *buffer4;
00186     FILE *file_result;
00187
00188 #if DEBUG_OPTIMIZE
00189     fprintf (stderr, "optimize_parse: start\n");
00190     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00191             experiment);
00192 #endif
00193
00194     // Opening input files
00195     for (i = 0; i < optimize->ninputs; ++i)
00196     {
00197         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00198 #if DEBUG_OPTIMIZE
00199         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00200 #endif
00201         optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00202     }
00203     for (; i < MAX_NINPUTS; ++i)
00204         strcpy (&input[i][0], "");
00205 #if DEBUG_OPTIMIZE
00206     fprintf (stderr, "optimize_parse: parsing end\n");
00207 #endif
00208
00209     // Performing the simulation
00210     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00211     buffer2 = g_path_get_dirname (optimize->simulator);
00212     buffer3 = g_path_get_basename (optimize->simulator);
00213     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00214     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00215             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00216             input[6], input[7], output);
00217     g_free (buffer4);
00218     g_free (buffer3);
00219     g_free (buffer2);
00220 #if DEBUG_OPTIMIZE
00221     fprintf (stderr, "optimize_parse: %s\n", buffer);
00222 #endif

```

```

00233     system (buffer);
00234
00235     // Checking the objective value function
00236     if (optimize->evaluator)
00237     {
00238         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00239         buffer2 = g_path_get_dirname (optimize->evaluator);
00240         buffer3 = g_path_get_basename (optimize->evaluator);
00241         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00242         snprintf (buffer, 512, "\"%s\" %s %s %s",
00243                 buffer4, output, optimize->experiment[experiment], result);
00244         g_free (buffer4);
00245         g_free (buffer3);
00246         g_free (buffer2);
00247     #if DEBUG_OPTIMIZE
00248         fprintf (stderr, "optimize_parse: %s\n", buffer);
00249     #endif
00250         system (buffer);
00251         file_result = g_fopen (result, "r");
00252         e = atof (fgets (buffer, 512, file_result));
00253         fclose (file_result);
00254     }
00255     else
00256     {
00257         strcpy (result, "");
00258         file_result = g_fopen (output, "r");
00259         e = atof (fgets (buffer, 512, file_result));
00260         fclose (file_result);
00261     }
00262
00263     // Removing files
00264     #if !DEBUG_OPTIMIZE
00265     for (i = 0; i < optimize->ninputs; ++i)
00266     {
00267         if (optimize->file[i][0])
00268         {
00269             snprintf (buffer, 512, RM " %s", &input[i][0]);
00270             system (buffer);
00271         }
00272     }
00273     snprintf (buffer, 512, RM " %s %s", output, result);
00274     system (buffer);
00275     #endif
00276
00277     // Processing pending events
00278     show_pending ();
00279
00280     #if DEBUG_OPTIMIZE
00281     fprintf (stderr, "optimize_parse: end\n");
00282     #endif
00283
00284     // Returning the objective function
00285     return e * optimize->weight[experiment];
00286 }
00287
00295 double
00296 optimize_norm_euclidian (unsigned int simulation)
00297 {
00298     double e, ei;
00299     unsigned int i;
00300     #if DEBUG_OPTIMIZE
00301     fprintf (stderr, "optimize_norm_euclidian: start\n");
00302     #endif
00303     e = 0.;
00304     for (i = 0; i < optimize->nexperiments; ++i)
00305     {
00306         ei = optimize_parse (simulation, i);
00307         e += ei * ei;
00308     }
00309     e = sqrt (e);
00310     #if DEBUG_OPTIMIZE
00311     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00312     fprintf (stderr, "optimize_norm_euclidian: end\n");
00313     #endif
00314     return e;
00315 }
00316
00324 double
00325 optimize_norm_maximum (unsigned int simulation)
00326 {
00327     double e, ei;
00328     unsigned int i;
00329     #if DEBUG_OPTIMIZE
00330     fprintf (stderr, "optimize_norm_maximum: start\n");
00331     #endif
00332     e = 0.;
00333     for (i = 0; i < optimize->nexperiments; ++i)

```

```

00334     {
00335         ei = fabs (optimize_parse (simulation, i));
00336         e = fmax (e, ei);
00337     }
00338 #if DEBUG_OPTIMIZE
00339     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00340     fprintf (stderr, "optimize_norm_maximum: end\n");
00341 #endif
00342     return e;
00343 }
00344
00352 double
00353 optimize_norm_p (unsigned int simulation)
00354 {
00355     double e, ei;
00356     unsigned int i;
00357 #if DEBUG_OPTIMIZE
00358     fprintf (stderr, "optimize_norm_p: start\n");
00359 #endif
00360     e = 0.;
00361     for (i = 0; i < optimize->nexperiments; ++i)
00362     {
00363         ei = fabs (optimize_parse (simulation, i));
00364         e += pow (ei, optimize->p);
00365     }
00366     e = pow (e, 1. / optimize->p);
00367 #if DEBUG_OPTIMIZE
00368     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00369     fprintf (stderr, "optimize_norm_p: end\n");
00370 #endif
00371     return e;
00372 }
00373
00381 double
00382 optimize_norm_taxicab (unsigned int simulation)
00383 {
00384     double e;
00385     unsigned int i;
00386 #if DEBUG_OPTIMIZE
00387     fprintf (stderr, "optimize_norm_taxicab: start\n");
00388 #endif
00389     e = 0.;
00390     for (i = 0; i < optimize->nexperiments; ++i)
00391         e += fabs (optimize_parse (simulation, i));
00392 #if DEBUG_OPTIMIZE
00393     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00394     fprintf (stderr, "optimize_norm_taxicab: end\n");
00395 #endif
00396     return e;
00397 }
00398
00403 void
00404 optimize_print ()
00405 {
00406     unsigned int i;
00407     char buffer[512];
00408 #if HAVE_MPI
00409     if (optimize->mpi_rank)
00410         return;
00411 #endif
00412     printf ("%s\n", gettext ("Best result"));
00413     fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
00414     printf ("error = %.15le\n", optimize->error_old[0]);
00415     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00416     for (i = 0; i < optimize->nvariables; ++i)
00417     {
00418         snprintf (buffer, 512, "%s = %s\n",
00419                 optimize->label[i], format[optimize->precision[i]]);
00420         printf (buffer, optimize->value_old[i]);
00421         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00422     }
00423     fflush (optimize->file_result);
00424 }
00425
00434 void
00435 optimize_save_variables (unsigned int simulation, double error)
00436 {
00437     unsigned int i;
00438     char buffer[64];
00439 #if DEBUG_OPTIMIZE
00440     fprintf (stderr, "optimize_save_variables: start\n");
00441 #endif
00442     for (i = 0; i < optimize->nvariables; ++i)
00443     {
00444         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00445         fprintf (optimize->file_variables, buffer,

```

```

00446         optimize->value[simulation * optimize->nvariables + i]);
00447     }
00448     fprintf (optimize->file_variables, "%.14le\n", error);
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_save_variables: end\n");
00451     #endif
00452 }
00453
00462 void
00463 optimize_best (unsigned int simulation, double value)
00464 {
00465     unsigned int i, j;
00466     double e;
00467     #if DEBUG_OPTIMIZE
00468     fprintf (stderr, "optimize_best: start\n");
00469     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00470             optimize->nsaveds, optimize->nbest);
00471     #endif
00472     if (optimize->nsaveds < optimize->nbest
00473         || value < optimize->error_best[optimize->nsaveds - 1])
00474     {
00475         if (optimize->nsaveds < optimize->nbest)
00476             ++optimize->nsaveds;
00477         optimize->error_best[optimize->nsaveds - 1] = value;
00478         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00479         for (i = optimize->nsaveds; --i;)
00480         {
00481             if (optimize->error_best[i] < optimize->error_best[i - 1])
00482             {
00483                 j = optimize->simulation_best[i];
00484                 e = optimize->error_best[i];
00485                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00486                 optimize->error_best[i] = optimize->error_best[i - 1];
00487                 optimize->simulation_best[i - 1] = j;
00488                 optimize->error_best[i - 1] = e;
00489             }
00490             else
00491                 break;
00492         }
00493     }
00494     #if DEBUG_OPTIMIZE
00495     fprintf (stderr, "optimize_best: end\n");
00496     #endif
00497 }
00498
00503 void
00504 optimize_sequential ()
00505 {
00506     unsigned int i;
00507     double e;
00508     #if DEBUG_OPTIMIZE
00509     fprintf (stderr, "optimize_sequential: start\n");
00510     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00511             optimize->nstart, optimize->nend);
00512     #endif
00513     for (i = optimize->nstart; i < optimize->nend; ++i)
00514     {
00515         e = optimize_norm (i);
00516         optimize_best (i, e);
00517         optimize_save_variables (i, e);
00518         if (e < optimize->threshold)
00519         {
00520             optimize->stop = 1;
00521             break;
00522         }
00523     }
00524     #if DEBUG_OPTIMIZE
00525     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00526     #endif
00527 }
00528 #if DEBUG_OPTIMIZE
00529 fprintf (stderr, "optimize_sequential: end\n");
00530 #endif
00531 }
00532
00539 void *
00540 optimize_thread (ParallelData * data)
00541 {
00542     unsigned int i, thread;
00543     double e;
00544     #if DEBUG_OPTIMIZE
00545     fprintf (stderr, "optimize_thread: start\n");
00546     #endif
00547     thread = data->thread;
00548     #if DEBUG_OPTIMIZE
00549     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00550             optimize->thread[thread], optimize->thread[thread + 1]);

```

```

00551 #endif
00552     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00553     {
00554         e = optimize_norm (i);
00555         g_mutex_lock (mutex);
00556         optimize_best (i, e);
00557         optimize_save_variables (i, e);
00558         if (e < optimize->threshold)
00559             optimize->stop = 1;
00560         g_mutex_unlock (mutex);
00561         if (optimize->stop)
00562             break;
00563 #if DEBUG_OPTIMIZE
00564         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00565 #endif
00566     }
00567 #if DEBUG_OPTIMIZE
00568     fprintf (stderr, "optimize_thread: end\n");
00569 #endif
00570     g_thread_exit (NULL);
00571     return NULL;
00572 }
00573
00585 void
00586 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00587                 double *error_best)
00588 {
00589     unsigned int i, j, k, s[optimize->nbest];
00590     double e[optimize->nbest];
00591 #if DEBUG_OPTIMIZE
00592     fprintf (stderr, "optimize_merge: start\n");
00593 #endif
00594     i = j = k = 0;
00595     do
00596     {
00597         if (i == optimize->nsaveds)
00598         {
00599             s[k] = simulation_best[j];
00600             e[k] = error_best[j];
00601             ++j;
00602             ++k;
00603             if (j == nsaveds)
00604                 break;
00605         }
00606         else if (j == nsaveds)
00607         {
00608             s[k] = optimize->simulation_best[i];
00609             e[k] = optimize->error_best[i];
00610             ++i;
00611             ++k;
00612             if (i == optimize->nsaveds)
00613                 break;
00614         }
00615         else if (optimize->error_best[i] > error_best[j])
00616         {
00617             s[k] = simulation_best[j];
00618             e[k] = error_best[j];
00619             ++j;
00620             ++k;
00621         }
00622         else
00623         {
00624             s[k] = optimize->simulation_best[i];
00625             e[k] = optimize->error_best[i];
00626             ++i;
00627             ++k;
00628         }
00629     }
00630     while (k < optimize->nbest);
00631     optimize->nsaveds = k;
00632     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00633     memcpy (optimize->error_best, e, k * sizeof (double));
00634 #if DEBUG_OPTIMIZE
00635     fprintf (stderr, "optimize_merge: end\n");
00636 #endif
00637 }
00638
00643 #if HAVE_MPI
00644 void
00645 optimize_synchronise ()
00646 {
00647     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00648     double error_best[optimize->nbest];
00649     MPI_Status mpi_stat;
00650 #if DEBUG_OPTIMIZE
00651     fprintf (stderr, "optimize_synchronise: start\n");
00652 #endif

```

```

00653     if (optimize->mpi_rank == 0)
00654     {
00655         for (i = 1; i < ntasks; ++i)
00656         {
00657             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00658             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00659                     MPI_COMM_WORLD, &mpi_stat);
00660             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00661                     MPI_COMM_WORLD, &mpi_stat);
00662             optimize_merge (nsaveds, simulation_best, error_best);
00663             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00664             if (stop)
00665                 optimize->stop = 1;
00666         }
00667         for (i = 1; i < ntasks; ++i)
00668             MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00669     }
00670     else
00671     {
00672         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00673         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00674                 MPI_COMM_WORLD);
00675         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00676                 MPI_COMM_WORLD);
00677         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00678         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00679         if (stop)
00680             optimize->stop = 1;
00681     }
00682     #if DEBUG_OPTIMIZE
00683     fprintf (stderr, "optimize_synchronise: end\n");
00684     #endif
00685 }
00686 #endif
00687
00692 void
00693 optimize_sweep ()
00694 {
00695     unsigned int i, j, k, l;
00696     double e;
00697     GThread *thread[nthreads];
00698     ParallelData data[nthreads];
00699     #if DEBUG_OPTIMIZE
00700     fprintf (stderr, "optimize_sweep: start\n");
00701     #endif
00702     for (i = 0; i < optimize->nsimulations; ++i)
00703     {
00704         k = i;
00705         for (j = 0; j < optimize->nvariables; ++j)
00706         {
00707             l = k % optimize->nsweeps[j];
00708             k /= optimize->nsweeps[j];
00709             e = optimize->rangemin[j];
00710             if (optimize->nsweeps[j] > 1)
00711                 e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00712                     / (optimize->nsweeps[j] - 1);
00713             optimize->value[i * optimize->nvariables + j] = e;
00714         }
00715     }
00716     optimize->nsaveds = 0;
00717     if (nthreads <= 1)
00718         optimize_sequential ();
00719     else
00720     {
00721         for (i = 0; i < nthreads; ++i)
00722         {
00723             data[i].thread = i;
00724             thread[i] = g_thread_new (NULL, (void (*)(void *)) optimize_thread, &data[i]);
00725         }
00726         for (i = 0; i < nthreads; ++i)
00727             g_thread_join (thread[i]);
00728     }
00729     #if HAVE_MPI
00730     // Communicating tasks results
00731     optimize_synchronise ();
00732     #endif
00733     #if DEBUG_OPTIMIZE
00734     fprintf (stderr, "optimize_sweep: end\n");
00735     #endif
00736 }
00737
00742 void
00743 optimize_MonteCarlo ()
00744 {
00745     unsigned int i, j;
00746     GThread *thread[nthreads];
00747     ParallelData data[nthreads];

```



```

00748 #if DEBUG_OPTIMIZE
00749 fprintf (stderr, "optimize_MonteCarlo: start\n");
00750 #endif
00751 for (i = 0; i < optimize->nsimulations; ++i)
00752     for (j = 0; j < optimize->nvariables; ++j)
00753         optimize->value[i * optimize->nvariables + j]
00754             = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00755               * (optimize->rangemax[j] - optimize->rangemin[j]);
00756 optimize->nsaveds = 0;
00757 if (nthreads <= 1)
00758     optimize_sequential ();
00759 else
00760     {
00761         for (i = 0; i < nthreads; ++i)
00762             {
00763                 data[i].thread = i;
00764                 thread[i] = g_thread_new (NULL, (void (*) ) optimize_thread, &data[i]);
00765             }
00766         for (i = 0; i < nthreads; ++i)
00767             g_thread_join (thread[i]);
00768     }
00769 #if HAVE_MPI
00770     // Communicating tasks results
00771     optimize_synchronise ();
00772 #endif
00773 #if DEBUG_OPTIMIZE
00774 fprintf (stderr, "optimize_MonteCarlo: end\n");
00775 #endif
00776 }
00777
00787 void
00788 optimize_best_direction (unsigned int simulation, double value)
00789 {
00790     #if DEBUG_OPTIMIZE
00791         fprintf (stderr, "optimize_best_direction: start\n");
00792         fprintf (stderr,
00793             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00794             simulation, value, optimize->error_best[0]);
00795     #endif
00796     if (value < optimize->error_best[0])
00797     {
00798         optimize->error_best[0] = value;
00799         optimize->simulation_best[0] = simulation;
00800     }
00801     #if DEBUG_OPTIMIZE
00802         fprintf (stderr,
00803             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00804             simulation, value);
00805     #endif
00806     #if DEBUG_OPTIMIZE
00807         fprintf (stderr, "optimize_best_direction: end\n");
00808     #endif
00809 }
00810
00817 void
00818 optimize_direction_sequential (unsigned int simulation)
00819 {
00820     unsigned int i, j;
00821     double e;
00822     #if DEBUG_OPTIMIZE
00823         fprintf (stderr, "optimize_direction_sequential: start\n");
00824         fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00825             "nend_direction=%u\n",
00826             optimize->nstart_direction, optimize->nend_direction);
00827     #endif
00828     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00829     {
00830         j = simulation + i;
00831         e = optimize_norm (j);
00832         optimize_best_direction (j, e);
00833         optimize_save_variables (j, e);
00834         if (e < optimize->threshold)
00835         {
00836             optimize->stop = 1;
00837             break;
00838         }
00839     }
00840     #if DEBUG_OPTIMIZE
00841         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00842     #endif
00843     #if DEBUG_OPTIMIZE
00844         fprintf (stderr, "optimize_direction_sequential: end\n");
00845     #endif
00846 }
00847
00855 void *
00856 optimize_direction_thread (ParallelData * data)

```

```

00857 {
00858     unsigned int i, thread;
00859     double e;
00860     #if DEBUG_OPTIMIZE
00861     fprintf (stderr, "optimize_direction_thread: start\n");
00862     #endif
00863     thread = data->thread;
00864     #if DEBUG_OPTIMIZE
00865     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00866             thread,
00867             optimize->thread_direction[thread],
00868             optimize->thread_direction[thread + 1]);
00869     #endif
00870     for (i = optimize->thread_direction[thread];
00871          i < optimize->thread_direction[thread + 1]; ++i)
00872     {
00873         e = optimize_norm (i);
00874         g_mutex_lock (mutex);
00875         optimize_best_direction (i, e);
00876         optimize_save_variables (i, e);
00877         if (e < optimize->threshold)
00878             optimize->stop = 1;
00879         g_mutex_unlock (mutex);
00880         if (optimize->stop)
00881             break;
00882     #if DEBUG_OPTIMIZE
00883     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00884     #endif
00885     }
00886     #if DEBUG_OPTIMIZE
00887     fprintf (stderr, "optimize_direction_thread: end\n");
00888     #endif
00889     g_thread_exit (NULL);
00890     return NULL;
00891 }
00892
00902 double
00903 optimize_estimate_direction_random (unsigned int variable,
00904                                     unsigned int estimate)
00905 {
00906     double x;
00907     #if DEBUG_OPTIMIZE
00908     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00909     #endif
00910     x = optimize->direction[variable]
00911         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00912     #if DEBUG_OPTIMIZE
00913     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00914             variable, x);
00915     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00916     #endif
00917     return x;
00918 }
00919
00929 double
00930 optimize_estimate_direction_coordinates (unsigned int variable,
00931                                         unsigned int estimate)
00932 {
00933     double x;
00934     #if DEBUG_OPTIMIZE
00935     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00936     #endif
00937     x = optimize->direction[variable];
00938     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00939     {
00940         if (estimate & 1)
00941             x += optimize->step[variable];
00942         else
00943             x -= optimize->step[variable];
00944     }
00945     #if DEBUG_OPTIMIZE
00946     fprintf (stderr,
00947             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00948             variable, x);
00949     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00950     #endif
00951     return x;
00952 }
00953
00960 void
00961 optimize_step_direction (unsigned int simulation)
00962 {
00963     GThread *thread[nthreads_direction];
00964     ParallelData data[nthreads_direction];
00965     unsigned int i, j, k, b;
00966     #if DEBUG_OPTIMIZE
00967     fprintf (stderr, "optimize_step_direction: start\n");

```

```

00968 #endif
00969     for (i = 0; i < optimize->nestimates; ++i)
00970     {
00971         k = (simulation + i) * optimize->nvariables;
00972         b = optimize->simulation_best[0] * optimize->nvariables;
00973         #if DEBUG_OPTIMIZE
00974             fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00975                     simulation + i, optimize->simulation_best[0]);
00976         #endif
00977         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00978         {
00979             #if DEBUG_OPTIMIZE
00980                 fprintf (stderr,
00981                         "optimize_step_direction: estimate=%u best%u=%.14le\n",
00982                         i, j, optimize->value[b]);
00983             #endif
00984             optimize->value[k]
00985                 = optimize->value[b] + optimize_estimate_direction (j, i);
00986             optimize->value[k] = fmin (fmax (optimize->value[k],
00987                                             optimize->rangeminabs[j]),
00988                                       optimize->rangemaxabs[j]);
00989             #if DEBUG_OPTIMIZE
00990                 fprintf (stderr,
00991                         "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00992                         i, j, optimize->value[k]);
00993             #endif
00994         }
00995     }
00996     if (nthreads_direction == 1)
00997         optimize_direction_sequential (simulation);
00998     else
00999     {
01000         for (i = 0; i <= nthreads_direction; ++i)
01001         {
01002             optimize->thread_direction[i]
01003                 = simulation + optimize->nstart_direction
01004                   + i * (optimize->nend_direction - optimize->
01005                        nstart_direction)
01006                   / nthreads_direction;
01007             #if DEBUG_OPTIMIZE
01008                 fprintf (stderr,
01009                         "optimize_step_direction: i=%u thread_direction=%u\n",
01010                         i, optimize->thread_direction[i]);
01011             #endif
01012             for (i = 0; i < nthreads_direction; ++i)
01013             {
01014                 data[i].thread = i;
01015                 thread[i] = g_thread_new
01016                     (NULL, (void (*) ) optimize_direction_thread, &data[i]);
01017             }
01018             for (i = 0; i < nthreads_direction; ++i)
01019                 g_thread_join (thread[i]);
01020         }
01021         #if DEBUG_OPTIMIZE
01022             fprintf (stderr, "optimize_step_direction: end\n");
01023         #endif
01024     }
01025 void
01030 optimize_direction ()
01032 {
01033     unsigned int i, j, k, b, s, adjust;
01034     #if DEBUG_OPTIMIZE
01035         fprintf (stderr, "optimize_direction: start\n");
01036     #endif
01037     for (i = 0; i < optimize->nvariables; ++i)
01038         optimize->direction[i] = 0.;
01039     b = optimize->simulation_best[0] * optimize->nvariables;
01040     s = optimize->nsimulations;
01041     adjust = 1;
01042     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01043     {
01044         #if DEBUG_OPTIMIZE
01045             fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01046                     i, optimize->simulation_best[0]);
01047         #endif
01048         optimize_step_direction (s);
01049         k = optimize->simulation_best[0] * optimize->nvariables;
01050         #if DEBUG_OPTIMIZE
01051             fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01052                     i, optimize->simulation_best[0]);
01053         #endif
01054         if (k == b)
01055         {
01056             if (adjust)
01057                 for (j = 0; j < optimize->nvariables; ++j)

```

```

01058         optimize->step[j] *= 0.5;
01059     for (j = 0; j < optimize->nvariables; ++j)
01060         optimize->direction[j] = 0.;
01061     adjust = 1;
01062 }
01063 else
01064 {
01065     for (j = 0; j < optimize->nvariables; ++j)
01066     {
01067 #if DEBUG_OPTIMIZE
01068         fprintf (stderr,
01069             "optimize_direction: best%u=%.14le old%u=%.14le\n",
01070             j, optimize->value[k + j], j, optimize->value[b + j]);
01071 #endif
01072         optimize->direction[j]
01073             = (1. - optimize->relaxation) * optimize->direction[j]
01074             + optimize->relaxation
01075             * (optimize->value[k + j] - optimize->value[b + j]);
01076 #if DEBUG_OPTIMIZE
01077         fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01078             j, optimize->direction[j]);
01079 #endif
01080     }
01081     adjust = 0;
01082 }
01083 }
01084 #if DEBUG_OPTIMIZE
01085 fprintf (stderr, "optimize_direction: end\n");
01086 #endif
01087 }
01088
01096 double
01097 optimize_genetic_objective (Entity * entity)
01098 {
01099     unsigned int j;
01100     double objective;
01101     char buffer[64];
01102 #if DEBUG_OPTIMIZE
01103     fprintf (stderr, "optimize_genetic_objective: start\n");
01104 #endif
01105     for (j = 0; j < optimize->nvariables; ++j)
01106     {
01107         optimize->value[entity->id * optimize->nvariables + j]
01108             = genetic_get_variable (entity, optimize->genetic_variable + j);
01109     }
01110     objective = optimize_norm (entity->id);
01111     g_mutex_lock (mutex);
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01115         fprintf (optimize->file_variables, buffer,
01116             genetic_get_variable (entity, optimize->genetic_variable + j));
01117     }
01118     fprintf (optimize->file_variables, "%.14le\n", objective);
01119     g_mutex_unlock (mutex);
01120 #if DEBUG_OPTIMIZE
01121     fprintf (stderr, "optimize_genetic_objective: end\n");
01122 #endif
01123     return objective;
01124 }
01125
01130 void
01131 optimize_genetic ()
01132 {
01133     char *best_genome;
01134     double best_objective, *best_variable;
01135 #if DEBUG_OPTIMIZE
01136     fprintf (stderr, "optimize_genetic: start\n");
01137     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01138         nthreads);
01139     fprintf (stderr,
01140         "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01141         optimize->nvariables, optimize->nsimulations, optimize->
01142         niterations);
01143     fprintf (stderr,
01144         "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01145         optimize->mutation_ratio, optimize->reproduction_ratio,
01146         optimize->adaptation_ratio);
01147 #endif
01147     genetic_algorithm_default (optimize->nvariables,
01148         optimize->genetic_variable,
01149         optimize->nsimulations,
01150         optimize->niterations,
01151         optimize->mutation_ratio,
01152         optimize->reproduction_ratio,
01153         optimize->adaptation_ratio,
01154         optimize->seed,

```

```

01155             optimize->threshold,
01156             &optimize_genetic_objective,
01157             &best_genome, &best_variable, &best_objective);
01158 #if DEBUG_OPTIMIZE
01159     fprintf (stderr, "optimize_genetic: the best\n");
01160 #endif
01161     optimize->error_old = (double *) g_malloc (sizeof (double));
01162     optimize->value_old
01163         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01164     optimize->error_old[0] = best_objective;
01165     memcpy (optimize->value_old, best_variable,
01166            optimize->nvariables * sizeof (double));
01167     g_free (best_genome);
01168     g_free (best_variable);
01169     optimize_print ();
01170 #if DEBUG_OPTIMIZE
01171     fprintf (stderr, "optimize_genetic: end\n");
01172 #endif
01173 }
01174
01175 void
01176 optimize_save_old ()
01177 {
01178     unsigned int i, j;
01179 #if DEBUG_OPTIMIZE
01180     fprintf (stderr, "optimize_save_old: start\n");
01181     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01182 #endif
01183     memcpy (optimize->error_old, optimize->error_best,
01184            optimize->nbest * sizeof (double));
01185     for (i = 0; i < optimize->nbest; ++i)
01186     {
01187         j = optimize->simulation_best[i];
01188 #if DEBUG_OPTIMIZE
01189         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01190 #endif
01191         memcpy (optimize->value_old + i * optimize->nvariables,
01192                optimize->value + j * optimize->nvariables,
01193                optimize->nvariables * sizeof (double));
01194     }
01195 #if DEBUG_OPTIMIZE
01196     for (i = 0; i < optimize->nvariables; ++i)
01197         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01198                 i, optimize->value_old[i]);
01199     fprintf (stderr, "optimize_save_old: end\n");
01200 #endif
01201 }
01202
01203 void
01204 optimize_merge_old ()
01205 {
01206     unsigned int i, j, k;
01207     double v[optimize->nbest * optimize->nvariables], e[optimize->
01208         nbest],
01209          *enew, *eold;
01210 #if DEBUG_OPTIMIZE
01211     fprintf (stderr, "optimize_merge_old: start\n");
01212 #endif
01213     anew = optimize->error_best;
01214     eold = optimize->error_old;
01215     i = j = k = 0;
01216     do
01217     {
01218         if (*enew < *eold)
01219         {
01220             memcpy (v + k * optimize->nvariables,
01221                    optimize->value
01222                    + optimize->simulation_best[i] * optimize->
01223                    nvariables,
01224                    optimize->nvariables * sizeof (double));
01225             e[k] = *enew;
01226             ++k;
01227             ++enew;
01228             ++i;
01229         }
01230         else
01231         {
01232             memcpy (v + k * optimize->nvariables,
01233                    optimize->value_old + j * optimize->nvariables,
01234                    optimize->nvariables * sizeof (double));
01235             e[k] = *eold;
01236             ++k;
01237             ++eold;
01238             ++j;
01239         }
01240     }
01241     while (k < optimize->nbest);

```

```

01249 memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01250 memcpy (optimize->error_old, e, k * sizeof (double));
01251 #if DEBUG_OPTIMIZE
01252 fprintf (stderr, "optimize_merge_old: end\n");
01253 #endif
01254 }
01255
01261 void
01262 optimize_refine ()
01263 {
01264     unsigned int i, j;
01265     double d;
01266     #if HAVE_MPI
01267     MPI_Status mpi_stat;
01268     #endif
01269     #if DEBUG_OPTIMIZE
01270     fprintf (stderr, "optimize_refine: start\n");
01271     #endif
01272     #if HAVE_MPI
01273     if (!optimize->mpi_rank)
01274     {
01275     #endif
01276         for (j = 0; j < optimize->nvariables; ++j)
01277         {
01278             optimize->rangemin[j] = optimize->rangemax[j]
01279             = optimize->value_old[j];
01280         }
01281         for (i = 0; ++i < optimize->nbest; )
01282         {
01283             for (j = 0; j < optimize->nvariables; ++j)
01284             {
01285                 optimize->rangemin[j]
01286                 = fmin (optimize->rangemin[j],
01287                     optimize->value_old[i * optimize->nvariables + j]);
01288                 optimize->rangemax[j]
01289                 = fmax (optimize->rangemax[j],
01290                     optimize->value_old[i * optimize->nvariables + j]);
01291             }
01292         }
01293         for (j = 0; j < optimize->nvariables; ++j)
01294         {
01295             d = optimize->tolerance
01296             * (optimize->rangemax[j] - optimize->rangemin[j]);
01297             switch (optimize->algorithm)
01298             {
01299                 case ALGORITHM_MONTE_CARLO:
01300                     d *= 0.5;
01301                     break;
01302                 default:
01303                     if (optimize->nsweeps[j] > 1)
01304                         d /= optimize->nsweeps[j] - 1;
01305                     else
01306                         d = 0.;
01307             }
01308             optimize->rangemin[j] -= d;
01309             optimize->rangemin[j]
01310             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01311             optimize->rangemax[j] += d;
01312             optimize->rangemax[j]
01313             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01314             printf ("%s min=%lg max=%lg\n", optimize->label[j],
01315                 optimize->rangemin[j], optimize->rangemax[j]);
01316             fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01317                 optimize->label[j], optimize->rangemin[j],
01318                 optimize->rangemax[j]);
01319         }
01320     #if HAVE_MPI
01321         for (i = 1; i < ntasks; ++i)
01322         {
01323             MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01324                 1, MPI_COMM_WORLD);
01325             MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01326                 1, MPI_COMM_WORLD);
01327         }
01328     }
01329     else
01330     {
01331         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01332             MPI_COMM_WORLD, &mpi_stat);
01333         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01334             MPI_COMM_WORLD, &mpi_stat);
01335     }
01336     #endif
01337     #if DEBUG_OPTIMIZE
01338     fprintf (stderr, "optimize_refine: end\n");
01339     #endif
01340 }

```

```

01341
01346 void
01347 optimize_step ()
01348 {
01349     #if DEBUG_OPTIMIZE
01350         fprintf (stderr, "optimize_step: start\n");
01351     #endif
01352     optimize_algorithm ();
01353     if (optimize->nsteps)
01354         optimize_direction ();
01355     #if DEBUG_OPTIMIZE
01356         fprintf (stderr, "optimize_step: end\n");
01357     #endif
01358 }
01359
01364 void
01365 optimize_iterate ()
01366 {
01367     unsigned int i;
01368     #if DEBUG_OPTIMIZE
01369         fprintf (stderr, "optimize_iterate: start\n");
01370     #endif
01371     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01372     optimize->value_old = (double *)
01373         g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01374     optimize_step ();
01375     optimize_save_old ();
01376     optimize_refine ();
01377     optimize_print ();
01378     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01379     {
01380         optimize_step ();
01381         optimize_merge_old ();
01382         optimize_refine ();
01383         optimize_print ();
01384     }
01385     #if DEBUG_OPTIMIZE
01386         fprintf (stderr, "optimize_iterate: end\n");
01387     #endif
01388 }
01389
01394 void
01395 optimize_free ()
01396 {
01397     unsigned int i, j;
01398     #if DEBUG_OPTIMIZE
01399         fprintf (stderr, "optimize_free: start\n");
01400     #endif
01401     for (j = 0; j < optimize->ninputs; ++j)
01402     {
01403         for (i = 0; i < optimize->nexperiments; ++i)
01404             g_mapped_file_unref (optimize->file[j][i]);
01405         g_free (optimize->file[j]);
01406     }
01407     g_free (optimize->error_old);
01408     g_free (optimize->value_old);
01409     g_free (optimize->value);
01410     g_free (optimize->genetic_variable);
01411     #if DEBUG_OPTIMIZE
01412         fprintf (stderr, "optimize_free: end\n");
01413     #endif
01414 }
01415
01420 void
01421 optimize_open ()
01422 {
01423     GTimeZone *tz;
01424     GDateTime *t0, *t;
01425     unsigned int i, j;
01426
01427     #if DEBUG_OPTIMIZE
01428         char *buffer;
01429         fprintf (stderr, "optimize_open: start\n");
01430     #endif
01431
01432     // Getting initial time
01433     #if DEBUG_OPTIMIZE
01434         fprintf (stderr, "optimize_open: getting initial time\n");
01435     #endif
01436     tz = g_time_zone_new_utc ();
01437     t0 = g_date_time_new_now (tz);
01438
01439     // Obtaining and initing the pseudo-random numbers generator seed
01440     #if DEBUG_OPTIMIZE
01441         fprintf (stderr, "optimize_open: getting initial seed\n");
01442     #endif
01443     if (optimize->seed == DEFAULT_RANDOM_SEED)

```

```

01444     optimize->seed = input->seed;
01445     gsl_rng_set (optimize->rng, optimize->seed);
01446
01447     // Replacing the working directory
01448     #if DEBUG_OPTIMIZE
01449     fprintf (stderr, "optimize_open: replacing the working directory\n");
01450     #endif
01451     g_chdir (input->directory);
01452
01453     // Getting results file names
01454     optimize->result = input->result;
01455     optimize->variables = input->variables;
01456
01457     // Obtaining the simulator file
01458     optimize->simulator = input->simulator;
01459
01460     // Obtaining the evaluator file
01461     optimize->evaluator = input->evaluator;
01462
01463     // Reading the algorithm
01464     optimize->algorithm = input->algorithm;
01465     switch (optimize->algorithm)
01466     {
01467     case ALGORITHM_MONTE_CARLO:
01468         optimize_algorithm = optimize_MonteCarlo;
01469         break;
01470     case ALGORITHM_SWEEP:
01471         optimize_algorithm = optimize_sweep;
01472         break;
01473     default:
01474         optimize_algorithm = optimize_genetic;
01475         optimize->mutation_ratio = input->mutation_ratio;
01476         optimize->reproduction_ratio = input->
reproduction_ratio;
01477         optimize->adaptation_ratio = input->adaptation_ratio;
01478     }
01479     optimize->nvariables = input->nvariables;
01480     optimize->nsimulations = input->nsimulations;
01481     optimize->niterations = input->niterations;
01482     optimize->nbest = input->nbest;
01483     optimize->tolerance = input->tolerance;
01484     optimize->nsteps = input->nsteps;
01485     optimize->nestimates = 0;
01486     optimize->threshold = input->threshold;
01487     optimize->stop = 0;
01488     if (input->nsteps)
01489     {
01490         optimize->relaxation = input->relaxation;
01491         switch (input->direction)
01492         {
01493         case DIRECTION_METHOD_COORDINATES:
01494             optimize->nestimates = 2 * optimize->nvariables;
01495             optimize_estimate_direction =
optimize_estimate_direction_coordinates;
01496             break;
01497         default:
01498             optimize->nestimates = input->nestimates;
01499             optimize_estimate_direction =
optimize_estimate_direction_random;
01500         }
01501     }
01502
01503     #if DEBUG_OPTIMIZE
01504     fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01505     #endif
01506     optimize->simulation_best
01507     = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01508     optimize->error_best = (double *) alloca (optimize->nbest * sizeof (double));
01509
01510     // Reading the experimental data
01511     #if DEBUG_OPTIMIZE
01512     buffer = g_get_current_dir ();
01513     fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01514     g_free (buffer);
01515     #endif
01516     optimize->nexperiments = input->nexperiments;
01517     optimize->ninputs = input->experiment->ninputs;
01518     optimize->experiment
01519     = (char **) alloca (input->nexperiments * sizeof (char *));
01520     optimize->weight = (double *) alloca (input->nexperiments * sizeof (double));
01521     for (i = 0; i < input->experiment->ninputs; ++i)
01522         optimize->file[i] = (GMappedFile **)
01523         g_malloc (input->nexperiments * sizeof (GMappedFile *));
01524     for (i = 0; i < input->nexperiments; ++i)
01525     {
01526     #if DEBUG_OPTIMIZE
01527         fprintf (stderr, "optimize_open: i=%u\n", i);

```



```

01528 #endif
01529     optimize->experiment[i] = input->experiment[i].
        name;
01530     optimize->weight[i] = input->experiment[i].weight;
01531 #if DEBUG_OPTIMIZE
01532     fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01533             optimize->experiment[i], optimize->weight[i]);
01534 #endif
01535     for (j = 0; j < input->experiment->ninputs; ++j)
01536     {
01537 #if DEBUG_OPTIMIZE
01538         fprintf (stderr, "optimize_open: template%u\n", j + 1);
01539 #endif
01540         optimize->file[j][i]
01541             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01542     }
01543 }
01544
01545 // Reading the variables data
01546 #if DEBUG_OPTIMIZE
01547     fprintf (stderr, "optimize_open: reading variables\n");
01548 #endif
01549     optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01550     j = input->nvariables * sizeof (double);
01551     optimize->rangemin = (double *) alloca (j);
01552     optimize->rangeminabs = (double *) alloca (j);
01553     optimize->rangemax = (double *) alloca (j);
01554     optimize->rangemaxabs = (double *) alloca (j);
01555     optimize->step = (double *) alloca (j);
01556     j = input->nvariables * sizeof (unsigned int);
01557     optimize->precision = (unsigned int *) alloca (j);
01558     optimize->nsweeps = (unsigned int *) alloca (j);
01559     optimize->nbits = (unsigned int *) alloca (j);
01560     for (i = 0; i < input->nvariables; ++i)
01561     {
01562         optimize->label[i] = input->variable[i].name;
01563         optimize->rangemin[i] = input->variable[i].rangemin;
01564         optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
01565         optimize->rangemax[i] = input->variable[i].rangemax;
01566         optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
01567         optimize->precision[i] = input->variable[i].
precision;
01568         optimize->step[i] = input->variable[i].step;
01569         optimize->nsweeps[i] = input->variable[i].nsweeps;
01570         optimize->nbits[i] = input->variable[i].nbits;
01571     }
01572     if (input->algorithm == ALGORITHM_SWEEP)
01573     {
01574         optimize->nsimulations = 1;
01575         for (i = 0; i < input->nvariables; ++i)
01576         {
01577             if (input->algorithm == ALGORITHM_SWEEP)
01578             {
01579                 optimize->nsimulations *= optimize->nsweeps[i];
01580 #if DEBUG_OPTIMIZE
01581                 fprintf (stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01582                         optimize->nsweeps[i], optimize->nsimulations);
01583 #endif
01584             }
01585         }
01586     }
01587     if (optimize->nsteps)
01588         optimize->direction
01589             = (double *) alloca (optimize->nvariables * sizeof (double));
01590
01591 // Setting error norm
01592 switch (input->norm)
01593 {
01594     case ERROR_NORM_EUCLIDIAN:
01595         optimize_norm = optimize_norm_euclidian;
01596         break;
01597     case ERROR_NORM_MAXIMUM:
01598         optimize_norm = optimize_norm_maximum;
01599         break;
01600     case ERROR_NORM_P:
01601         optimize_norm = optimize_norm_p;
01602         optimize->p = input->p;
01603         break;
01604     default:
01605         optimize_norm = optimize_norm_taxicab;
01606 }
01607
01608 // Allocating values
01609 #if DEBUG_OPTIMIZE
01610     fprintf (stderr, "optimize_open: allocating variables\n");

```

```

01611     fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01612              optimize->nvariables, optimize->algorithm);
01613 #endif
01614     optimize->genetic_variable = NULL;
01615     if (optimize->algorithm == ALGORITHM_GENETIC)
01616     {
01617         optimize->genetic_variable = (GeneticVariable *)
01618             g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01619         for (i = 0; i < optimize->nvariables; ++i)
01620         {
01621             #if DEBUG_OPTIMIZE
01622                 fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01623                          i, optimize->rangemin[i], optimize->rangemax[i],
01624                          optimize->nbits[i]);
01625             #endif
01626             optimize->genetic_variable[i].minimum = optimize->
01627                 rangemin[i];
01628             optimize->genetic_variable[i].maximum = optimize->
01629                 rangemax[i];
01630             optimize->genetic_variable[i].nbits = optimize->nbits[i];
01631         }
01632     }
01633     #if DEBUG_OPTIMIZE
01634     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01635              optimize->nvariables, optimize->nsimulations);
01636     #endif
01637     optimize->value = (double *)
01638         g_malloc ((optimize->nsimulations
01639                   + optimize->nestimates * optimize->nsteps)
01640                  * optimize->nvariables * sizeof (double));
01641     // Calculating simulations to perform for each task
01642     #if HAVE_MPI
01643     #if DEBUG_OPTIMIZE
01644     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01645              optimize->mpi_rank, ntasks);
01646     #endif
01647     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01648         ntasks;
01649     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01650         ntasks;
01651     if (optimize->nsteps)
01652     {
01653         optimize->nstart_direction
01654             = optimize->mpi_rank * optimize->nestimates / ntasks;
01655         optimize->nend_direction
01656             = (1 + optimize->mpi_rank) * optimize->nestimates /
01657                 ntasks;
01658     }
01659     #else
01660     optimize->nstart = 0;
01661     optimize->nend = optimize->nsimulations;
01662     if (optimize->nsteps)
01663     {
01664         optimize->nstart_direction = 0;
01665         optimize->nend_direction = optimize->nestimates;
01666     }
01667     #endif
01668     #if DEBUG_OPTIMIZE
01669     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01670              optimize->nend);
01671     #endif
01672     // Calculating simulations to perform for each thread
01673     optimize->thread
01674         = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01675     for (i = 0; i <= nthreads; ++i)
01676     {
01677         optimize->thread[i] = optimize->nstart
01678             + i * (optimize->nend - optimize->nstart) / nthreads;
01679     }
01680     #if DEBUG_OPTIMIZE
01681     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01682              optimize->thread[i]);
01683     #endif
01684     if (optimize->nsteps)
01685     {
01686         optimize->thread_direction = (unsigned int *)
01687             alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01688     }
01689     // Opening result files
01690     optimize->file_result = g_fopen (optimize->result, "w");
01691     optimize->file_variables = g_fopen (optimize->variables, "w");
01692     // Performing the algorithm
01693     switch (optimize->algorithm)
01694     {
01695         // Genetic algorithm

```

```

01693     case ALGORITHM_GENETIC:
01694         optimize_genetic ();
01695         break;
01696
01697         // Iterative algorithm
01698     default:
01699         optimize_iterate ();
01700     }
01701
01702     // Getting calculation time
01703     t = g_date_time_new_now (tz);
01704     optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01705     g_date_time_unref (t);
01706     g_date_time_unref (t0);
01707     g_time_zone_unref (tz);
01708     printf ("%s = %.6lg s\n",
01709             gettext ("Calculation time"), optimize->calculation_time);
01710     fprintf (optimize->file_result, "%s = %.6lg s\n",
01711             gettext ("Calculation time"), optimize->calculation_time);
01712
01713     // Closing result files
01714     fclose (optimize->file_variables);
01715     fclose (optimize->file_result);
01716
01717 #if DEBUG_OPTIMIZE
01718     fprintf (stderr, "optimize_open: end\n");
01719 #endif
01720 }

```

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

Copyright 2012-2016, all rights reserved.

Definition in file [optimize.h](#).

5.19.2 Function Documentation

5.19.2.1 optimize_best()

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

Function to save the best simulations.

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 463 of file [optimize.c](#).

```

00464 {
00465     unsigned int i, j;
00466     double e;
00467     #if DEBUG_OPTIMIZE
00468         fprintf (stderr, "optimize_best: start\n");
00469         fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00470                 optimize->nsaveds, optimize->nbest);
00471     #endif
00472     if (optimize->nsaveds < optimize->nbest
00473         || value < optimize->error_best[optimize->nsaveds - 1])
00474     {
00475         if (optimize->nsaveds < optimize->nbest)
00476             ++optimize->nsaveds;
00477         optimize->error_best[optimize->nsaveds - 1] = value;
00478         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00479         for (i = optimize->nsaveds; --i;)
00480         {
00481             if (optimize->error_best[i] < optimize->
00482                 error_best[i - 1])
00483             {
00484                 j = optimize->simulation_best[i];
00485                 e = optimize->error_best[i];
00486                 optimize->simulation_best[i] = optimize->
00487                     simulation_best[i - 1];
00488                 optimize->error_best[i] = optimize->
00489                     error_best[i - 1];
00490                 optimize->simulation_best[i - 1] = j;
00491                 optimize->error_best[i - 1] = e;
00492             }
00493             else
00494                 break;
00495         }
00496     }
00497     #if DEBUG_OPTIMIZE
00498         fprintf (stderr, "optimize_best: end\n");
00499     #endif
00500 }

```

5.19.2.2 optimize_best_direction()

```

void optimize_best_direction (
    unsigned int simulation,
    double value )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>value</i>	Objective function value.

Definition at line 788 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     }
00800     #if DEBUG_OPTIMIZE
00801         fprintf (stderr,
00802                 "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00803                 simulation, value);
00804     #endif
00805 }

```

```

00804 #endif
00805     }
00806 #if DEBUG_OPTIMIZE
00807     fprintf (stderr, "optimize_best_direction: end\n");
00808 #endif
00809 }

```

5.19.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 818 of file [optimize.c](#).

```

00819 {
00820     unsigned int i, j;
00821     double e;
00822 #if DEBUG_OPTIMIZE
00823     fprintf (stderr, "optimize_direction_sequential: start\n");
00824     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00825             "nend_direction=%u\n",
00826             optimize->nstart_direction, optimize->
00827             nend_direction);
00828 #endif
00829     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00830     {
00831         j = simulation + i;
00832         e = optimize_norm (j);
00833         optimize_best_direction (j, e);
00834         optimize_save_variables (j, e);
00835         if (e < optimize->threshold)
00836         {
00837             optimize->stop = 1;
00838             break;
00839         }
00840 #if DEBUG_OPTIMIZE
00841         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00842 #endif
00843 #if DEBUG_OPTIMIZE
00844     fprintf (stderr, "optimize_direction_sequential: end\n");
00845 #endif
00846 }

```

Here is the call graph for this function:

5.19.2.4 optimize_direction_thread()

```

void* optimize_direction_thread (
    ParallelData * data )

```

Function to estimate the direction search on a thread.

Parameters

<i>data</i>	Function data.
-------------	----------------

Returns

NULL

Definition at line 856 of file [optimize.c](#).

```

00857 {
00858     unsigned int i, thread;
00859     double e;
00860     #if DEBUG_OPTIMIZE
00861     fprintf (stderr, "optimize_direction_thread: start\n");
00862     #endif
00863     thread = data->thread;
00864     #if DEBUG_OPTIMIZE
00865     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00866             thread,
00867             optimize->thread_direction[thread],
00868             optimize->thread_direction[thread + 1]);
00869     #endif
00870     for (i = optimize->thread_direction[thread];
00871          i < optimize->thread_direction[thread + 1]; ++i)
00872     {
00873         e = optimize_norm (i);
00874         g_mutex_lock (mutex);
00875         optimize_best_direction (i, e);
00876         optimize_save_variables (i, e);
00877         if (e < optimize->threshold)
00878             optimize->stop = 1;
00879         g_mutex_unlock (mutex);
00880         if (optimize->stop)
00881             break;
00882     #if DEBUG_OPTIMIZE
00883     fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00884     #endif
00885     }
00886     #if DEBUG_OPTIMIZE
00887     fprintf (stderr, "optimize_direction_thread: end\n");
00888     #endif
00889     g_thread_exit (NULL);
00890     return NULL;
00891 }

```

Here is the call graph for this function:

5.19.2.5 optimize_estimate_direction_coordinates()

```

double optimize_estimate_direction_coordinates (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the direction search vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 930 of file [optimize.c](#).

```

00932 {
00933     double x;
00934     #if DEBUG_OPTIMIZE
00935     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00936     #endif
00937     x = optimize->direction[variable];
00938     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00939     {

```



```

00940     if (estimate & 1)
00941         x += optimize->step[variable];
00942     else
00943         x -= optimize->step[variable];
00944     }
00945 #if DEBUG_OPTIMIZE
00946     fprintf (stderr,
00947             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00948             variable, x);
00949     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00950 #endif
00951     return x;
00952 }

```

5.19.2.6 optimize_estimate_direction_random()

```

double optimize_estimate_direction_random (
    unsigned int variable,
    unsigned int estimate )

```

Function to estimate a component of the direction search vector.

Parameters

<i>variable</i>	Variable number.
<i>estimate</i>	Estimate number.

Definition at line 903 of file [optimize.c](#).

```

00905 {
00906     double x;
00907 #if DEBUG_OPTIMIZE
00908     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00909 #endif
00910     x = optimize->direction[variable]
00911         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00912         step[variable];
00913 #if DEBUG_OPTIMIZE
00914     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00915             variable, x);
00916     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00917 #endif
00918     return x;
00919 }

```

5.19.2.7 optimize_genetic_objective()

```

double optimize_genetic_objective (
    Entity * entity )

```

Function to calculate the objective function of an entity.

Parameters

<i>entity</i>	entity data.
---------------	--------------

Returns

objective function value.

Definition at line 1097 of file [optimize.c](#).

```

01098 {
01099     unsigned int j;
01100     double objective;
01101     char buffer[64];
01102     #if DEBUG_OPTIMIZE
01103     fprintf (stderr, "optimize_genetic_objective: start\n");
01104     #endif
01105     for (j = 0; j < optimize->nvariables; ++j)
01106     {
01107         optimize->value[entity->id * optimize->nvariables + j]
01108         = genetic_get_variable (entity, optimize->genetic_variable + j);
01109     }
01110     objective = optimize_norm (entity->id);
01111     g_mutex_lock (mutex);
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01115         fprintf (optimize->file_variables, buffer,
01116                 genetic_get_variable (entity, optimize->genetic_variable + j));
01117     }
01118     fprintf (optimize->file_variables, "%.14le\n", objective);
01119     g_mutex_unlock (mutex);
01120     #if DEBUG_OPTIMIZE
01121     fprintf (stderr, "optimize_genetic_objective: end\n");
01122     #endif
01123     return objective;
01124 }

```

5.19.2.8 optimize_input()

```

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

```

Function to write the simulation input file.

Parameters

<i>simulation</i>	Simulation number.
<i>input</i>	Input file name.
<i>template</i>	Template of the input file name.

Definition at line 104 of file [optimize.c](#).

```

00105 {
00106     unsigned int i;
00107     char buffer[32], value[32], *buffer2, *buffer3, *content;
00108     FILE *file;
00109     gsize length;
00110     GRegex *regex;
00111
00112     #if DEBUG_OPTIMIZE
00113     fprintf (stderr, "optimize_input: start\n");
00114     #endif
00115
00116     // Checking the file
00117     if (!template)
00118         goto optimize_input_end;
00119

```

```

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

```

5.19.2.9 optimize_merge()

```

void optimize_merge (
    unsigned int nsaveds,
    unsigned int * simulation_best,
    double * error_best )

```

Function to merge the 2 optimization results.

Parameters

<i>nsaveds</i>	Number of saved results.
<i>simulation_best</i>	Array of best simulation numbers.
<i>error_best</i>	Array of best objective function values.

Definition at line 586 of file [optimize.c](#).

```

00588 {
00589     unsigned int i, j, k, s[optimize->nbest];
00590     double e[optimize->nbest];
00591     #if DEBUG_OPTIMIZE
00592     fprintf (stderr, "optimize_merge: start\n");
00593     #endif
00594     i = j = k = 0;
00595     do
00596     {
00597         if (i == optimize->nsaveds)
00598         {
00599             s[k] = simulation_best[j];
00600             e[k] = error_best[j];
00601             ++j;
00602             ++k;
00603             if (j == nsaveds)
00604                 break;
00605         }
00606         else if (j == nsaveds)
00607         {
00608             s[k] = optimize->simulation_best[i];
00609             e[k] = optimize->error_best[i];
00610             ++i;
00611             ++k;
00612             if (i == optimize->nsaveds)
00613                 break;
00614         }
00615         else if (optimize->error_best[i] > error_best[j])
00616         {
00617             s[k] = simulation_best[j];
00618             e[k] = error_best[j];
00619             ++j;
00620             ++k;
00621         }
00622         else
00623         {
00624             s[k] = optimize->simulation_best[i];
00625             e[k] = optimize->error_best[i];
00626             ++i;
00627             ++k;
00628         }
00629     }
00630     while (k < optimize->nbest);
00631     optimize->nsaveds = k;
00632     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00633     memcpy (optimize->error_best, e, k * sizeof (double));
00634     #if DEBUG_OPTIMIZE
00635     fprintf (stderr, "optimize_merge: end\n");
00636     #endif
00637 }

```

5.19.2.10 optimize_norm_euclidian()

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

Function to calculate the Euclidian error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Euclidian error norm.

Definition at line 296 of file [optimize.c](#).

```

00297 {
00298     double e, ei;
00299     unsigned int i;
00300     #if DEBUG_OPTIMIZE
00301     fprintf (stderr, "optimize_norm_euclidian: start\n");
00302     #endif
00303     e = 0.;
00304     for (i = 0; i < optimize->nexperiments; ++i)
00305     {
00306         ei = optimize_parse (simulation, i);
00307         e += ei * ei;
00308     }
00309     e = sqrt (e);
00310     #if DEBUG_OPTIMIZE
00311     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00312     fprintf (stderr, "optimize_norm_euclidian: end\n");
00313     #endif
00314     return e;
00315 }

```

Here is the call graph for this function:

5.19.2.11 optimize_norm_maximum()

```

double optimize_norm_maximum (
    unsigned int simulation )

```

Function to calculate the maximum error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Maximum error norm.

Definition at line 325 of file [optimize.c](#).

```

00326 {
00327     double e, ei;
00328     unsigned int i;
00329     #if DEBUG_OPTIMIZE
00330     fprintf (stderr, "optimize_norm_maximum: start\n");
00331     #endif
00332     e = 0.;
00333     for (i = 0; i < optimize->nexperiments; ++i)
00334     {
00335         ei = fabs (optimize_parse (simulation, i));
00336         e = fmax (e, ei);
00337     }
00338     #if DEBUG_OPTIMIZE
00339     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00340     fprintf (stderr, "optimize_norm_maximum: end\n");
00341     #endif
00342     return e;
00343 }

```

Here is the call graph for this function:

5.19.2.12 optimize_norm_p()

```

double optimize_norm_p (
    unsigned int simulation )

```

Function to calculate the P error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

P error norm.

Definition at line 353 of file [optimize.c](#).

```

00354 {
00355     double e, ei;
00356     unsigned int i;
00357     #if DEBUG_OPTIMIZE
00358     fprintf (stderr, "optimize_norm_p: start\n");
00359     #endif
00360     e = 0.;
00361     for (i = 0; i < optimize->nexperiments; ++i)
00362     {
00363         ei = fabs (optimize_parse (simulation, i));
00364         e += pow (ei, optimize->p);
00365     }
00366     e = pow (e, 1. / optimize->p);
00367     #if DEBUG_OPTIMIZE
00368     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00369     fprintf (stderr, "optimize_norm_p: end\n");
00370     #endif
00371     return e;
00372 }
```

Here is the call graph for this function:

5.19.2.13 optimize_norm_taxicab()

```

double optimize_norm_taxicab (
    unsigned int simulation )
```

Function to calculate the taxicab error norm.

Parameters

<i>simulation</i>	simulation number.
-------------------	--------------------

Returns

Taxicab error norm.

Definition at line 382 of file [optimize.c](#).

```

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

```

00394     fprintf (stderr, "optimize_norm_taxicab: end\n");
00395 #endif
00396     return e;
00397 }

```

Here is the call graph for this function:

5.19.2.14 optimize_parse()

```

double optimize_parse (
    unsigned int simulation,
    unsigned int experiment )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>experiment</i>	Experiment number.

Returns

Objective function value.

Definition at line 190 of file [optimize.c](#).

```

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

```

```

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

```

Here is the call graph for this function:

5.19.2.15 optimize_save_variables()

```

void optimize_save_variables (
    unsigned int simulation,
    double error )

```

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

Parameters

<i>simulation</i>	Simulation number.
<i>error</i>	Error value.

Definition at line 435 of file [optimize.c](#).


```

00436 {
00437     unsigned int i;
00438     char buffer[64];
00439     #if DEBUG_OPTIMIZE
00440     fprintf (stderr, "optimize_save_variables: start\n");
00441     #endif
00442     for (i = 0; i < optimize->nvariables; ++i)
00443     {
00444         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00445         fprintf (optimize->file_variables, buffer,
00446                 optimize->value[simulation * optimize->
nvariables + i]);
00447     }
00448     fprintf (optimize->file_variables, "%.14le\n", error);
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_save_variables: end\n");
00451     #endif
00452 }

```

5.19.2.16 optimize_step_direction()

```

void optimize_step_direction (
    unsigned int simulation )

```

Function to do a step of the direction search method.

Parameters

<i>simulation</i>	Simulation number.
-------------------	--------------------

Definition at line 961 of file [optimize.c](#).

```

00962 {
00963     GThread *thread[nthreads_direction];
00964     ParallelData data[nthreads_direction];
00965     unsigned int i, j, k, b;
00966     #if DEBUG_OPTIMIZE
00967     fprintf (stderr, "optimize_step_direction: start\n");
00968     #endif
00969     for (i = 0; i < optimize->nestimates; ++i)
00970     {
00971         k = (simulation + i) * optimize->nvariables;
00972         b = optimize->simulation_best[0] * optimize->
nvariables;
00973     #if DEBUG_OPTIMIZE
00974         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00975                 simulation + i, optimize->simulation_best[0]);
00976     #endif
00977         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00978         {
00979             #if DEBUG_OPTIMIZE
00980             fprintf (stderr,
00981                     "optimize_step_direction: estimate=%u best%u=%.14le\n",
00982                     i, j, optimize->value[b]);
00983             #endif
00984             optimize->value[k]
00985             = optimize->value[b] + optimize_estimate_direction (j,
i);
00986             optimize->value[k] = fmin (fmax (optimize->value[k],
00987                                         optimize->rangeminabs[j]),
00988                                         optimize->rangemaxabs[j]);
00989             #if DEBUG_OPTIMIZE
00990             fprintf (stderr,
00991                     "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00992                     i, j, optimize->value[k]);
00993             #endif
00994         }
00995     }
00996     if (nthreads_direction == 1)
00997         optimize_direction_sequential (simulation);
00998     else
00999     {
01000         for (i = 0; i <= nthreads_direction; ++i)

```

```

01001     {
01002         optimize->thread_direction[i]
01003         = simulation + optimize->nstart_direction
01004         + i * (optimize->nend_direction - optimize->
nstart_direction)
01005         / nthreads_direction;
01006 #if DEBUG_OPTIMIZE
01007     fprintf (stderr,
01008             "optimize_step_direction: i=%u thread_direction=%u\n",
01009             i, optimize->thread_direction[i]);
01010 #endif
01011     }
01012     for (i = 0; i < nthreads_direction; ++i)
01013     {
01014         data[i].thread = i;
01015         thread[i] = g_thread_new
01016             (NULL, (void (*)(*)) optimize_direction_thread, &data[i]);
01017     }
01018     for (i = 0; i < nthreads_direction; ++i)
01019         g_thread_join (thread[i]);
01020 }
01021 #if DEBUG_OPTIMIZE
01022 fprintf (stderr, "optimize_step_direction: end\n");
01023 #endif
01024 }

```

Here is the call graph for this function:

5.19.2.17 optimize_thread()

```

void* optimize_thread (
    ParallelData * data )

```

Function to optimize on a thread.

Parameters

<i>data</i>	Function data.
-------------	----------------

Returns

NULL

Definition at line 540 of file [optimize.c](#).

```

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

```

```

00565 #endif
00566     }
00567 #if DEBUG_OPTIMIZE
00568     fprintf (stderr, "optimize_thread: end\n");
00569 #endif
00570     g_thread_exit (NULL);
00571     return NULL;
00572 }

```

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
00032 #ifndef OPTIMIZE__H
00033 #define OPTIMIZE__H 1
00034
00035 typedef struct
00036 {
00037     GMappedFile **file[MAX_NINPUTS];
00038     char **experiment;
00039     char **label;
00040     gsl_rng *rng;
00041     GeneticVariable *genetic_variable;
00042     FILE *file_result;
00043     FILE *file_variables;
00044     char *result;
00045     char *variables;
00046     char *simulator;
00047     char *evaluator;
00048     double *value;
00049     double *rangemin;
00050     double *rangemax;
00051     double *rangeminabs;
00052     double *rangemaxabs;
00053     double *error_best;
00054     double *weight;
00055     double *step;
00056     double *direction;
00057     double *value_old;
00058     double *error_old;
00059     unsigned int *precision;
00060     unsigned int *nsweeps;
00061     unsigned int *nbits;
00062     unsigned int *thread;
00063     unsigned int *thread_direction;
00064     unsigned int *simulation_best;
00065     double tolerance;
00066     double mutation_ratio;
00067     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 <glib.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

- GtkWidget * [main_window](#)
- Main GtkWidget.*
- 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 [cores_number\(\)](#)

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [541](#) of file [utils.c](#).

```
00542 {
00543     #ifdef G_OS_WIN32
00544         SYSTEM_INFO sysinfo;
00545         GetSystemInfo (&sysinfo);
00546         return sysinfo.dwNumberOfProcessors;
00547     #else
00548         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00549     #endif
00550 }
```

5.21.2.2 gtk_array_get_active()

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

Function to get the active GtkRadioButton.

Parameters

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 565 of file [utils.c](#).

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

5.21.2.3 json_object_get_float()

```
double json_object_get_float (  
    JsonObject * object,  
    const char * prop,  
    int * error_code )
```

Function to get a floating point number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 431 of file [utils.c](#).

```
00432 {  
00433     const char *buffer;  
00434     double x = 0.;  
00435     buffer = json_object_get_string_member (object, prop);
```

```

00436  if (!buffer)
00437      *error_code = 1;
00438  else
00439      {
00440          if (sscanf (buffer, "%lf", &x) != 1)
00441              *error_code = 2;
00442          else
00443              *error_code = 0;
00444      }
00445  return x;
00446  }

```

5.21.2.4 json_object_get_float_with_default()

```

double json_object_get_float_with_default (
    JsonObject * object,
    const char * prop,
    double default_value,
    int * error_code )

```

Function to get a floating point number of a JSON object property with a default value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line [464](#) of file [utils.c](#).

```

00466  {
00467      double x;
00468      if (json_object_get_member (object, prop))
00469          x = json_object_get_float (object, prop, error_code);
00470      else
00471      {
00472          x = default_value;
00473          *error_code = 0;
00474      }
00475      return x;
00476  }

```

Here is the call graph for this function:

5.21.2.5 json_object_get_int()

```

int json_object_get_int (
    JsonObject * object,
    const char * prop,
    int * error_code )

```

Function to get an integer number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Integer number value.

Definition at line 341 of file [utils.c](#).

```
00342 {  
00343     const char *buffer;  
00344     int i = 0;  
00345     buffer = json_object_get_string_member (object, prop);  
00346     if (!buffer)  
00347         *error_code = 1;  
00348     else  
00349     {  
00350         if (sscanf (buffer, "%d", &i) != 1)  
00351             *error_code = 2;  
00352         else  
00353             *error_code = 0;  
00354     }  
00355     return i;  
00356 }
```

5.21.2.6 json_object_get_uint()

```
int json_object_get_uint (  
    JsonObject * object,  
    const char * prop,  
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 371 of file [utils.c](#).

```
00372 {  
00373     const char *buffer;  
00374     unsigned int i = 0;  
00375     buffer = json_object_get_string_member (object, prop);  
00376     if (!buffer)  
00377         *error_code = 1;  
00378     else
```

```

00379     {
00380         if (sscanf (buffer, "%u", &i) != 1)
00381             *error_code = 2;
00382         else
00383             *error_code = 0;
00384     }
00385     return i;
00386 }

```

5.21.2.7 json_object_get_uint_with_default()

```

int json_object_get_uint_with_default (
    JsonObject * object,
    const char * prop,
    unsigned int default_value,
    int * error_code )

```

Function to get an unsigned integer number of a JSON object property with a default value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line [404](#) of file [utils.c](#).

```

00406 {
00407     unsigned int i;
00408     if (json_object_get_member (object, prop))
00409         i = json_object_get_uint (object, prop, error_code);
00410     else
00411     {
00412         i = default_value;
00413         *error_code = 0;
00414     }
00415     return i;
00416 }

```

Here is the call graph for this function:

5.21.2.8 json_object_set_float()

```

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

```

Function to set a floating point number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Floating point number value.

Definition at line 528 of file [utils.c](#).

```
00529 {  
00530     char buffer[64];  
00531     snprintf (buffer, 64, "%.14lg", value);  
00532     json_object_set_string_member (object, prop, buffer);  
00533 }
```

5.21.2.9 json_object_set_int()

```
void json_object_set_int (  
    JsonObject * object,  
    const char * prop,  
    int value )
```

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

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Integer number value.

Definition at line 490 of file [utils.c](#).

```
00491 {  
00492     char buffer[64];  
00493     snprintf (buffer, 64, "%d", value);  
00494     json_object_set_string_member (object, prop, buffer);  
00495 }
```

5.21.2.10 json_object_set_uint()

```
void json_object_set_uint (  
    JsonObject * object,  
    const char * prop,  
    unsigned int value )
```

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

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Unsigned integer number value.

Definition at line 509 of file [utils.c](#).

```
00510 {
00511     char buffer[64];
00512     snprintf (buffer, 64, "%u", value);
00513     json_object_set_string_member (object, prop, buffer);
00514 }
```

5.21.2.11 show_error()

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

Function to show a dialog with an error message.

Parameters

<i>msg</i>	Error message.
------------	----------------

Definition at line 114 of file [utils.c](#).

```
00115 {
00116     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00117 }
```

Here is the call graph for this function:

5.21.2.12 show_message()

```
void show_message (
    char * title,
    char * msg,
    int type )
```

Function to show a dialog with a message.

Parameters

<i>title</i>	Title.
<i>msg</i>	Message.
<i>type</i>	Message type.

Definition at line 84 of file [utils.c](#).

```
00085 {
00086     #if HAVE_GTK
00087     GtkMessageDialog *dlg;
00088
00089     // Creating the dialog
00090     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00091         (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00092 }
```

```

00093 // Setting the dialog title
00094 gtk_window_set_title (GTK_WINDOW (dlg), title);
00095
00096 // Showing the dialog and waiting response
00097 gtk_dialog_run (GTK_DIALOG (dlg));
00098
00099 // Closing and freeing memory
00100 gtk_widget_destroy (GTK_WIDGET (dlg));
00101
00102 #else
00103     printf ("%s: %s\n", title, msg);
00104 #endif
00105 }

```

5.21.2.13 xml_node_get_float()

```

double xml_node_get_float (
    xmlNode * node,
    const xmlChar * prop,
    int * error_code )

```

Function to get a floating point number of a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 224 of file [utils.c](#).

```

00225 {
00226     double x = 0.;
00227     xmlChar *buffer;
00228     buffer = xmlGetProp (node, prop);
00229     if (!buffer)
00230         *error_code = 1;
00231     else
00232     {
00233         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00234             *error_code = 2;
00235         else
00236             *error_code = 0;
00237         xmlFree (buffer);
00238     }
00239     return x;
00240 }

```

5.21.2.14 xml_node_get_float_with_default()

```

double xml_node_get_float_with_default (
    xmlNode * node,
    const xmlChar * prop,
    double default_value,
    int * error_code )

```

Function to get a floating point number of a XML node property with a default value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 258 of file [utils.c](#).

```
00260 {
00261     double x;
00262     if (xmlHasProp (node, prop))
00263         x = xml_node_get_float (node, prop, error_code);
00264     else
00265     {
00266         x = default_value;
00267         *error_code = 0;
00268     }
00269     return x;
00270 }
```

Here is the call graph for this function:

5.21.2.15 xml_node_get_int()

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

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Integer number value.

Definition at line 132 of file [utils.c](#).

```
00133 {
00134     int i = 0;
00135     xmlChar *buffer;
00136     buffer = xmlGetProp (node, prop);
00137     if (!buffer)
00138         *error_code = 1;
00139     else
```

```

00140     {
00141         if (sscanf ((char *) buffer, "%d", &i) != 1)
00142             *error_code = 2;
00143         else
00144             *error_code = 0;
00145         xmlFree (buffer);
00146     }
00147     return i;
00148 }

```

5.21.2.16 xml_node_get_uint()

```

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

```

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 163 of file [utils.c](#).

```

00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }

```

5.21.2.17 xml_node_get_uint_with_default()

```

int xml_node_get_uint_with_default (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int default_value,
    int * error_code )

```

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 197 of file [utils.c](#).

```

00199 {
00200     unsigned int i;
00201     if (xmlHasProp (node, prop))
00202         i = xml_node_get_uint (node, prop, error_code);
00203     else
00204     {
00205         i = default_value;
00206         *error_code = 0;
00207     }
00208     return i;
00209 }
```

Here is the call graph for this function:

5.21.2.18 xml_node_set_float()

```

void xml_node_set_float (
    xmlNode * node,
    const xmlChar * prop,
    double value )
```

Function to set a floating point number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Floating point number value.

Definition at line 321 of file [utils.c](#).

```

00322 {
00323     xmlChar buffer[64];
00324     snprintf ((char *) buffer, 64, "%.14lg", value);
00325     xmlSetProp (node, prop, buffer);
00326 }
```

5.21.2.19 xml_node_set_int()

```

void xml_node_set_int (
    xmlNode * node,
```



```

    const xmlChar * prop,
    int value )

```

Function to set an integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Integer number value.

Definition at line 283 of file [utils.c](#).

```

00284 {
00285     xmlChar buffer[64];
00286     snprintf ((char *) buffer, 64, "%d", value);
00287     xmlSetProp (node, prop, buffer);
00288 }

```

5.21.2.20 xml_node_set_uint()

```

void xml_node_set_uint (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int value )

```

Function to set an unsigned integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Unsigned integer number value.

Definition at line 302 of file [utils.c](#).

```

00303 {
00304     xmlChar buffer[64];
00305     snprintf ((char *) buffer, 64, "%u", value);
00306     xmlSetProp (node, prop, buffer);
00307 }

```

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 Burquete 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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <unistd.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <json-glib/json-glib.h>
00040 #ifdef G_OS_WIN32
00041 #include <windows.h>
00042 #endif
00043 #if HAVE_GTK
00044 #include <gtk/gtk.h>
00045 #endif
00046 #include "utils.h"
00047
00048 #if HAVE_GTK
00049 GtkWidget *main_window;
00050 #endif
00051
00052 char *error_message;
00053
00054 void
00055 show_pending ()
00056 {
00057     #if HAVE_GTK
00058     while (gtk_events_pending ())
00059         gtk_main_iteration ();
00060     #endif
00061 }
00062
00063 void
00064 show_message (char *title, char *msg, int type)
00065 {
00066     #if HAVE_GTK
00067     GtkMessageDialog *dlg;
00068
00069     // Creating the dialog
00070     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00071         (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00072
00073     // Setting the dialog title
00074     gtk_window_set_title (GTK_WINDOW (dlg), title);
00075
00076     // Showing the dialog and waiting response
00077     gtk_dialog_run (GTK_DIALOG (dlg));
00078
00079     // Closing and freeing memory
00080     gtk_widget_destroy (GTK_WIDGET (dlg));
00081     #else
00082     printf ("%s: %s\n", title, msg);
00083     #endif
00084 }
00085
00086 void
00087 show_error (char *msg)
00088 {
00089     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00090 }
00091
00092 int
00093 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00094 {
00095     int i = 0;
00096     xmlChar *buffer;
00097     buffer = xmlGetProp (node, prop);

```

```

00137     if (!buffer)
00138         *error_code = 1;
00139     else
00140     {
00141         if (sscanf ((char *) buffer, "%d", &i) != 1)
00142             *error_code = 2;
00143         else
00144             *error_code = 0;
00145         xmlFree (buffer);
00146     }
00147     return i;
00148 }
00149
00162 unsigned int
00163 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }
00180
00196 unsigned int
00197 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
                                unsigned int default_value, int *error_code)
00198 {
00199     unsigned int i;
00200     if (xmlHasProp (node, prop))
00201         i = xml_node_get_uint (node, prop, error_code);
00202     else
00203     {
00204         i = default_value;
00205         *error_code = 0;
00206     }
00207     return i;
00208 }
00209
00210
00223 double
00224 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00225 {
00226     double x = 0.;
00227     xmlChar *buffer;
00228     buffer = xmlGetProp (node, prop);
00229     if (!buffer)
00230         *error_code = 1;
00231     else
00232     {
00233         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00234             *error_code = 2;
00235         else
00236             *error_code = 0;
00237         xmlFree (buffer);
00238     }
00239     return x;
00240 }
00241
00257 double
00258 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
                                double default_value, int *error_code)
00259 {
00260     double x;
00261     if (xmlHasProp (node, prop))
00262         x = xml_node_get_float (node, prop, error_code);
00263     else
00264     {
00265         x = default_value;
00266         *error_code = 0;
00267     }
00268     return x;
00269 }
00270
00271
00282 void
00283 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00284 {
00285     xmlChar buffer[64];
00286     snprintf ((char *) buffer, 64, "%d", value);
00287     xmlSetProp (node, prop, buffer);

```

```

00288 }
00289
00301 void
00302 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00303 {
00304     xmlChar buffer[64];
00305     snprintf ((char *) buffer, 64, "%u", value);
00306     xmlSetProp (node, prop, buffer);
00307 }
00308
00320 void
00321 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00322 {
00323     xmlChar buffer[64];
00324     snprintf ((char *) buffer, 64, "%.14lg", value);
00325     xmlSetProp (node, prop, buffer);
00326 }
00327
00340 int
00341 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00342 {
00343     const char *buffer;
00344     int i = 0;
00345     buffer = json_object_get_string_member (object, prop);
00346     if (!buffer)
00347         *error_code = 1;
00348     else
00349     {
00350         if (sscanf (buffer, "%d", &i) != 1)
00351             *error_code = 2;
00352         else
00353             *error_code = 0;
00354     }
00355     return i;
00356 }
00357
00370 unsigned int
00371 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00372 {
00373     const char *buffer;
00374     unsigned int i = 0;
00375     buffer = json_object_get_string_member (object, prop);
00376     if (!buffer)
00377         *error_code = 1;
00378     else
00379     {
00380         if (sscanf (buffer, "%u", &i) != 1)
00381             *error_code = 2;
00382         else
00383             *error_code = 0;
00384     }
00385     return i;
00386 }
00387
00403 unsigned int
00404 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00405                                   unsigned int default_value, int *error_code)
00406 {
00407     unsigned int i;
00408     if (json_object_get_member (object, prop))
00409         i = json_object_get_uint (object, prop, error_code);
00410     else
00411     {
00412         i = default_value;
00413         *error_code = 0;
00414     }
00415     return i;
00416 }
00417
00430 double
00431 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00432 {
00433     const char *buffer;
00434     double x = 0.;
00435     buffer = json_object_get_string_member (object, prop);
00436     if (!buffer)
00437         *error_code = 1;
00438     else
00439     {
00440         if (sscanf (buffer, "%lf", &x) != 1)
00441             *error_code = 2;
00442         else
00443             *error_code = 0;
00444     }
00445     return x;
00446 }
00447

```

```

00463 double
00464 json_object_get_float_with_default (JsonObject * object, const char *prop
00465                                     ,
00466                                     double default_value, int *error_code)
00467 {
00468     double x;
00469     if (json_object_get_member (object, prop))
00470         x = json_object_get_float (object, prop, error_code);
00471     else
00472     {
00473         x = default_value;
00474         *error_code = 0;
00475     }
00476     return x;
00477 }
00489 void
00490 json_object_set_int (JsonObject * object, const char *prop, int value)
00491 {
00492     char buffer[64];
00493     snprintf (buffer, 64, "%d", value);
00494     json_object_set_string_member (object, prop, buffer);
00495 }
00496
00508 void
00509 json_object_set_uint (JsonObject * object, const char *prop, unsigned int value)
00510 {
00511     char buffer[64];
00512     snprintf (buffer, 64, "%u", value);
00513     json_object_set_string_member (object, prop, buffer);
00514 }
00515
00527 void
00528 json_object_set_float (JsonObject * object, const char *prop, double value)
00529 {
00530     char buffer[64];
00531     snprintf (buffer, 64, "%.14lg", value);
00532     json_object_set_string_member (object, prop, buffer);
00533 }
00534
00540 int
00541 cores_number ()
00542 {
00543     #ifdef G_OS_WIN32
00544         SYSTEM_INFO sysinfo;
00545         GetSystemInfo (&sysinfo);
00546         return sysinfo.dwNumberOfProcessors;
00547     #else
00548         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00549     #endif
00550 }
00551
00552 #if HAVE_GTK
00553
00564 unsigned int
00565 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00566 {
00567     unsigned int i;
00568     for (i = 0; i < n; ++i)
00569         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570             break;
00571     return i;
00572 }
00573
00574 #endif

```

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

- GtkWidget * [main_window](#)
Main GtkWidget.
- 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 cores_number()

```
int cores_number ( )
```

Function to obtain the cores number.

Returns

Cores number.

Definition at line [541](#) of file [utils.c](#).

```
00542 {  
00543     #ifdef G_OS_WIN32  
00544         SYSTEM_INFO sysinfo;  
00545         GetSystemInfo (&sysinfo);  
00546         return sysinfo.dwNumberOfProcessors;  
00547     #else  
00548         return (int) sysconf (_SC_NPROCESSORS_ONLN);  
00549     #endif  
00550 }
```

5.23.2.2 gtk_array_get_active()

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

Function to get the active GtkWidget.

Parameters

<i>array</i>	Array of GtkRadioButtons.
<i>n</i>	Number of GtkRadioButtons.

Returns

Active GtkRadioButton.

Definition at line 565 of file [utils.c](#).

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

5.23.2.3 json_object_get_float()

```
double json_object_get_float (  
    JsonObject * object,  
    const char * prop,  
    int * error_code )
```

Function to get a floating point number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 431 of file [utils.c](#).

```
00432 {  
00433     const char *buffer;  
00434     double x = 0.;  
00435     buffer = json_object_get_string_member (object, prop);  
00436     if (!buffer)  
00437         *error_code = 1;  
00438     else  
00439     {  
00440         if (sscanf (buffer, "%lf", &x) != 1)  
00441             *error_code = 2;  
00442         else  
00443             *error_code = 0;  
00444     }  
00445     return x;  
00446 }
```


5.23.2.4 json_object_get_float_with_default()

```
double json_object_get_float_with_default (
    JsonObject * object,
    const char * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a JSON object property with a default value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line [464](#) of file [utils.c](#).

```
00466 {
00467     double x;
00468     if (json_object_get_member (object, prop))
00469         x = json_object_get_float (object, prop, error_code);
00470     else
00471     {
00472         x = default_value;
00473         *error_code = 0;
00474     }
00475     return x;
00476 }
```

Here is the call graph for this function:

5.23.2.5 json_object_get_int()

```
int json_object_get_int (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an integer number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Integer number value.

Definition at line 341 of file [utils.c](#).

```
00342 {
00343     const char *buffer;
00344     int i = 0;
00345     buffer = json_object_get_string_member (object, prop);
00346     if (!buffer)
00347         *error_code = 1;
00348     else
00349     {
00350         if (sscanf (buffer, "%d", &i) != 1)
00351             *error_code = 2;
00352         else
00353             *error_code = 0;
00354     }
00355     return i;
00356 }
```

5.23.2.6 json_object_get_uint()

```
unsigned int json_object_get_uint (
    JsonObject * object,
    const char * prop,
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 371 of file [utils.c](#).

```
00372 {
00373     const char *buffer;
00374     unsigned int i = 0;
00375     buffer = json_object_get_string_member (object, prop);
00376     if (!buffer)
00377         *error_code = 1;
00378     else
00379     {
00380         if (sscanf (buffer, "%u", &i) != 1)
00381             *error_code = 2;
00382         else
00383             *error_code = 0;
00384     }
00385     return i;
00386 }
```

5.23.2.7 json_object_get_uint_with_default()

```
unsigned int json_object_get_uint_with_default (  
    JsonObject * object,  
    const char * prop,  
    unsigned int default_value,  
    int * error_code )
```

Function to get an unsigned integer number of a JSON object property with a default value.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line [404](#) of file [utils.c](#).

```
00406 {  
00407     unsigned int i;  
00408     if (json_object_get_member (object, prop))  
00409         i = json_object_get_uint (object, prop, error_code);  
00410     else  
00411     {  
00412         i = default_value;  
00413         *error_code = 0;  
00414     }  
00415     return i;  
00416 }
```

Here is the call graph for this function:

5.23.2.8 json_object_set_float()

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

Function to set a floating point number in a JSON object property.

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Floating point number value.

Definition at line [528](#) of file [utils.c](#).

```
00529 {
00530     char buffer[64];
00531     snprintf (buffer, 64, "%.14lg", value);
00532     json_object_set_string_member (object, prop, buffer);
00533 }
```

5.23.2.9 json_object_set_int()

```
void json_object_set_int (
    JsonObject * object,
    const char * prop,
    int value )
```

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

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Integer number value.

Definition at line 490 of file [utils.c](#).

```
00491 {
00492     char buffer[64];
00493     snprintf (buffer, 64, "%d", value);
00494     json_object_set_string_member (object, prop, buffer);
00495 }
```

5.23.2.10 json_object_set_uint()

```
void json_object_set_uint (
    JsonObject * object,
    const char * prop,
    unsigned int value )
```

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

Parameters

<i>object</i>	JSON object.
<i>prop</i>	JSON property.
<i>value</i>	Unsigned integer number value.

Definition at line 509 of file [utils.c](#).

```
00510 {
00511     char buffer[64];
00512     snprintf (buffer, 64, "%u", value);
00513     json_object_set_string_member (object, prop, buffer);
00514 }
```

5.23.2.11 show_error()

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

Function to show a dialog with an error message.

Parameters

<i>msg</i>	Error message.
------------	----------------

Definition at line 114 of file [utils.c](#).

```
00115 {
00116     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00117 }
```

Here is the call graph for this function:

5.23.2.12 show_message()

```
void show_message (
    char * title,
    char * msg,
    int type )
```

Function to show a dialog with a message.

Parameters

<i>title</i>	Title.
<i>msg</i>	Message.
<i>type</i>	Message type.

Definition at line 84 of file [utils.c](#).

```
00085 {
00086     #if HAVE_GTK
00087         GtkMessageDialog *dlg;
00088
00089         // Creating the dialog
00090         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00091             (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00092
00093         // Setting the dialog title
00094         gtk_window_set_title (GTK_WINDOW (dlg), title);
00095
00096         // Showing the dialog and waiting response
00097         gtk_dialog_run (GTK_DIALOG (dlg));
00098
00099         // Closing and freeing memory
00100         gtk_widget_destroy (GTK_WIDGET (dlg));
00101
00102     #else
00103         printf ("%s: %s\n", title, msg);
00104     #endif
00105 }
```

5.23.2.13 xml_node_get_float()

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

Function to get a floating point number of a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 224 of file [utils.c](#).

```
00225 {
00226     double x = 0.;
00227     xmlChar *buffer;
00228     buffer = xmlGetProp (node, prop);
00229     if (!buffer)
00230         *error_code = 1;
00231     else
00232     {
00233         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00234             *error_code = 2;
00235         else
00236             *error_code = 0;
00237         xmlFree (buffer);
00238     }
00239     return x;
00240 }
```

5.23.2.14 xml_node_get_float_with_default()

```
double xml_node_get_float_with_default (
    xmlNode * node,
    const xmlChar * prop,
    double default_value,
    int * error_code )
```

Function to get a floating point number of a XML node property with a default value.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Floating point number value.

Definition at line 258 of file [utils.c](#).

```
00260 {
00261     double x;
00262     if (xmlHasProp (node, prop))
00263         x = xml_node_get_float (node, prop, error_code);
00264     else
00265     {
00266         x = default_value;
00267         *error_code = 0;
00268     }
00269     return x;
00270 }
```

Here is the call graph for this function:

5.23.2.15 xml_node_get_int()

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

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Integer number value.

Definition at line 132 of file [utils.c](#).

```
00133 {
00134     int i = 0;
00135     xmlChar *buffer;
00136     buffer = xmlGetProp (node, prop);
00137     if (!buffer)
00138         *error_code = 1;
00139     else
00140     {
00141         if (sscanf ((char *) buffer, "%d", &i) != 1)
00142             *error_code = 2;
00143         else
00144             *error_code = 0;
00145         xmlFree (buffer);
00146     }
00147     return i;
00148 }
```

5.23.2.16 xml_node_get_uint()

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

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 163 of file [utils.c](#).

```
00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }
```

5.23.2.17 xml_node_get_uint_with_default()

```
unsigned int xml_node_get_uint_with_default (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int default_value,
    int * error_code )
```

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

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>default_value</i>	default value.
<i>error_code</i>	Error code.

Returns

Unsigned integer number value.

Definition at line 197 of file [utils.c](#).

```

00199 {
00200     unsigned int i;
00201     if (xmlHasProp (node, prop))
00202         i = xml_node_get_uint (node, prop, error_code);
00203     else
00204     {
00205         i = default_value;
00206         *error_code = 0;
00207     }
00208     return i;
00209 }
```

Here is the call graph for this function:

5.23.2.18 xml_node_set_float()

```

void xml_node_set_float (
    xmlNode * node,
    const xmlChar * prop,
    double value )
```

Function to set a floating point number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Floating point number value.

Definition at line 321 of file [utils.c](#).

```

00322 {
00323     xmlChar buffer[64];
00324     snprintf ((char *) buffer, 64, "%.14lg", value);
00325     xmlSetProp (node, prop, buffer);
00326 }
```

5.23.2.19 xml_node_set_int()

```

void xml_node_set_int (
    xmlNode * node,
    const xmlChar * prop,
    int value )
```

Function to set an integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Integer number value.

Definition at line 283 of file [utils.c](#).

```
00284 {
00285     xmlChar buffer[64];
00286     snprintf ((char *) buffer, 64, "%d", value);
00287     xmlSetProp (node, prop, buffer);
00288 }
```

5.23.2.20 xml_node_set_uint()

```
void xml_node_set_uint (
    xmlNode * node,
    const xmlChar * prop,
    unsigned int value )
```

Function to set an unsigned integer number in a XML node property.

Parameters

<i>node</i>	XML node.
<i>prop</i>	XML property.
<i>value</i>	Unsigned integer number value.

Definition at line 302 of file [utils.c](#).

```
00303 {
00304     xmlChar buffer[64];
00305     snprintf ((char *) buffer, 64, "%u", value);
00306     xmlSetProp (node, prop, buffer);
00307 }
```

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 GtkWidget *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

Copyright 2012-2016, all rights reserved.

Definition in file `variable.c`.

5.25.2 Function Documentation

5.25.2.1 `variable_error()`

```
void variable_error (  
    Variable * variable,  
    char * message )
```

Function to print a message error opening an `Variable` struct.

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	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 variable_free()

```

void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	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 variable_new()

```

void variable_new (
    Variable * variable )
```

Function to create a new [Variable](#) struct.

Parameters

<i>variable</i>	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 variable_open_json()

```
int variable_open_json (
    Variable * variable,
    XmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Parameters

<i>variable</i>	Variable struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	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,
00330             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,
00358     LABEL_ABSOLUTE_MAXIMUM,
00359     G_MAXDOUBLE, &error_code);
00360     if (error_code)
00361     {
00362         variable_error (variable, gettext ("bad absolute maximum"));
00363         goto exit_on_error;
00364     }
00365     if (variable->rangemax > variable->rangemaxabs)
00366     {
00367         variable_error (variable, gettext ("maximum range not allowed"));
00368         goto exit_on_error;
00369     }
00370     if (variable->rangemax < variable->rangemin)
00371     {
00372         variable_error (variable, gettext ("bad range"));
00373         goto exit_on_error;
00374     }
00375 }
00376 else
00377 {
00378     variable_error (variable, gettext ("no maximum range"));
00379     goto exit_on_error;
00380 }
00381 variable->precision
00382 = json_object_get_uint_with_default (object,
00383 LABEL_PRECISION,
00384     DEFAULT_PRECISION, &error_code);
00385 if (error_code || variable->precision >= NPRECISIONS)
00386 {
00387     variable_error (variable, gettext ("bad precision"));
00388     goto exit_on_error;
00389 }
00390 if (algorithm == ALGORITHM_SWEEP)
00391 {
00392     if (json_object_get_member (object, LABEL_NSWEEPS))
00393     {
00394         variable->nsweeps
00395         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00396         if (error_code || !variable->nsweeps)
00397         {
00398             variable_error (variable, gettext ("bad sweeps"));
00399             goto exit_on_error;
00400         }
00401     }
00402     else
00403     {
00404         variable_error (variable, gettext ("no sweeps number"));
00405         goto exit_on_error;
00406     }
00407 }
00408 #if DEBUG_VARIABLE
00409     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00410 #endif
00411 if (algorithm == ALGORITHM_GENETIC)
00412 {
00413     // Obtaining bits representing each variable
00414     if (json_object_get_member (object, LABEL_NBITS))
00415     {
00416         variable->nbits
00417         = json_object_get_uint (object, LABEL_NBITS, &error_code);
00418         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,
00431     LABEL_STEP, &error_code);
00432     if (error_code || variable->step < 0.)
00433     {
00434         variable_error (variable, gettext ("bad step size"));
00435         goto exit_on_error;
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 variable_open_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,
    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

Parameters

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

```

Here is the call graph for this function:

5.25.3 Variable Documentation

5.25.3.1 format

```
const char* format[NPRECISIONS]
```

Initial value:

```

= {
    "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
    "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
}

```

Array of C-strings with variable formats.

Definition at line 50 of file [variable.c](#).

5.25.3.2 precision

```
const double precision[NPRECISIONS]
```

Initial value:

```
= {
    1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
    1e-13, 1e-14
}
```

Array of variable precisions.

Definition at line 55 of file [variable.c](#).

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 Burquete 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
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "variable.h"
00041
00042 #define DEBUG_VARIABLE 0
00043
00044 const char *format[NPRECISIONS] = {
00045     "%.01f", "%.11f", "%.21f", "%.31f", "%.41f", "%.51f", "%.61f", "%.71f",
00046     "%.81f", "%.91f", "%.101f", "%.111f", "%.121f", "%.131f", "%.141f"
00047 };
00048
00049 const double precision[NPRECISIONS] = {
00050     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,
00051     1e-13, 1e-14
00052 };
00053
00054 void
00055 variable_new (Variable * variable)
00056 {
00057     #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 *)
00155             LABEL_MINIMUM,
00156             &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad minimum"));
00160             goto exit_on_error;
00161         }
00162         variable->rangeminabs = xml_node_get_float_with_default
00163             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00164             &error_code);
00165         if (error_code)
00166         {
00167             variable_error (variable, gettext ("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170         if (variable->rangemin < variable->rangeminabs)
00171         {
00172             variable_error (variable, gettext ("minimum range not allowed"));
00173             goto exit_on_error;
00174         }
00175     }
00176     else
00177     {
00178         variable_error (variable, gettext ("no minimum range"));
00179         goto exit_on_error;
00180     }
00181     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00182     {
00183         variable->rangemax
00184             = xml_node_get_float (node, (const xmlChar *)
00185             LABEL_MAXIMUM,
00186             &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 *)
00216     LABEL_PRECISION,
00217                                     DEFAULT_PRECISION, &error_code);
00218     if (error_code || variable->precision >= NPRECISIONS)
00219     {
00220         variable_error (variable, gettext ("bad precision"));
00221         goto exit_on_error;
00222     }
00223     if (algorithm == ALGORITHM_SWEEP)
00224     {
00225         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00226         {
00227             variable->nsweeps
00228             = xml_node_get_uint (node, (const xmlChar *)
00229             LABEL_NSWEEPS,
00230                                 &error_code);
00231             if (error_code || !variable->nsweeps)
00232             {
00233                 variable_error (variable, gettext ("bad sweeps"));
00234                 goto exit_on_error;
00235             }
00236         }
00237         else
00238         {
00239             variable_error (variable, gettext ("no sweeps number"));
00240             goto exit_on_error;
00241         }
00242     }
00243     #if DEBUG_VARIABLE
00244     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00245     #endif
00246     if (algorithm == ALGORITHM_GENETIC)
00247     {
00248         // Obtaining bits representing each variable
00249         if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00250         {
00251             variable->nbits
00252             = xml_node_get_uint (node, (const xmlChar *)
00253             LABEL_NBITS,
00254                                 &error_code);
00255             if (error_code || !variable->nbits)
00256             {
00257                 variable_error (variable, gettext ("invalid bits number"));
00258                 goto exit_on_error;
00259             }
00260         }
00261         else
00262         {
00263             variable_error (variable, gettext ("no bits number"));
00264             goto exit_on_error;
00265         }
00266     }
00267     else if (nsteps)
00268     {
00269         variable->step
00270         = xml_node_get_float (node, (const xmlChar *)
00271         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,
00330         LABEL_ABSOLUTE_MINIMUM,
00331         -G_MAXDOUBLE, &error_code);
00332         if (error_code)
00333         {
00334             variable_error (variable, gettext ("bad absolute minimum"));
00335             goto exit_on_error;
00336         }
00337         if (variable->rangemin < variable->rangeminabs)
00338         {
00339             variable_error (variable, gettext ("minimum range not allowed"));
00340             goto exit_on_error;
00341         }
00342     }
00343     else
00344     {
00345         variable_error (variable, gettext ("no minimum range"));
00346         goto exit_on_error;
00347     }
00348     if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350         variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352         if (error_code)
00353         {
00354             variable_error (variable, gettext ("bad maximum"));
00355             goto exit_on_error;
00356         }
00357         variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
00359         LABEL_ABSOLUTE_MAXIMUM,
00360         G_MAXDOUBLE, &error_code);
00361         if (error_code)
00362         {
00363             variable_error (variable, gettext ("bad absolute maximum"));
00364             goto exit_on_error;
00365         }
00366         if (variable->rangemax > variable->rangemaxabs)
00367         {
00368             variable_error (variable, gettext ("maximum range not allowed"));
00369         }
00370     }

```

```

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 #if DEBUG_VARIABLE
00438     fprintf (stderr, "variable_open_json: end\n");
00439 #endif
00440 return 1;
00441 exit_on_error:
00442     variable_free (variable, INPUT_TYPE_JSON);
00443 #if DEBUG_VARIABLE
00444     fprintf (stderr, "variable_open_json: end\n");
00445 #endif
00446 return 0;
00447 }

```

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

Copyright 2012-2016, all rights reserved.

Definition in file [variable.h](#).

5.27.2 Enumeration Type Documentation

5.27.2.1 Algorithm

enum [Algorithm](#)

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO	Monte-Carlo algorithm.
ALGORITHM_SWEEP	Sweep algorithm.
ALGORITHM_GENETIC	Genetic algorithm.

Definition at line 45 of file [variable.h](#).

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

5.27.3 Function Documentation

5.27.3.1 `variable_error()`

```
void variable_error (
    Variable * variable,
    char * message )
```

Function to print a message error opening an [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	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 `variable_free()`

```
void variable_free (
    Variable * variable,
    unsigned int type )
```

Function to free the memory of a [Variable](#) struct.

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	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 variable_new()

```
void variable_new (
    Variable * variable )
```

Function to create a new [Variable](#) struct.

Parameters

<i>variable</i>	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 variable_open_json()

```
int variable_open_json (
    Variable * variable,
    JsonNode * node,
    unsigned int algorithm,
    unsigned int nsteps )
```

Function to open the variable file.

Parameters

<i>variable</i>	Variable struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	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,
00330 LABEL_ABSOLUTE_MINIMUM,
00331                                                     -G_MAXDOUBLE, &error_code);
00332         if (error_code)
00333         {
00334             variable_error (variable, gettext ("bad absolute minimum"));
00335             goto exit_on_error;
00336         }
00337         if (variable->rangemin < variable->rangeminabs)
00338         {
00339             variable_error (variable, gettext ("minimum range not allowed"));
00340             goto exit_on_error;
00341         }
00342     }
00343     else
00344     {
00345         variable_error (variable, gettext ("no minimum range"));
00346         goto exit_on_error;
00347     }
00348     if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350         variable->rangemax
00351             = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352         if (error_code)
00353         {
00354             variable_error (variable, gettext ("bad maximum"));
00355             goto exit_on_error;
00356         }
00357         variable->rangemaxabs
00358             = json_object_get_float_with_default (object,
00359 LABEL_ABSOLUTE_MAXIMUM,
00360                                                     G_MAXDOUBLE, &error_code);
00361         if (error_code)
00362         {
00363             variable_error (variable, gettext ("bad absolute maximum"));
00364             goto exit_on_error;
00365         }
00366         if (variable->rangemax > variable->rangemaxabs)
00367         {
00368             variable_error (variable, gettext ("maximum range not allowed"));
00369             goto exit_on_error;
00370         }
00371         if (variable->rangemax < variable->rangemin)
00372         {
00373             variable_error (variable, gettext ("bad range"));
00374             goto exit_on_error;
00375         }
00376     }
00377     else
00378     {
00379         variable_error (variable, gettext ("no maximum range"));
00380         goto exit_on_error;
00381     }
00382 }

```

```

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

```

Here is the call graph for this function:

5.27.3.5 variable_open_xml()

```

int variable_open_xml (
    Variable * variable,
    xmlNode * node,

```

```

    unsigned int algorithm,
    unsigned int nsteps )

```

Function to open the variable file.

Parameters

<i>variable</i>	Variable struct.
<i>node</i>	XML node.
<i>algorithm</i>	Algorithm type.
<i>nsteps</i>	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 *)
00155         LABEL_MINIMUM,
00156         &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad minimum"));
00160             goto exit_on_error;
00161         }
00162         variable->rangeminabs = xml_node_get_float_with_default
00163         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00164         &error_code);
00165         if (error_code)
00166         {
00167             variable_error (variable, gettext ("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170         if (variable->rangemin < variable->rangeminabs)
00171         {
00172             variable_error (variable, gettext ("minimum range not allowed"));
00173             goto exit_on_error;
00174         }
00175     }
00176     else
00177     {
00178         variable_error (variable, gettext ("no minimum range"));
00179         goto exit_on_error;
00180     }
00181     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00182     {
00183         variable->rangemax
00184         = xml_node_get_float (node, (const xmlChar *)
00185         LABEL_MAXIMUM,
00186         &error_code);
00187         if (error_code)
00188         {
00189             variable_error (variable, gettext ("bad maximum"));
00190             goto exit_on_error;
00191         }
00192         variable->rangemaxabs = xml_node_get_float_with_default
00193         (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
00032 #ifndef VARIABLE__H
00033 #define VARIABLE__H 1
00034
00035 enum Algorithm
00036 {
00037     ALGORITHM_MONTE_CARLO = 0,
00038     ALGORITHM_SWEEP = 1,
00039     ALGORITHM_GENETIC = 2
00040 };
00041
00042 typedef struct
00043 {
00044     char *name;
00045     double rangemin;
00046     double rangemax;
00047     double rangeminabs;
00048     double rangemaxabs;
00049     double step;
00050     unsigned int precision;
00051     unsigned int nsweeps;
00052     unsigned int nbits;
00053 } Variable;
00054
00055 extern const char *format[NPRECISIONS];
00056 extern const double precision[NPRECISIONS];
00057
00058 // Public functions
00059 void variable_new(Variable * variable);
00060 void variable_free(Variable * variable, unsigned int type);
00061 void variable_error(Variable * variable, char *message);
00062 int variable_open_xml(Variable * variable, xmlNode * node,
00063                     unsigned int algorithm, unsigned int nsteps);
00064 int variable_open_json(Variable * variable, JsonNode * node,

```



```
00079         unsigned int algorithm, unsigned int nsteps);  
00080  
00081 #endif
```


Index

Algorithm
variable.h, 247

config.h, 29
cores_number
utils.c, 204
utils.h, 221

DirectionMethod
input.h, 73

ErrorNorm
input.h, 74

Experiment, 15
experiment.c, 30
experiment_error, 31
experiment_free, 32
experiment_new, 32
experiment_open_json, 33
experiment_open_xml, 34
template, 36
experiment.h, 40
experiment_error, 41
experiment_free, 41
experiment_new, 43
experiment_open_json, 43
experiment_open_xml, 45

experiment_error
experiment.c, 31
experiment.h, 41

experiment_free
experiment.c, 32
experiment.h, 41

experiment_new
experiment.c, 32
experiment.h, 43

experiment_open_json
experiment.c, 33
experiment.h, 43

experiment_open_xml
experiment.c, 34
experiment.h, 45

format
variable.c, 240

gtk_array_get_active
interface.h, 137
utils.c, 204
utils.h, 221

Input, 15
input.c, 47
input_error, 49
input_open, 49
input_open_json, 50
input_open_xml, 55

input.h, 72
DirectionMethod, 73
ErrorNorm, 74
input_error, 74
input_open, 75
input_open_json, 76
input_open_xml, 81

input_error
input.c, 49
input.h, 74

input_open
input.c, 49
input.h, 75

input_open_json
input.c, 50
input.h, 76

input_open_xml
input.c, 55
input.h, 81

input_save
interface.c, 90
interface.h, 137

input_save_direction_json
interface.c, 91

input_save_direction_xml
interface.c, 92

input_save_json
interface.c, 92

input_save_xml
interface.c, 95

interface.c, 87
input_save, 90
input_save_direction_json, 91
input_save_direction_xml, 92
input_save_json, 92
input_save_xml, 95
window_get_algorithm, 97
window_get_direction, 97
window_get_norm, 98
window_read, 98
window_save, 100
window_template_experiment, 102
interface.h, 134

- gtk_array_get_active, 137
- input_save, 137
- window_get_algorithm, 138
- window_get_direction, 138
- window_get_norm, 139
- window_read, 139
- window_save, 141
- window_template_experiment, 143
- json_object_get_float
 - utils.c, 205
 - utils.h, 222
- json_object_get_float_with_default
 - utils.c, 206
 - utils.h, 222
- json_object_get_int
 - utils.c, 206
 - utils.h, 223
- json_object_get_uint
 - utils.c, 207
 - utils.h, 224
- json_object_get_uint_with_default
 - utils.c, 208
 - utils.h, 224
- json_object_set_float
 - utils.c, 208
 - utils.h, 225
- json_object_set_int
 - utils.c, 209
 - utils.h, 226
- json_object_set_uint
 - utils.c, 209
 - utils.h, 226
- main.c, 147
- Optimize, 17
 - thread_direction, 19
- optimize.c, 150
 - optimize_best, 153
 - optimize_best_direction, 154
 - optimize_direction_sequential, 154
 - optimize_direction_thread, 155
 - optimize_estimate_direction_coordinates, 156
 - optimize_estimate_direction_random, 156
 - optimize_genetic_objective, 157
 - optimize_input, 157
 - optimize_merge, 159
 - optimize_norm_euclidian, 160
 - optimize_norm_maximum, 160
 - optimize_norm_p, 161
 - optimize_norm_taxicab, 162
 - optimize_parse, 162
 - optimize_save_variables, 164
 - optimize_step_direction, 164
 - optimize_thread, 165
- optimize.h, 185
 - optimize_best, 187
 - optimize_best_direction, 188
 - optimize_direction_sequential, 189
 - optimize_direction_thread, 189
 - optimize_estimate_direction_coordinates, 190
 - optimize_estimate_direction_random, 191
 - optimize_genetic_objective, 191
 - optimize_input, 192
 - optimize_merge, 193
 - optimize_norm_euclidian, 194
 - optimize_norm_maximum, 195
 - optimize_norm_p, 195
 - optimize_norm_taxicab, 196
 - optimize_parse, 197
 - optimize_save_variables, 198
 - optimize_step_direction, 199
 - optimize_thread, 200
- optimize_best
 - optimize.c, 153
 - optimize.h, 187
- optimize_best_direction
 - optimize.c, 154
 - optimize.h, 188
- optimize_direction_sequential
 - optimize.c, 154
 - optimize.h, 189
- optimize_direction_thread
 - optimize.c, 155
 - optimize.h, 189
- optimize_estimate_direction_coordinates
 - optimize.c, 156
 - optimize.h, 190
- optimize_estimate_direction_random
 - optimize.c, 156
 - optimize.h, 191
- optimize_genetic_objective
 - optimize.c, 157
 - optimize.h, 191
- optimize_input
 - optimize.c, 157
 - optimize.h, 192
- optimize_merge
 - optimize.c, 159
 - optimize.h, 193
- optimize_norm_euclidian
 - optimize.c, 160
 - optimize.h, 194
- optimize_norm_maximum
 - optimize.c, 160
 - optimize.h, 195
- optimize_norm_p
 - optimize.c, 161
 - optimize.h, 195
- optimize_norm_taxicab
 - optimize.c, 162
 - optimize.h, 196
- optimize_parse
 - optimize.c, 162
 - optimize.h, 197
- optimize_save_variables

- optimize.c, [164](#)
- optimize.h, [198](#)
- optimize_step_direction
 - optimize.c, [164](#)
 - optimize.h, [199](#)
- optimize_thread
 - optimize.c, [165](#)
 - optimize.h, [200](#)
- Options, [20](#)
- ParallelData, [20](#)
- precision
 - variable.c, [240](#)
- Running, [21](#)
- show_error
 - utils.c, [210](#)
 - utils.h, [226](#)
- show_message
 - utils.c, [210](#)
 - utils.h, [227](#)
- template
 - experiment.c, [36](#)
- thread_direction
 - Optimize, [19](#)
- utils.c, [203](#)
 - cores_number, [204](#)
 - gtk_array_get_active, [204](#)
 - json_object_get_float, [205](#)
 - json_object_get_float_with_default, [206](#)
 - json_object_get_int, [206](#)
 - json_object_get_uint, [207](#)
 - json_object_get_uint_with_default, [208](#)
 - json_object_set_float, [208](#)
 - json_object_set_int, [209](#)
 - json_object_set_uint, [209](#)
 - show_error, [210](#)
 - show_message, [210](#)
 - xml_node_get_float, [211](#)
 - xml_node_get_float_with_default, [211](#)
 - xml_node_get_int, [212](#)
 - xml_node_get_uint, [213](#)
 - xml_node_get_uint_with_default, [213](#)
 - xml_node_set_float, [214](#)
 - xml_node_set_int, [214](#)
 - xml_node_set_uint, [215](#)
- utils.h, [219](#)
 - cores_number, [221](#)
 - gtk_array_get_active, [221](#)
 - json_object_get_float, [222](#)
 - json_object_get_float_with_default, [222](#)
 - json_object_get_int, [223](#)
 - json_object_get_uint, [224](#)
 - json_object_get_uint_with_default, [224](#)
 - json_object_set_float, [225](#)
 - json_object_set_int, [226](#)
 - json_object_set_uint, [226](#)
 - show_error, [226](#)
 - show_message, [227](#)
 - xml_node_get_float, [227](#)
 - xml_node_get_float_with_default, [228](#)
 - xml_node_get_int, [229](#)
 - xml_node_get_uint, [229](#)
 - xml_node_get_uint_with_default, [230](#)
 - xml_node_set_float, [231](#)
 - xml_node_set_int, [231](#)
 - xml_node_set_uint, [232](#)
- Variable, [21](#)
- variable.c, [233](#)
 - format, [240](#)
 - precision, [240](#)
 - variable_error, [234](#)
 - variable_free, [235](#)
 - variable_new, [235](#)
 - variable_open_json, [236](#)
 - variable_open_xml, [238](#)
- variable.h, [246](#)
 - Algorithm, [247](#)
 - variable_error, [248](#)
 - variable_free, [248](#)
 - variable_new, [249](#)
 - variable_open_json, [249](#)
 - variable_open_xml, [251](#)
- variable_error
 - variable.c, [234](#)
 - variable.h, [248](#)
- variable_free
 - variable.c, [235](#)
 - variable.h, [248](#)
- variable_new
 - variable.c, [235](#)
 - variable.h, [249](#)
- variable_open_json
 - variable.c, [236](#)
 - variable.h, [249](#)
- variable_open_xml
 - variable.c, [238](#)
 - variable.h, [251](#)
- Window, [22](#)
- window_get_algorithm
 - interface.c, [97](#)
 - interface.h, [138](#)
- window_get_direction
 - interface.c, [97](#)
 - interface.h, [138](#)
- window_get_norm
 - interface.c, [98](#)
 - interface.h, [139](#)
- window_read
 - interface.c, [98](#)
 - interface.h, [139](#)
- window_save
 - interface.c, [100](#)

- interface.h, [141](#)
- window_template_experiment
 - interface.c, [102](#)
 - interface.h, [143](#)
- xml_node_get_float
 - utils.c, [211](#)
 - utils.h, [227](#)
- xml_node_get_float_with_default
 - utils.c, [211](#)
 - utils.h, [228](#)
- xml_node_get_int
 - utils.c, [212](#)
 - utils.h, [229](#)
- xml_node_get_uint
 - utils.c, [213](#)
 - utils.h, [229](#)
- xml_node_get_uint_with_default
 - utils.c, [213](#)
 - utils.h, [230](#)
- xml_node_set_float
 - utils.c, [214](#)
 - utils.h, [231](#)
- xml_node_set_int
 - utils.c, [214](#)
 - utils.h, [231](#)
- xml_node_set_uint
 - utils.c, [215](#)
 - utils.h, [232](#)