

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	16
4.2.1	Detailed Description	17
4.3	Optimize Struct Reference	17
4.3.1	Detailed Description	20
4.3.2	Field Documentation	20
4.3.2.1	thread_direction	20
4.4	Options Struct Reference	20
4.4.1	Detailed Description	21
4.5	ParallelData Struct Reference	21
4.5.1	Detailed Description	21
4.6	Running Struct Reference	21
4.6.1	Detailed Description	22
4.7	Variable Struct Reference	22
4.7.1	Detailed Description	22
4.8	Window Struct Reference	23
4.8.1	Detailed Description	27

5	File Documentation	29
5.1	config.h File Reference	29
5.1.1	Detailed Description	32
5.1.2	Enumeration Type Documentation	32
5.1.2.1	INPUT_TYPE	32
5.2	config.h	32
5.3	experiment.c File Reference	34
5.3.1	Detailed Description	35
5.3.2	Function Documentation	35
5.3.2.1	experiment_error()	35
5.3.2.2	experiment_free()	36
5.3.2.3	experiment_new()	36
5.3.2.4	experiment_open_json()	37
5.3.2.5	experiment_open_xml()	39
5.3.3	Variable Documentation	41
5.3.3.1	template	41
5.4	experiment.c	41
5.5	experiment.h File Reference	45
5.5.1	Detailed Description	46
5.5.2	Function Documentation	46
5.5.2.1	experiment_error()	46
5.5.2.2	experiment_free()	46
5.5.2.3	experiment_new()	48
5.5.2.4	experiment_open_json()	48
5.5.2.5	experiment_open_xml()	50
5.6	experiment.h	52
5.7	input.c File Reference	53
5.7.1	Detailed Description	54
5.7.2	Function Documentation	54
5.7.2.1	input_error()	54

5.7.2.2	input_open()	55
5.7.2.3	input_open_json()	56
5.7.2.4	input_open_xml()	62
5.8	input.c	68
5.9	input.h File Reference	80
5.9.1	Detailed Description	81
5.9.2	Enumeration Type Documentation	81
5.9.2.1	DirectionMethod	81
5.9.2.2	ErrorNorm	81
5.9.3	Function Documentation	82
5.9.3.1	input_error()	82
5.9.3.2	input_open()	82
5.9.3.3	input_open_json()	84
5.9.3.4	input_open_xml()	90
5.10	input.h	96
5.11	interface.c File Reference	97
5.11.1	Detailed Description	100
5.11.2	Function Documentation	100
5.11.2.1	input_save()	100
5.11.2.2	input_save_direction_json()	101
5.11.2.3	input_save_direction_xml()	102
5.11.2.4	input_save_json()	103
5.11.2.5	input_save_xml()	106
5.11.2.6	window_get_algorithm()	108
5.11.2.7	window_get_direction()	109
5.11.2.8	window_get_norm()	110
5.11.2.9	window_read()	110
5.11.2.10	window_save()	113
5.11.2.11	window_template_experiment()	115
5.12	interface.c	116

5.13 interface.h File Reference	147
5.13.1 Detailed Description	150
5.13.2 Function Documentation	150
5.13.2.1 gtk_array_get_active()	150
5.13.2.2 input_save()	151
5.13.2.3 window_get_algorithm()	152
5.13.2.4 window_get_direction()	153
5.13.2.5 window_get_norm()	153
5.13.2.6 window_read()	154
5.13.2.7 window_save()	156
5.13.2.8 window_template_experiment()	158
5.14 interface.h	159
5.15 main.c File Reference	162
5.15.1 Detailed Description	163
5.16 main.c	163
5.17 optimize.c File Reference	166
5.17.1 Detailed Description	168
5.17.2 Function Documentation	168
5.17.2.1 optimize_best()	168
5.17.2.2 optimize_best_direction()	169
5.17.2.3 optimize_direction_sequential()	170
5.17.2.4 optimize_direction_thread()	171
5.17.2.5 optimize_estimate_direction_coordinates()	172
5.17.2.6 optimize_estimate_direction_random()	172
5.17.2.7 optimize_genetic_objective()	173
5.17.2.8 optimize_input()	173
5.17.2.9 optimize_merge()	175
5.17.2.10 optimize_norm_euclidian()	175
5.17.2.11 optimize_norm_maximum()	176
5.17.2.12 optimize_norm_p()	177

5.17.2.13 optimize_norm_taxicab()	178
5.17.2.14 optimize_parse()	179
5.17.2.15 optimize_save_variables()	181
5.17.2.16 optimize_step_direction()	181
5.17.2.17 optimize_thread()	183
5.18 optimize.c	184
5.19 optimize.h File Reference	202
5.19.1 Detailed Description	204
5.19.2 Function Documentation	204
5.19.2.1 optimize_best()	204
5.19.2.2 optimize_best_direction()	205
5.19.2.3 optimize_direction_sequential()	206
5.19.2.4 optimize_direction_thread()	206
5.19.2.5 optimize_estimate_direction_coordinates()	207
5.19.2.6 optimize_estimate_direction_random()	208
5.19.2.7 optimize_genetic_objective()	209
5.19.2.8 optimize_input()	209
5.19.2.9 optimize_merge()	210
5.19.2.10 optimize_norm_euclidian()	211
5.19.2.11 optimize_norm_maximum()	212
5.19.2.12 optimize_norm_p()	213
5.19.2.13 optimize_norm_taxicab()	214
5.19.2.14 optimize_parse()	215
5.19.2.15 optimize_save_variables()	217
5.19.2.16 optimize_step_direction()	217
5.19.2.17 optimize_thread()	219
5.20 optimize.h	220
5.21 utils.c File Reference	222
5.21.1 Detailed Description	223
5.21.2 Function Documentation	224

5.21.2.1	<code>cores_number()</code>	224
5.21.2.2	<code>gtk_array_get_active()</code>	224
5.21.2.3	<code>json_object_get_float()</code>	225
5.21.2.4	<code>json_object_get_float_with_default()</code>	225
5.21.2.5	<code>json_object_get_int()</code>	226
5.21.2.6	<code>json_object_get_uint()</code>	227
5.21.2.7	<code>json_object_get_uint_with_default()</code>	227
5.21.2.8	<code>json_object_set_float()</code>	228
5.21.2.9	<code>json_object_set_int()</code>	229
5.21.2.10	<code>json_object_set_uint()</code>	229
5.21.2.11	<code>show_error()</code>	230
5.21.2.12	<code>show_message()</code>	230
5.21.2.13	<code>xml_node_get_float()</code>	231
5.21.2.14	<code>xml_node_get_float_with_default()</code>	231
5.21.2.15	<code>xml_node_get_int()</code>	232
5.21.2.16	<code>xml_node_get_uint()</code>	233
5.21.2.17	<code>xml_node_get_uint_with_default()</code>	234
5.21.2.18	<code>xml_node_set_float()</code>	234
5.21.2.19	<code>xml_node_set_int()</code>	235
5.21.2.20	<code>xml_node_set_uint()</code>	235
5.22	<code>utils.c</code>	236
5.23	<code>utils.h</code> File Reference	240
5.23.1	Detailed Description	241
5.23.2	Function Documentation	242
5.23.2.1	<code>cores_number()</code>	242
5.23.2.2	<code>gtk_array_get_active()</code>	242
5.23.2.3	<code>json_object_get_float()</code>	243
5.23.2.4	<code>json_object_get_float_with_default()</code>	243
5.23.2.5	<code>json_object_get_int()</code>	244
5.23.2.6	<code>json_object_get_uint()</code>	245

5.23.2.7	<code>json_object_get_uint_with_default()</code>	245
5.23.2.8	<code>json_object_set_float()</code>	246
5.23.2.9	<code>json_object_set_int()</code>	247
5.23.2.10	<code>json_object_set_uint()</code>	247
5.23.2.11	<code>show_error()</code>	248
5.23.2.12	<code>show_message()</code>	248
5.23.2.13	<code>xml_node_get_float()</code>	249
5.23.2.14	<code>xml_node_get_float_with_default()</code>	249
5.23.2.15	<code>xml_node_get_int()</code>	250
5.23.2.16	<code>xml_node_get_uint()</code>	251
5.23.2.17	<code>xml_node_get_uint_with_default()</code>	252
5.23.2.18	<code>xml_node_set_float()</code>	252
5.23.2.19	<code>xml_node_set_int()</code>	253
5.23.2.20	<code>xml_node_set_uint()</code>	253
5.24	<code>utils.h</code>	254
5.25	<code>variable.c</code> File Reference	255
5.25.1	Detailed Description	256
5.25.2	Function Documentation	256
5.25.2.1	<code>variable_error()</code>	256
5.25.2.2	<code>variable_free()</code>	257
5.25.2.3	<code>variable_new()</code>	257
5.25.2.4	<code>variable_open_json()</code>	257
5.25.2.5	<code>variable_open_xml()</code>	260
5.25.3	Variable Documentation	263
5.25.3.1	<code>format</code>	263
5.25.3.2	<code>precision</code>	263
5.26	<code>variable.c</code>	264
5.27	<code>variable.h</code> File Reference	268
5.27.1	Detailed Description	269
5.27.2	Enumeration Type Documentation	269
5.27.2.1	<code>Algorithm</code>	269
5.27.3	Function Documentation	270
5.27.3.1	<code>variable_error()</code>	270
5.27.3.2	<code>variable_free()</code>	270
5.27.3.3	<code>variable_new()</code>	271
5.27.3.4	<code>variable_open_json()</code>	271
5.27.3.5	<code>variable_open_xml()</code>	274
5.28	<code>variable.h</code>	277

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: $4 \times 5 \times 5 \times 5 \times 5 = 2500$.
- 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	16
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	21
Running	Struct to define the running dialog	21
Variable	Struct to define the variable data	22
Window	Struct to define the main window	23

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	34
experiment.h	Header file to define the experiment data	45
generate.c	??
input.c	Source file to define the input functions	53
input.h	Header file to define the input functions	80
interface.c	Source file to define the graphical interface functions	97
interface.h	Header file to define the graphical interface functions	147
main.c	Main source file	162
optimize.c	Source file to define the optimization functions	166
optimize.h	Header file to define the optimization functions	202
utils.c	Source file to define some useful functions	222
utils.h	Header file to define some useful functions	240
variable.c	Source file to define the variable data	255
variable.h	Header file to define the variable data	268

Chapter 4

Data Structure Documentation

4.1 Experiment Struct Reference

Struct to define the experiment data.

```
#include <experiment.h>
```

Data Fields

- char * [name](#)
File name.
- char * [template](#) [[MAX_NINPUTS](#)]
Array of template names of input files.
- double [weight](#)
Objective function weight.
- unsigned int [ninputs](#)
Number of input files to the simulator.

4.1.1 Detailed Description

Struct to define the experiment data.

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

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

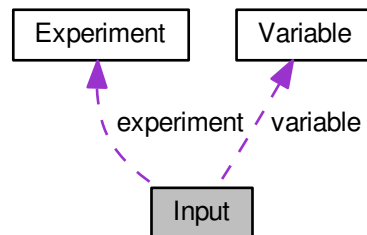
- [experiment.h](#)

4.2 Input Struct Reference

Struct to define the optimization input file.

```
#include <input.h>
```

Collaboration diagram for Input:



Data Fields

- **Experiment** * **experiment**
Array or experiments.
- **Variable** * **variable**
Array of variables.
- char * **result**
Name of the result file.
- char * **variables**
Name of the variables file.
- char * **simulator**
Name of the simulator program.
- char * **evaluator**
Name of the program to evaluate the objective function.
- char * **directory**
Working directory.
- char * **name**
Input data file name.
- double **tolerance**
Algorithm tolerance.
- double **mutation_ratio**
Mutation probability.
- double **reproduction_ratio**
Reproduction probability.
- double **adaptation_ratio**
Adaptation probability.
- double **relaxation**
Relaxation parameter.

- double [p](#)
Exponent of the P error norm.
- double [threshold](#)
Threshold to finish the optimization.
- unsigned long int [seed](#)
Seed of the pseudo-random numbers generator.
- unsigned int [nvariables](#)
Variables number.
- unsigned int [nexperiments](#)
Experiments number.
- unsigned int [nsimulations](#)
Simulations number per experiment.
- unsigned int [algorithm](#)
Algorithm type.
- unsigned int [nsteps](#)
Number of steps to do the direction search method.
- unsigned int [direction](#)
Method to estimate the direction search.
- unsigned int [nestimates](#)
Number of simulations to estimate the direction search.
- unsigned int [niterations](#)
Number of algorithm iterations.
- unsigned int [nbest](#)
Number of best simulations.
- unsigned int [norm](#)
Error norm type.
- unsigned int [type](#)
Type of input file.

4.2.1 Detailed Description

Struct to define the optimization input file.

Definition at line [71](#) of file [input.h](#).

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

- [input.h](#)

4.3 Optimize Struct Reference

Struct to define the optimization ation data.

```
#include <optimize.h>
```

Data Fields

- GMappedFile ** [file](#) [MAX_NINPUTS]
Matrix of input template files.
- char ** [experiment](#)
Array of experimental data file names.
- char ** [label](#)
Array of variable names.
- gsl_rng * [rng](#)
GSL random number generator.
- GeneticVariable * [genetic_variable](#)
Array of variables for the genetic algorithm.
- FILE * [file_result](#)
Result file.
- FILE * [file_variables](#)
Variables file.
- char * [result](#)
Name of the result file.
- char * [variables](#)
Name of the variables file.
- char * [simulator](#)
Name of the simulator program.
- char * [evaluator](#)
Name of the program to evaluate the objective function.
- double * [value](#)
Array of variable values.
- double * [rangemin](#)
Array of minimum variable values.
- double * [rangemax](#)
Array of maximum variable values.
- double * [rangeminabs](#)
Array of absolute minimum variable values.
- double * [rangemaxabs](#)
Array of absolute maximum variable values.
- double * [error_best](#)
Array of the best minimum errors.
- double * [weight](#)
Array of the experiment weights.
- double * [step](#)
Array of direction search method step sizes.
- double * [direction](#)
Vector of direction search estimation.
- double * [value_old](#)
Array of the best variable values on the previous step.
- double * [error_old](#)
Array of the best minimum errors on the previous step.
- unsigned int * [precision](#)
Array of variable precisions.
- unsigned int * [nsweeps](#)
Array of sweeps of the sweep algorithm.
- unsigned int * [nbits](#)

- Array of bits number of the genetic algorithm.*
- unsigned int * [thread](#)
 - Array of simulation numbers to calculate on the thread.*
- unsigned int * [thread_direction](#)
- unsigned int * [simulation_best](#)
 - Array of best simulation numbers.*
- double [tolerance](#)
 - Algorithm tolerance.*
- double [mutation_ratio](#)
 - Mutation probability.*
- double [reproduction_ratio](#)
 - Reproduction probability.*
- double [adaptation_ratio](#)
 - Adaptation probability.*
- double [relaxation](#)
 - Relaxation parameter.*
- double [calculation_time](#)
 - Calculation time.*
- double [p](#)
 - Exponent of the P error norm.*
- double [threshold](#)
 - Threshold to finish the optimization.*
- unsigned long int [seed](#)
 - Seed of the pseudo-random numbers generator.*
- unsigned int [nvariables](#)
 - Variables number.*
- unsigned int [nexperiments](#)
 - Experiments number.*
- unsigned int [ninputs](#)
 - Number of input files to the simulator.*
- unsigned int [nsimulations](#)
 - Simulations number per experiment.*
- unsigned int [nsteps](#)
 - Number of steps for the direction search method.*
- unsigned int [nestimates](#)
 - Number of simulations to estimate the direction.*
- unsigned int [algorithm](#)
 - Algorithm type.*
- unsigned int [nstart](#)
 - Beginning simulation number of the task.*
- unsigned int [nend](#)
 - Ending simulation number of the task.*
- unsigned int [nstart_direction](#)
 - Beginning simulation number of the task for the direction search method.*
- unsigned int [nend_direction](#)
 - Ending simulation number of the task for the direction search method.*
- unsigned int [niterations](#)
 - Number of algorithm iterations.*
- unsigned int [nbest](#)
 - Number of best simulations.*
- unsigned int [nsaveds](#)

- *Number of saved simulations.*
- unsigned int [stop](#)
To stop the simulations.
- int [mpi_rank](#)
Number of MPI task.

4.3.1 Detailed Description

Struct to define the optimization ation data.

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

4.3.2 Field Documentation

4.3.2.1 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

- GtkWidget * [dialog](#)
Main GtkDialog.
- GtkWidget * [grid](#)
Main GtkGrid.
- GtkWidget * [label_seed](#)
Pseudo-random numbers generator seed GtkWidget.
- GtkWidget * [spin_seed](#)
Pseudo-random numbers generator seed GtkWidget.
- GtkWidget * [label_threads](#)
Threads number GtkWidget.
- GtkWidget * [spin_threads](#)
Threads number GtkWidget.
- GtkWidget * [label_direction](#)
Direction threads number GtkWidget.
- GtkWidget * [spin_direction](#)
Direction threads number GtkWidget.

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

- GtkWidget * [dialog](#)
Main GtkDialog.
- GtkWidget * [label](#)
Label GtkWidget.
- GtkWidget * [spinner](#)
Animation GtkWidget.
- GtkWidget * [grid](#)
Grid GtkWidget.

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:

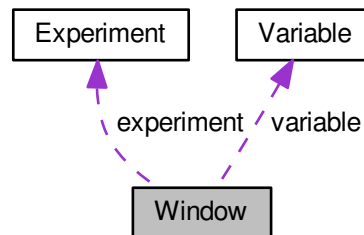
- [variable.h](#)

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](#)
GtkWidget 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.

- GtkCheckButton * [check_evaluator](#)
Evaluator program GtkCheckButton.
- GtkFileChooserButton * [button_evaluator](#)
Evaluator program GtkFileChooserButton.
- GtkLabel * [label_result](#)
Result file GtkLabel.
- GtkEntry * [entry_result](#)
Result file GtkEntry.
- GtkLabel * [label_variables](#)
Variables file GtkLabel.
- GtkEntry * [entry_variables](#)
Variables file GtkEntry.
- GtkFrame * [frame_norm](#)
GtkFrame to set the error norm.
- GtkGrid * [grid_norm](#)
GtkGrid to set the error norm.
- GtkRadioButton * [button_norm](#) [NNORMS]
Array of GtkButtons to set the error norm.
- GtkLabel * [label_p](#)
GtkLabel to set the p parameter.
- GtkSpinButton * [spin_p](#)
GtkSpinButton to set the p parameter.
- GtkScrolledWindow * [scrolled_p](#)
GtkScrolledWindow to set the p parameter.
- GtkFrame * [frame_algorithm](#)
GtkFrame to set the algorithm.
- GtkGrid * [grid_algorithm](#)
GtkGrid to set the algorithm.
- GtkRadioButton * [button_algorithm](#) [NALGORITHMS]
Array of GtkButtons to set the algorithm.
- GtkLabel * [label_simulations](#)
GtkLabel to set the simulations number.
- GtkSpinButton * [spin_simulations](#)
GtkSpinButton to set the simulations number.
- GtkLabel * [label_iterations](#)
GtkLabel to set the iterations number.
- GtkSpinButton * [spin_iterations](#)
GtkSpinButton to set the iterations number.
- GtkLabel * [label_tolerance](#)
GtkLabel to set the tolerance.
- GtkSpinButton * [spin_tolerance](#)
GtkSpinButton to set the tolerance.
- GtkLabel * [label_bests](#)
GtkLabel to set the best number.
- GtkSpinButton * [spin_bests](#)
GtkSpinButton to set the best number.
- GtkLabel * [label_population](#)
GtkLabel to set the population number.
- GtkSpinButton * [spin_population](#)
GtkSpinButton to set the population number.
- GtkLabel * [label_generations](#)

- GtkLabel to set the generations number.*
- `GtkSpinButton * spin_generations`
GtkSpinButton to set the generations number.
- `GtkLabel * label_mutation`
GtkLabel to set the mutation ratio.
- `GtkSpinButton * spin_mutation`
GtkSpinButton to set the mutation ratio.
- `GtkLabel * label_reproduction`
GtkLabel to set the reproduction ratio.
- `GtkSpinButton * spin_reproduction`
GtkSpinButton to set the reproduction ratio.
- `GtkLabel * label_adaptation`
GtkLabel to set the adaptation ratio.
- `GtkSpinButton * spin_adaptation`
GtkSpinButton to set the adaptation ratio.
- `GtkCheckButton * check_direction`
GtkCheckButton to check running the direction search method.
- `GtkGrid * grid_direction`
GtkGrid to pack the direction search method widgets.
- `GtkRadioButton * button_direction [NDIRECTIONS]`
GtkRadioButtons array to set the direction estimate method.
- `GtkLabel * label_steps`
GtkLabel to set the steps number.
- `GtkSpinButton * spin_steps`
GtkSpinButton to set the steps number.
- `GtkLabel * label_estimates`
GtkLabel to set the estimates number.
- `GtkSpinButton * spin_estimates`
GtkSpinButton to set the estimates number.
- `GtkLabel * label_relaxation`
GtkLabel to set the relaxation parameter.
- `GtkSpinButton * spin_relaxation`
GtkSpinButton to set the relaxation parameter.
- `GtkLabel * label_threshold`
GtkLabel to set the threshold.
- `GtkSpinButton * spin_threshold`
GtkSpinButton to set the threshold.
- `GtkScrolledWindow * scrolled_threshold`
GtkScrolledWindow to set the threshold.
- `GtkFrame * frame_variable`
Variable GtkFrame.
- `GtkGrid * grid_variable`
Variable GtkGrid.
- `GtkComboBoxText * combo_variable`
GtkComboBoxEntry to select a variable.
- `GtkButton * button_add_variable`
GtkButton to add a variable.
- `GtkButton * button_remove_variable`
GtkButton to remove a variable.
- `GtkLabel * label_variable`
Variable GtkLabel.

- GtkEntry * [entry_variable](#)
GtkEntry to set the variable name.
- GtkLabel * [label_min](#)
Minimum GtkLabel.
- GtkSpinButton * [spin_min](#)
Minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_min](#)
Minimum GtkScrolledWindow.
- GtkLabel * [label_max](#)
Maximum GtkLabel.
- GtkSpinButton * [spin_max](#)
Maximum GtkSpinButton.
- GtkScrolledWindow * [scrolled_max](#)
Maximum GtkScrolledWindow.
- GtkCheckButton * [check_minabs](#)
Absolute minimum GtkCheckButton.
- GtkSpinButton * [spin_minabs](#)
Absolute minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_minabs](#)
Absolute minimum GtkScrolledWindow.
- GtkCheckButton * [check_maxabs](#)
Absolute maximum GtkCheckButton.
- GtkSpinButton * [spin_maxabs](#)
Absolute maximum GtkSpinButton.
- GtkScrolledWindow * [scrolled_maxabs](#)
Absolute maximum GtkScrolledWindow.
- GtkLabel * [label_precision](#)
Precision GtkLabel.
- GtkSpinButton * [spin_precision](#)
Precision digits GtkSpinButton.
- GtkLabel * [label_sweeps](#)
Sweeps number GtkLabel.
- GtkSpinButton * [spin_sweeps](#)
Sweeps number GtkSpinButton.
- GtkLabel * [label_bits](#)
Bits number GtkLabel.
- GtkSpinButton * [spin_bits](#)
Bits number GtkSpinButton.
- GtkLabel * [label_step](#)
GtkLabel to set the step.
- GtkSpinButton * [spin_step](#)
GtkSpinButton to set the step.
- GtkScrolledWindow * [scrolled_step](#)
step GtkScrolledWindow.
- GtkFrame * [frame_experiment](#)
Experiment GtkFrame.
- GtkGrid * [grid_experiment](#)
Experiment GtkGrid.
- GtkComboBoxText * [combo_experiment](#)
Experiment GtkComboBoxEntry.
- GtkButton * [button_add_experiment](#)

- *GtkButton to add a experiment.*
- `GtkButton * button_remove_experiment`
- *GtkButton to remove a experiment.*
- `GtkLabel * label_experiment`
- *[Experiment](#) GtkLabel.*
- `GtkFileChooserButton * button_experiment`
- *GtkFileChooserButton to set the experimental data file.*
- `GtkLabel * label_weight`
- *Weight GtkLabel.*
- `GtkSpinButton * spin_weight`
- *Weight GtkSpinButton.*
- `GtkCheckButton * check_template [MAX_NINPUTS]`
- *Array of GtkCheckButtons to set the input templates.*
- `GtkFileChooserButton * button_template [MAX_NINPUTS]`
- *Array of GtkFileChooserButtons to set the input templates.*
- `GdkPixbuf * logo`
- *Logo GdkPixbuf.*
- `Experiment * experiment`
- *Array of experiments data.*
- `Variable * variable`
- *Array of variables data.*
- `char * application_directory`
- *Application directory.*
- `gulong id_experiment`
- *Identifier of the combo_experiment signal.*
- `gulong id_experiment_name`
- *Identifier of the button_experiment signal.*
- `gulong id_variable`
- *Identifier of the combo_variable signal.*
- `gulong id_variable_label`
- *Identifier of the entry_variable signal.*
- `gulong id_template [MAX_NINPUTS]`
- *Array of identifiers of the check_template signal.*
- `gulong id_input [MAX_NINPUTS]`
- *Array of identifiers of the button_template signal.*
- `unsigned int nexperiments`
- *Number of experiments.*
- `unsigned int nvariables`
- *Number of variables.*

4.8.1 Detailed Description

Struct to define the main window.

Definition at line 79 of file [interface.h](#).

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

- [interface.h](#)

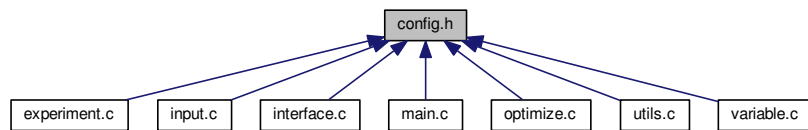
Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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



Macros

- `#define MAX_NINPUTS 8`
Maximum number of input files in the simulator program.
- `#define NALGORITHMS 3`
Number of stochastic algorithms.
- `#define NDIRECTIONS 2`
Number of direction estimate methods.
- `#define NNORMS 4`
Number of error norms.
- `#define NPRECISIONS 15`
Number of precisions.
- `#define DEFAULT_PRECISION (NPRECISIONS - 1)`
Default precision digits.
- `#define DEFAULT_RANDOM_SEED 7007`
Default pseudo-random numbers seed.
- `#define DEFAULT_RELAXATION 1.`
Default relaxation parameter.
- `#define LOCALE_DIR "locales"`

Locales directory.

- #define PROGRAM_INTERFACE "mpcotool"
Name of the interface program.
- #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
absolute minimum label.
- #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
absolute maximum label.
- #define LABEL_ADAPTATION "adaptation"
adaption label.
- #define LABEL_ALGORITHM "algorithm"
algoritm label.
- #define LABEL_OPTIMIZE "optimize"
optimize label.
- #define LABEL_COORDINATES "coordinates"
coordinates label.
- #define LABEL_DIRECTION "direction"
direction label.
- #define LABEL_EUCLIDIAN "euclidian"
euclidian label.
- #define LABEL_EVALUATOR "evaluator"
evaluator label.
- #define LABEL_EXPERIMENT "experiment"
experiment label.
- #define LABEL_EXPERIMENTS "experiments"
experiment label.
- #define LABEL_GENETIC "genetic"
genetic label.
- #define LABEL_MINIMUM "minimum"
minimum label.
- #define LABEL_MAXIMUM "maximum"
maximum label.
- #define LABEL_MONTE_CARLO "Monte-Carlo"
Monte-Carlo label.
- #define LABEL_MUTATION "mutation"
mutation label.
- #define LABEL_NAME "name"
name label.
- #define LABEL_NBEST "nbest"
nbest label.
- #define LABEL_NBITS "nbits"
nbits label.
- #define LABEL_NESTIMATES "nestimates"
nestimates label.
- #define LABEL_NGENERATIONS "ngenerations"
ngenerations label.
- #define LABEL_NITERATIONS "niterations"
niterations label.
- #define LABEL_NORM "norm"
norm label.
- #define LABEL_NPOPULATION "npopulation"
npopulation label.

- #define LABEL_NSIMULATIONS "nsimulations"
nsimulations label.
- #define LABEL_NSTEPS "nsteps"
nsteps label.
- #define LABEL_NSWEEPS "nsweeps"
nsweeps label.
- #define LABEL_P "p"
p label.
- #define LABEL_PRECISION "precision"
precision label.
- #define LABEL_RANDOM "random"
random label.
- #define LABEL_RELAXATION "relaxation"
relaxation label.
- #define LABEL_REPRODUCTION "reproduction"
reproduction label.
- #define LABEL_RESULT_FILE "result_file"
result_file label.
- #define LABEL_SIMULATOR "simulator"
simulator label.
- #define LABEL_SEED "seed"
seed label.
- #define LABEL_STEP "step"
step label.
- #define LABEL_SWEEP "sweep"
sweep label.
- #define LABEL_TAXICAB "taxicab"
taxicab label.
- #define LABEL_TEMPLATE1 "template1"
template1 label.
- #define LABEL_TEMPLATE2 "template2"
template2 label.
- #define LABEL_TEMPLATE3 "template3"
template3 label.
- #define LABEL_TEMPLATE4 "template4"
template4 label.
- #define LABEL_TEMPLATE5 "template5"
template5 label.
- #define LABEL_TEMPLATE6 "template6"
template6 label.
- #define LABEL_TEMPLATE7 "template7"
template7 label.
- #define LABEL_TEMPLATE8 "template8"
template8 label.
- #define LABEL_THRESHOLD "threshold"
threshold label.
- #define LABEL_TOLERANCE "tolerance"
tolerance label.
- #define LABEL_VARIABLE "variable"
variable label.
- #define LABEL_VARIABLES "variables"

- variables label.*
- #define `LABEL_VARIABLES_FILE` "variables_file"
variables label.
- #define `LABEL_WEIGHT` "weight"
weight label.

Enumerations

- enum `INPUT_TYPE` { `INPUT_TYPE_XML` = 0, `INPUT_TYPE_JSON` = 1 }
- Enum to define the input file types.*

5.1.1 Detailed Description

Configuration header file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.1.2 Enumeration Type Documentation

5.1.2.1 INPUT_TYPE

enum `INPUT_TYPE`

Enum to define the input file types.

Enumerator

<code>INPUT_TYPE_XML</code>	XML input file.
<code>INPUT_TYPE_JSON</code>	JSON input file.

Definition at line 125 of file [config.h](#).

```
00126 {
00127     INPUT_TYPE_XML = 0,
00128     INPUT_TYPE_JSON = 1
00129 };
```

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
00044  // Default choices
00045
00046  #define DEFAULT_PRECISION (NPRECISIONS - 1)
00047  #define DEFAULT_RANDOM_SEED 7007
00048  #define DEFAULT_RELAXATION 1.
00049
00050  // Interface labels
00051
00052  #define LOCALE_DIR "locales"
00053  #define PROGRAM_INTERFACE "mpcotool"
00054
00055  // Labels
00056
00057  #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00058  #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00059  #define LABEL_ADAPTATION "adaptation"
00060  #define LABEL_ALGORITHM "algorithm"
00061  #define LABEL_OPTIMIZE "optimize"
00062  #define LABEL_COORDINATES "coordinates"
00063  #define LABEL_DIRECTION "direction"
00064  #define LABEL_EUCLIDIAN "euclidian"
00065  #define LABEL_EVALUATOR "evaluator"
00066  #define LABEL_EXPERIMENT "experiment"
00067  #define LABEL_EXPERIMENTS "experiments"
00068  #define LABEL_GENETIC "genetic"
00069  #define LABEL_MINIMUM "minimum"
00070  #define LABEL_MAXIMUM "maximum"
00071  #define LABEL_MONTE_CARLO "Monte-Carlo"
00072  #define LABEL_MUTATION "mutation"
00073  #define LABEL_NAME "name"
00074  #define LABEL_NBEST "nbest"
00075  #define LABEL_NBITS "nbits"
00076  #define LABEL_NESTIMATES "nestimates"
00077  #define LABEL_NGENERATIONS "ngenerations"
00078  #define LABEL_NITERATIONS "niterations"
00079  #define LABEL_NORM "norm"
00080  #define LABEL_NPOPULATION "npopulation"
00081  #define LABEL_NSIMULATIONS "nsimulations"
00082  #define LABEL_NSTEPS "nsteps"
00083  #define LABEL_NSWEEPS "nsweeps"
00084  #define LABEL_P "p"
00085  #define LABEL_PRECISION "precision"
00086  #define LABEL_RANDOM "random"
00087  #define LABEL_RELAXATION "relaxation"
00088  #define LABEL_REPRODUCTION "reproduction"

```

```

00098 #define LABEL_RESULT_FILE "result_file"
00099 #define LABEL_SIMULATOR "simulator"
00100 #define LABEL_SEED "seed"
00101 #define LABEL_STEP "step"
00102 #define LABEL_SWEEP "sweep"
00103 #define LABEL_TAXICAB "taxicab"
00104 #define LABEL_TEMPLATE1 "template1"
00105 #define LABEL_TEMPLATE2 "template2"
00106 #define LABEL_TEMPLATE3 "template3"
00107 #define LABEL_TEMPLATE4 "template4"
00108 #define LABEL_TEMPLATE5 "template5"
00109 #define LABEL_TEMPLATE6 "template6"
00110 #define LABEL_TEMPLATE7 "template7"
00111 #define LABEL_TEMPLATE8 "template8"
00112 #define LABEL_THRESHOLD "threshold"
00113 #define LABEL_TOLERANCE "tolerance"
00114 #define LABEL_VARIABLE "variable"
00115 #define LABEL_VARIABLES "variables"
00116 #define LABEL_VARIABLES_FILE "variables_file"
00117 #define LABEL_WEIGHT "weight"
00118
00119 // Enumerations
00120
00125 enum INPUT_TYPE
00126 {
00127     INPUT_TYPE_XML = 0,
00128     INPUT_TYPE_JSON = 1
00129 };
00130
00131 #endif

```

5.3 experiment.c File Reference

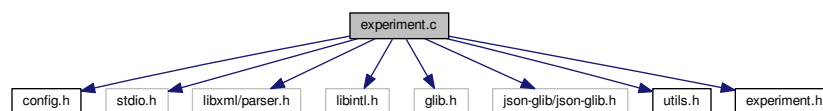
Source file to define the experiment data.

```

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

```

Include dependency graph for experiment.c:



Macros

- `#define _GNU_SOURCE`
- `#define DEBUG_EXPERIMENT 0`

Macro to debug experiment functions.

Functions

- void [experiment_new](#) ([Experiment](#) *experiment)
Function to create a new [Experiment](#) struct.
- void [experiment_free](#) ([Experiment](#) *experiment, unsigned int type)
Function to free the memory of an [Experiment](#) struct.
- void [experiment_error](#) ([Experiment](#) *experiment, char *message)
Function to print a message error opening an [Experiment](#) struct.
- int [experiment_open_xml](#) ([Experiment](#) *experiment, xmlNode *node, unsigned int ninputs)
Function to open the [Experiment](#) struct on a XML node.
- int [experiment_open_json](#) ([Experiment](#) *experiment, JsonNode *node, unsigned int ninputs)
Function to open the [Experiment](#) struct on a XML node.

Variables

- const char * [template](#) [[MAX_NINPUTS](#)]
Array of xmlChar strings with template labels.

5.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

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](#).

```

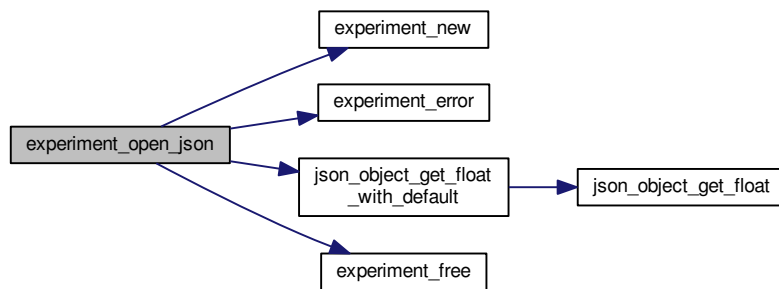
00254 {
00255     char buffer[64];
00256     JsonObject *object;
00257     const char *name;
00258     int error_code;
00259     unsigned int i;
00260
00261     #if DEBUG_EXPERIMENT
00262         fprintf (stderr, "experiment_open_json: start\n");
00263     #endif
00264
00265     // Resetting experiment 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 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
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         {
00338             break;
00339         }
00340     }
00341     #if DEBUG_EXPERIMENT
00342     fprintf (stderr, "experiment_open_json: end\n");
00343     #endif
00344     return 1;
00345 }
00346 exit_on_error:
00347     experiment_free (experiment, INPUT_TYPE_JSON);
00348     #if DEBUG_EXPERIMENT
00349     fprintf (stderr, "experiment_open_json: end\n");
00350     #endif
00351     return 0;
00352 }

```


Here is the call graph for this function:



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>	<code>Experiment</code> 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`.

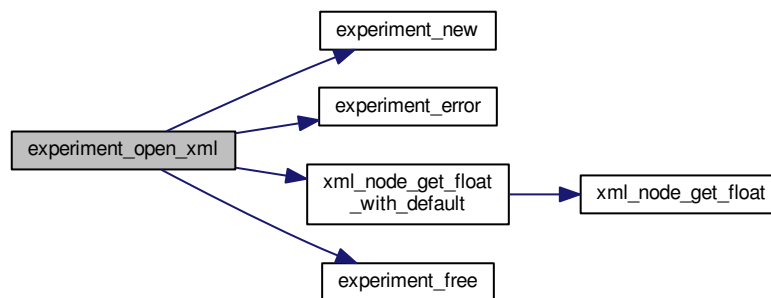
```
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // 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->name, experiment->nexperiments,
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:



5.3.3 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         g_free (experiment->name);
00085     }
00086     experiment->ninputs = 0;
00087     #if DEBUG_EXPERIMENT
00088     fprintf (stderr, "experiment_free: end\n");
00089     #endif
00090 }
00091
00092 void
00093 experiment_error (Experiment * experiment, char *message)
00094 {
00095     char buffer[64];
00096     if (!experiment->name)
00097         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00098     else
00099         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00100             message);
00101     error_message = g_strdup (buffer);
00102 }
00103
00104 int
00105 experiment_open_xml (Experiment * experiment, xmlNode * node,
00106     unsigned int ninputs)
00107 {
00108     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 }
00238
00239 int
00240 experiment_open_json (Experiment * experiment, JsonNode * node,
00241                     unsigned int ninputs)
00242 {
00243     char buffer[64];
00244     JsonObject *object;
00245     const char *name;
00246     int error_code;
00247     unsigned int i;
00248
00249 #if DEBUG_EXPERIMENT
00250     fprintf (stderr, "experiment_open_json: start\n");
00251 #endif
00252
00253     // Resetting experiment data
00254     experiment_new (experiment);
00255
00256     // Getting JSON object
00257     object = json_node_get_object (node);
00258
00259     // Reading the experimental data
00260     name = json_object_get_string_member (object, LABEL_NAME);
00261     if (!name)
00262     {
00263         experiment_error (experiment, gettext ("no data file name"));
00264         goto exit_on_error;
00265     }
00266     experiment->name = g_strdup (name);
00267 #if DEBUG_EXPERIMENT
00268     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00269 #endif
00270     experiment->weight
00271     = json_object_get_float_with_default (object,
00272     LABEL_WEIGHT, 1.,
00273     &error_code);
00274     if (error_code)
00275     {
00276         experiment_error (experiment, gettext ("bad weight"));
00277         goto exit_on_error;
00278     }
00279 #if DEBUG_EXPERIMENT
00280     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00281 #endif
00282     name = json_object_get_string_member (object, template[0]);
00283     if (name)
00284     {
00285         #if DEBUG_EXPERIMENT
00286             fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00287                     name, template[0]);
00288         #endif
00289         ++experiment->ninputs;
00290     }
00291     else
00292     {
00293         experiment_error (experiment, gettext ("no template"));
00294         goto exit_on_error;
00295     }
00296     experiment->template[0] = g_strdup (name);
00297     for (i = 1; i < MAX_NINPUTS; ++i)
00298     {
00299         #if DEBUG_EXPERIMENT
00300             fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00301         #endif
00302         if (json_object_get_member (object, template[i]))
00303         {
00304             if (ninputs && ninputs <= i)
00305             {
00306                 experiment_error (experiment, gettext ("bad templates number"));
00307                 goto exit_on_error;
00308             }
00309             name = json_object_get_string_member (object, template[i]);
00310             #if DEBUG_EXPERIMENT
00311                 fprintf (stderr,
00312                         "experiment_open_json: experiment=%s template%u=%s\n",
00313                         experiment->nexperiments, name, template[i]);
00314             #endif
00315             experiment->template[i] = g_strdup (name);
00316             ++experiment->ninputs;
00317         }
00318         else if (ninputs && ninputs > i)
00319         {
00320             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00321             experiment_error (experiment, buffer);
00322         }
00323     }

```

```

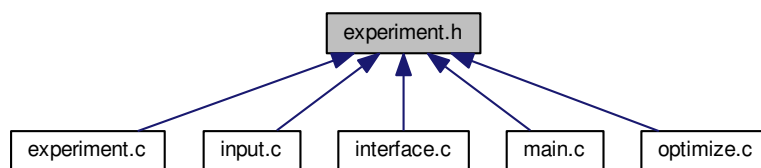
00333         goto exit_on_error;
00334     }
00335     else
00336         break;
00337 }
00338
00339 #if DEBUG_EXPERIMENT
00340 fprintf (stderr, "experiment_open_json: end\n");
00341 #endif
00342 return 1;
00343
00344 exit_on_error:
00345     experiment_free (experiment, INPUT_TYPE_JSON);
00346 #if 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, xmlDoc *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 JSON 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

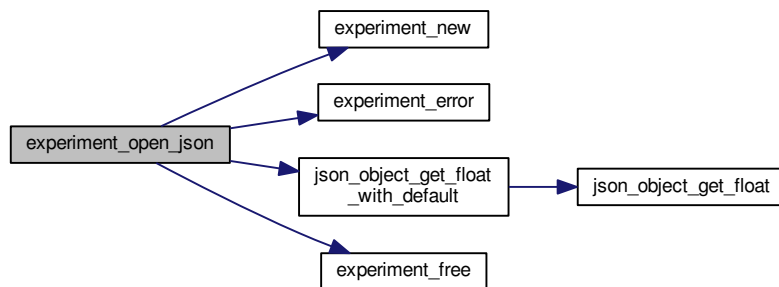
```

```

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

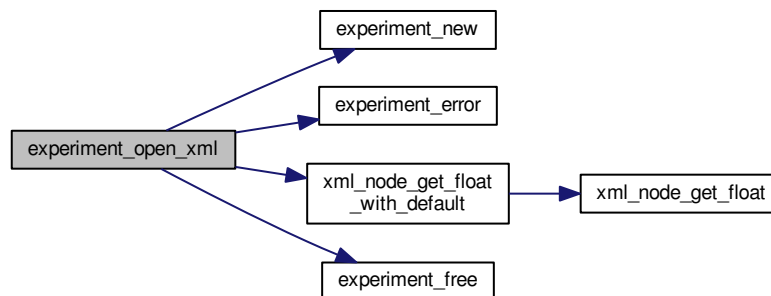
```

```

00211         experiment->nexperiments, experiment->name,
00212         experiment->template[i]);
00213 #endif
00214     ++experiment->ninputs;
00215 }
00216 else if (ninputs && ninputs > i)
00217 {
00218     snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00219     experiment_error (experiment, buffer);
00220     goto exit_on_error;
00221 }
00222 else
00223     break;
00224 }
00225
00226 #if DEBUG_EXPERIMENT
00227 fprintf (stderr, "experiment_open_xml: end\n");
00228 #endif
00229 return 1;
00230
00231 exit_on_error:
00232     experiment_free (experiment, INPUT_TYPE_XML);
00233 #if DEBUG_EXPERIMENT
00234 fprintf (stderr, "experiment_open_xml: end\n");
00235 #endif
00236     return 0;
00237 }

```

Here is the call graph for this function:



5.6 experiment.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier 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 EXPERIMENT__H
00039 #define EXPERIMENT__H 1
00040
00045 typedef struct
00046 {
00047     char *name;
00048     char *template[MAX_NINPUTS];
00049     double weight;
00050     unsigned int ninputs;
00051 } Experiment;
00052
00053 extern const char *template[MAX_NINPUTS];
00054
00055 // Public functions
00056 void experiment_new (Experiment * experiment);
00057 void experiment_free (Experiment * experiment, unsigned int type);
00058 void experiment_error (Experiment * experiment, char *message);
00059 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00060                          unsigned int ninputs);
00061 int experiment_open_json (Experiment * experiment, JsonNode * node,
00062                           unsigned int ninputs);
00063
00064 #endif

```

5.7 input.c File Reference

Source file to define the input functions.

```

#include "config.h"
#include <stdio.h>
#include <string.h>
#include <libxml/parser.h>
#include <libintl.h>
#include <glib.h>
#include <glib/gstdio.h>
#include <json-glib/json-glib.h>
#include "utils.h"
#include "experiment.h"
#include "variable.h"
#include "input.h"

```

Include dependency graph for input.c:



Macros

- `#define _GNU_SOURCE`
- `#define DEBUG_INPUT 0`

Macro to debug input functions.

Functions

- void `input_new` ()
Function to create a new `Input` struct.
- void `input_free` ()
Function to free the memory of the input file data.
- void `input_error` (char *message)
Function to print an error message opening an `Input` struct.
- int `input_open_xml` (xmlDoc *doc)
Function to open the input file in XML format.
- int `input_open_json` (JsonParser *parser)
Function to open the input file in JSON format.
- int `input_open` (char *filename)
Function to open the input file.

Variables

- `Input input` [1]
Global `Input` struct to set the input data.
- const char * `result_name` = "result"
Name of the result file.
- const char * `variables_name` = "variables"
Name of the variables file.

5.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file `input.c`.

5.7.2 Function Documentation

5.7.2.1 `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](#).

```

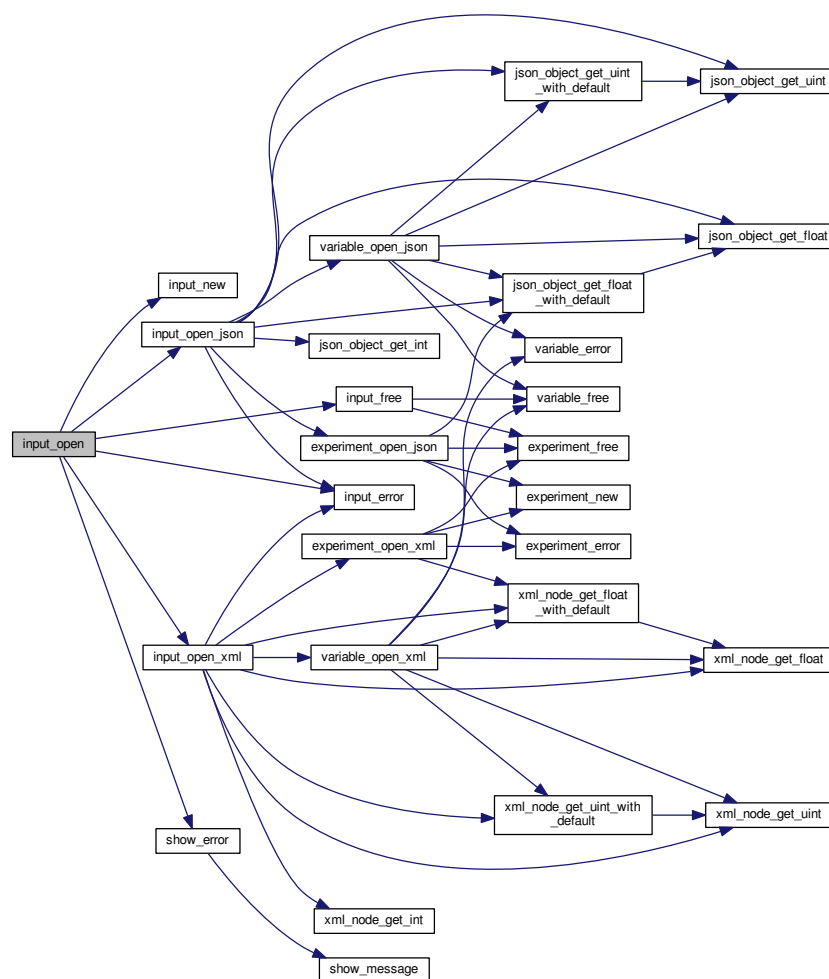
00948 {
00949     xmlDoc *doc;
00950     JsonParser *parser;
00951
00952     #if DEBUG_INPUT
00953         fprintf (stderr, "input_open: start\n");
00954     #endif
00955
00956     // Resetting input data
00957     input_new ();
00958
00959     // Opening input file
00960     #if DEBUG_INPUT
00961         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);

```

```

00626     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627     {
00628         input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630         // Obtaining simulations number
00631         input->nsimulations
00632             = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00633 );
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         else
00657         {
00658             input_error (gettext ("No population number"));
00659             goto exit_on_error;
00660         }
00661
00662         // Obtaining generations
00663         if (json_object_get_member (object, LABEL_NGENERATIONS))
00664         {
00665             input->niterations
00666                 = json_object_get_uint (object,
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         else
00674         {
00675             input_error (gettext ("No generations number"));
00676             goto exit_on_error;
00677         }
00678
00679         // Obtaining mutation probability
00680         if (json_object_get_member (object, LABEL_MUTATION))
00681         {
00682             input->mutation_ratio
00683                 = json_object_get_float (object, LABEL_MUTATION, &error_code
00684 );
00685             if (error_code || input->mutation_ratio < 0.
00686                 || input->mutation_ratio >= 1.)
00687             {
00688                 input_error (gettext ("Invalid mutation probability"));
00689                 goto exit_on_error;
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     }

```

```

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     // Checking survivals
00734     i = input->mutation_ratio * input->nsimulations;
00735     i += input->reproduction_ratio * input->
00736 nsimulations;
00737     i += input->adaptation_ratio * input->
00738 nsimulations;
00739     if (i > input->nsimulations - 2)
00740     {
00741         input_error
00742             (gettext
00743              ("No enough survival entities to reproduce the population"));
00744         goto exit_on_error;
00745     }
00746     else
00747     {
00748         input_error (gettext ("Unknown algorithm"));
00749         goto exit_on_error;
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         {

```

```

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,
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             input->nsteps = 0;
00830         // Obtaining the threshold
00831         input->threshold
00832         = json_object_get_float_with_default (object,
00833 LABEL_THRESHOLD, 0.,
00834                                     &error_code);
00835         if (error_code)
00836         {
00837             input_error (gettext ("Invalid threshold"));
00838             goto exit_on_error;
00839         }
00840         // Reading the experimental data
00841         array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00842         n = json_array_get_length (array);
00843         input->experiment = (Experiment *) g_malloc (n * sizeof (
00844 Experiment));
00845         for (i = 0; i < n; ++i)
00846         {
00847             #if DEBUG_INPUT
00848                 fprintf (stderr, "input_open_json: nexperiments=%u\n",
00849                     input->nexperiments);
00850             #endif
00851             child = json_array_get_element (array, i);
00852             if (!input->nexperiments)
00853             {
00854                 if (!experiment_open_json (input->experiment, child, 0))
00855                     goto exit_on_error;
00856             }
00857             else
00858             {
00859                 if (!experiment_open_json (input->experiment +
00860 input->nexperiments,
00861                                     child, input->experiment->
00862 ninputs))
00863                     goto exit_on_error;
00864             }
00865             ++input->nexperiments;
00866             #if DEBUG_INPUT
00867                 fprintf (stderr, "input_open_json: nexperiments=%u\n",
00868                     input->nexperiments);
00869             #endif
00870             if (!input->nexperiments)
00871             {

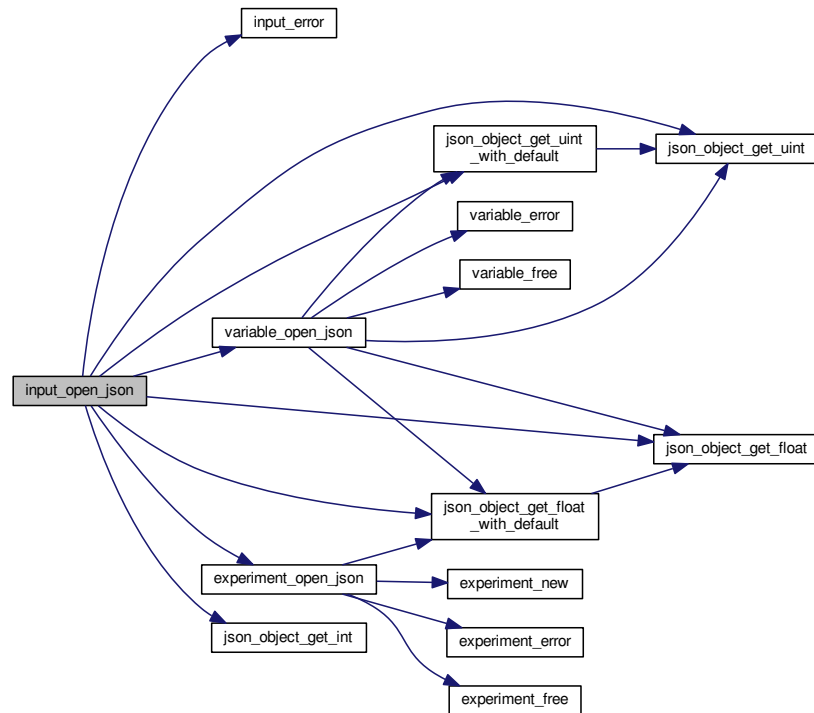
```

```

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

```

Here is the call graph for this function:



5.7.2.4 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

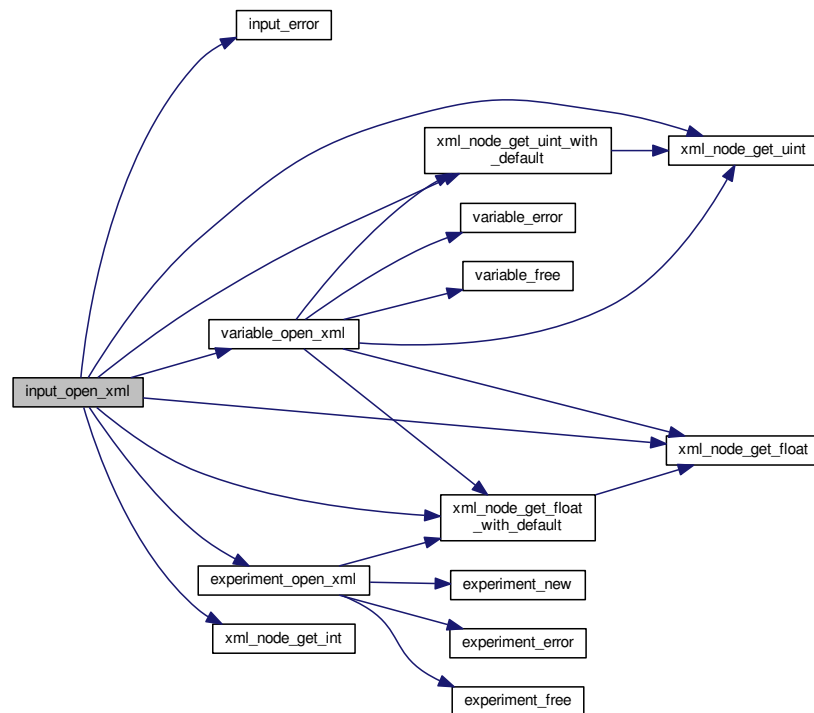
```

```

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

```

Here is the call graph for this function:



5.8 input.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>

```

```

00043 #include <libintl.h>
00044 #include <glib.h>
00045 #include <glib/gstdio.h>
00046 #include <json-glib/json-glib.h>
00047 #include "utils.h"
00048 #include "experiment.h"
00049 #include "variable.h"
00050 #include "input.h"
00051
00052 #define DEBUG_INPUT 0
00053
00054 Input input[1];
00055
00056 const char *result_name = "result";
00057 const char *variables_name = "variables";
00058
00063 void
00064 input_new ()
00065 {
00066     #if DEBUG_INPUT
00067         fprintf (stderr, "input_new: start\n");
00068     #endif
00069     input->nvariables = input->nexperiments = input->nsteps = 0;
00070     input->simulator = input->evaluator = input->directory = input->
        name = NULL;
00071     input->experiment = NULL;
00072     input->variable = NULL;
00073     #if DEBUG_INPUT
00074         fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085     unsigned int i;
00086     #if DEBUG_INPUT
00087         fprintf (stderr, "input_free: start\n");
00088     #endif
00089     g_free (input->name);
00090     g_free (input->directory);
00091     for (i = 0; i < input->nexperiments; ++i)
00092         experiment_free (input->experiment + i, input->type);
00093     for (i = 0; i < input->nvariables; ++i)
00094         variable_free (input->variable + i, input->type);
00095     g_free (input->experiment);
00096     g_free (input->variable);
00097     if (input->type == INPUT_TYPE_XML)
00098     {
00099         xmlFree (input->evaluator);
00100         xmlFree (input->simulator);
00101         xmlFree (input->result);
00102         xmlFree (input->variables);
00103     }
00104     else
00105     {
00106         g_free (input->evaluator);
00107         g_free (input->simulator);
00108         g_free (input->result);
00109         g_free (input->variables);
00110     }
00111     input->nexperiments = input->nvariables = input->nsteps = 0;
00112     #if DEBUG_INPUT
00113         fprintf (stderr, "input_free: end\n");
00114     #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
00130
00138 int
00139 input_open_xml (xmlDoc * doc)
00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif

```

```

00150
00151 // Resetting input data
00152 buffer = NULL;
00153 input->type = INPUT_TYPE_XML;
00154
00155 // Getting the root node
00156 #if DEBUG_INPUT
00157 fprintf (stderr, "input_open_xml: getting the root node\n");
00158 #endif
00159 node = xmlDocGetRootElement (doc);
00160 if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161 {
00162     input_error (gettext ("Bad root XML node"));
00163     goto exit_on_error;
00164 }
00165
00166 // Getting result and variables file names
00167 if (!input->result)
00168 {
00169     input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171     if (!input->result)
00172         input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173 }
00174 if (!input->variables)
00175 {
00176     input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178     if (!input->variables)
00179         input->variables =
00180             (char *) xmlStrdup ((const xmlChar *) variables_name);
00181 }
00182
00183 // Opening simulator program name
00184 input->simulator =
00185     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186 if (!input->simulator)
00187 {
00188     input_error (gettext ("Bad simulator program"));
00189     goto exit_on_error;
00190 }
00191
00192 // Opening evaluator program name
00193 input->evaluator =
00194     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196 // Obtaining pseudo-random numbers generator seed
00197 input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
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 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00224     input->algorithm = ALGORITHM_SWEEP;
00225 else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00226 {
00227     input->algorithm = ALGORITHM_GENETIC;
00228 }
00229 // Obtaining population
00230 if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00231 {
00232     input->nsimulations
00233         = xml_node_get_uint (node, (const xmlChar *)
00234 LABEL_NPOPULATION,
00235                             &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
00265     // Obtaining mutation probability
00266     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00267     {
00268         input->mutation_ratio
00269         = xml_node_get_float (node, (const xmlChar *)
00270 LABEL_MUTATION,
00271                             &error_code);
00272         if (error_code || input->mutation_ratio < 0.
00273             || input->mutation_ratio >= 1.)
00274         {
00275             input_error (gettext ("Invalid mutation probability"));
00276             goto exit_on_error;
00277         }
00278     }
00279     else
00280     {
00281         input_error (gettext ("No mutation probability"));
00282         goto exit_on_error;
00283     }
00284
00285     // Obtaining reproduction probability
00286     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00287     {
00288         input->reproduction_ratio
00289         = xml_node_get_float (node, (const xmlChar *)
00290 LABEL_REPRODUCTION,
00291                             &error_code);
00292         if (error_code || input->reproduction_ratio < 0.
00293             || input->reproduction_ratio >= 1.0)
00294         {
00295             input_error (gettext ("Invalid reproduction probability"));
00296             goto exit_on_error;
00297         }
00298     }
00299     else
00300     {
00301         input_error (gettext ("No reproduction probability"));
00302         goto exit_on_error;
00303     }
00304
00305     // Obtaining adaptation probability
00306     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00307     {
00308         input->adaptation_ratio
00309         = xml_node_get_float (node, (const xmlChar *)
00310 LABEL_ADAPTATION,
00311                             &error_code);
00312         if (error_code || input->adaptation_ratio < 0.
00313             || input->adaptation_ratio >= 1.)
00314         {
00315             input_error (gettext ("Invalid adaptation probability"));
00316             goto exit_on_error;
00317         }
00318     }
00319     else
00320     {

```

```

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 *)
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
00357     // Obtaining best number
00358     input->nbest
00359     = xml_node_get_uint_with_default (node, (const xmlChar *)
00360 LABEL_NBEST,
00361                                     1, &error_code);
00362     if (error_code || !input->nbest)
00363     {
00364         input_error (gettext ("Invalid best number"));
00365         goto exit_on_error;
00366     }
00367
00368     // Obtaining tolerance
00369     input->tolerance
00370     = xml_node_get_float_with_default (node,
00371                                     (const xmlChar *) LABEL_TOLERANCE,
00372                                     0., &error_code);
00373     if (error_code || input->tolerance < 0.)
00374     {
00375         input_error (gettext ("Invalid tolerance"));
00376         goto exit_on_error;
00377     }
00378
00379     // Getting direction search method parameters
00380     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00381     {
00382         input->nsteps =
00383         xml_node_get_uint (node, (const xmlChar *)
00384 LABEL_NSTEPS,
00385                           &error_code);
00386         if (error_code || !input->nsteps)
00387         {
00388             input_error (gettext ("Invalid steps number"));
00389             goto exit_on_error;
00390         }
00391         buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00392         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00393             input->direction = DIRECTION_METHOD_COORDINATES;
00394         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00395         {
00396             input->direction = DIRECTION_METHOD_RANDOM;
00397             input->nestimates
00398             = xml_node_get_uint (node, (const xmlChar *)
00399 LABEL_NESTIMATES,
00400                               &error_code);
00401             if (error_code || !input->nestimates)
00402             {

```

```

00400             input_error (gettext ("Invalid estimates number"));
00401             goto exit_on_error;
00402         }
00403     }
00404     else
00405     {
00406         input_error
00407         (gettext ("Unknown method to estimate the direction search"));
00408         goto exit_on_error;
00409     }
00410     xmlFree (buffer);
00411     buffer = NULL;
00412     input->relaxation
00413     = xml_node_get_float_with_default (node,
00414                                       (const xmlChar *)
00415                                       LABEL_RELAXATION,
00416                                       DEFAULT_RELAXATION, &error_code);
00417     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00418     {
00419         input_error (gettext ("Invalid relaxation parameter"));
00420         goto exit_on_error;
00421     }
00422 }
00423 else
00424     input->nsteps = 0;
00425 }
00426 // Obtaining the threshold
00427 input->threshold =
00428     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00429                                     0., &error_code);
00430 if (error_code)
00431 {
00432     input_error (gettext ("Invalid threshold"));
00433     goto exit_on_error;
00434 }
00435 // Reading the experimental data
00436 for (child = node->children; child; child = child->next)
00437 {
00438     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00439         break;
00440 #if DEBUG_INPUT
00441     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00442             input->nexperiments);
00443 #endif
00444     input->experiment = (Experiment *)
00445         g_realloc (input->experiment,
00446                   (1 + input->nexperiments) * sizeof (Experiment));
00447     if (!input->nexperiments)
00448     {
00449         if (!experiment_open_xml (input->experiment, child, 0))
00450             goto exit_on_error;
00451     }
00452     else
00453     {
00454         if (!experiment_open_xml (input->experiment + input->
nexperiments,
00455                                 child, input->experiment->ninputs))
00456             goto exit_on_error;
00457     }
00458     ++input->nexperiments;
00459 #if DEBUG_INPUT
00460     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00461             input->nexperiments);
00462 #endif
00463 }
00464 if (!input->nexperiments)
00465 {
00466     input_error (gettext ("No optimization experiments"));
00467     goto exit_on_error;
00468 }
00469 buffer = NULL;
00470 // Reading the variables data
00471 for (; child; child = child->next)
00472 {
00473     #if DEBUG_INPUT
00474     fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00475     #endif
00476     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00477     {
00478         snprintf (buffer2, 64, "%s %u: %s",
00479                 gettext ("Variable"),
00480                 input->nvariables + 1, gettext ("bad XML node"));
00481         input_error (buffer2);
00482     }
00483 }

```

```

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

```

```

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 // Opening algorithm
00625 buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00626 if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627 {
00628     input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630     // Obtaining simulations number
00631     input->nsimulations
00632         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00633 );
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     }
00658     else
00659     {
00660         input_error (gettext ("No population number"));
00661         goto exit_on_error;
00662     }

```

```

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);
00667         if (error_code || !input->niterations)
00668         {
00669             input_error (gettext ("Invalid generations number"));
00670             goto exit_on_error;
00671         }
00672     }
00673     else
00674     {
00675         input_error (gettext ("No generations number"));
00676         goto exit_on_error;
00677     }
00678
00679     // Obtaining mutation probability
00680     if (json_object_get_member (object, LABEL_MUTATION))
00681     {
00682         input->mutation_ratio
00683         = json_object_get_float (object, LABEL_MUTATION, &error_code
00684 );
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);
00702         if (error_code || input->reproduction_ratio < 0.
00703             || input->reproduction_ratio >= 1.0)
00704         {
00705             input_error (gettext ("Invalid reproduction probability"));
00706             goto exit_on_error;
00707         }
00708     }
00709     else
00710     {
00711         input_error (gettext ("No reproduction probability"));
00712         goto exit_on_error;
00713     }
00714
00715     // Obtaining adaptation probability
00716     if (json_object_get_member (object, LABEL_ADAPTATION))
00717     {
00718         input->adaptation_ratio
00719         = json_object_get_float (object,
00720 LABEL_ADAPTATION, &error_code);
00720         if (error_code || input->adaptation_ratio < 0.
00721             || input->adaptation_ratio >= 1.)
00722         {
00723             input_error (gettext ("Invalid adaptation probability"));
00724             goto exit_on_error;
00725         }
00726     }
00727     else
00728     {
00729         input_error (gettext ("No adaptation probability"));
00730         goto exit_on_error;
00731     }
00732
00733     // Checking survivals
00734     i = input->mutation_ratio * input->nsimulations;
00735     i += input->reproduction_ratio * input->nsimulations;
00736     i += input->adaptation_ratio * input->nsimulations;
00737     if (i > input->nsimulations - 2)
00738     {
00739         input_error
00740         (gettext
00741          ("No enough survival entities to reproduce the population"));
00742         goto exit_on_error;

```

```

00743     }
00744 }
00745 else
00746 {
00747     input_error (gettext ("Unknown algorithm"));
00748     goto exit_on_error;
00749 }
00750
00751 if (input->algorithm == ALGORITHM_MONTE_CARLO
00752 || input->algorithm == ALGORITHM_SWEEP)
00753 {
00754
00755     // Obtaining iterations number
00756     input->niterations
00757     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00758     if (error_code == 1)
00759         input->niterations = 1;
00760     else if (error_code)
00761     {
00762         input_error (gettext ("Bad iterations number"));
00763         goto exit_on_error;
00764     }
00765
00766     // Obtaining best number
00767     input->nbest
00768     = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
00769                                         &error_code);
00770     if (error_code || !input->nbest)
00771     {
00772         input_error (gettext ("Invalid best number"));
00773         goto exit_on_error;
00774     }
00775
00776     // Obtaining tolerance
00777     input->tolerance
00778     = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
00779                                         &error_code);
00780     if (error_code || input->tolerance < 0.)
00781     {
00782         input_error (gettext ("Invalid tolerance"));
00783         goto exit_on_error;
00784     }
00785
00786     // Getting direction search method parameters
00787     if (json_object_get_member (object, LABEL_NSTEPS))
00788     {
00789         input->nsteps
00790         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00791         if (error_code || !input->nsteps)
00792         {
00793             input_error (gettext ("Invalid steps number"));
00794             goto exit_on_error;
00795         }
00796         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00797         if (!strcmp (buffer, LABEL_COORDINATES))
00798             input->direction = DIRECTION_METHOD_COORDINATES;
00799         else if (!strcmp (buffer, LABEL_RANDOM))
00800         {
00801             input->direction = DIRECTION_METHOD_RANDOM;
00802             input->nestimates
00803             = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00804             if (error_code || !input->nestimates)
00805             {
00806                 input_error (gettext ("Invalid estimates number"));
00807                 goto exit_on_error;
00808             }
00809         }
00810         else
00811         {
00812             input_error
00813             (gettext ("Unknown method to estimate the direction search"));
00814             goto exit_on_error;
00815         }
00816         input->relaxation
00817         = json_object_get_float_with_default (object,
LABEL_RELAXATION,
00818                                             DEFAULT_RELAXATION,
00819                                             &error_code);
00820         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00821         {
00822             input_error (gettext ("Invalid relaxation parameter"));
00823             goto exit_on_error;

```

```

00824     }
00825     }
00826     else
00827         input->nsteps = 0;
00828     }
00829     // Obtaining the threshold
00830     input->threshold
00831     = json_object_get_float_with_default (object,
00832     LABEL_THRESHOLD, 0.,
00833     &error_code);
00834     if (error_code)
00835     {
00836         input_error (gettext ("Invalid threshold"));
00837         goto exit_on_error;
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 (
00843     Experiment));
00844     for (i = 0; i < n; ++i)
00845     {
00846         #if DEBUG_INPUT
00847         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00848         input->nexperiments);
00849         #endif
00850         child = json_array_get_element (array, i);
00851         if (!input->nexperiments)
00852         {
00853             if (!experiment_open_json (input->experiment, child, 0))
00854                 goto exit_on_error;
00855         }
00856         else
00857         {
00858             if (!experiment_open_json (input->experiment + input->
00859             nexperiments,
00860             child, input->experiment->ninputs))
00861                 goto exit_on_error;
00862             ++input->nexperiments;
00863         }
00864         #if DEBUG_INPUT
00865         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00866         input->nexperiments);
00867         #endif
00868         if (!input->nexperiments)
00869         {
00870             input_error (gettext ("No optimization experiments"));
00871             goto exit_on_error;
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->
00884             nvariables, child,
00885             input->algorithm, input->nsteps))
00886                 goto exit_on_error;
00887             ++input->nvariables;
00888         }
00889         if (!input->nvariables)
00890         {
00891             input_error (gettext ("No optimization variables"));
00892             goto exit_on_error;
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);

```



```

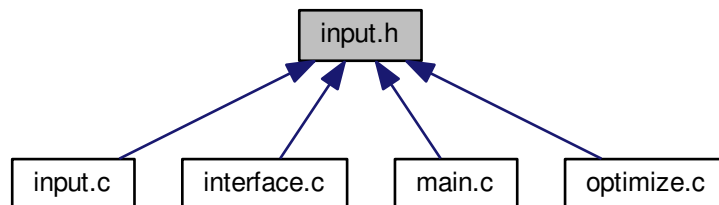
00906         if (!error_code)
00907         {
00908             input_error (gettext ("Bad P parameter"));
00909             goto exit_on_error;
00910         }
00911     }
00912     else if (!strcmp (buffer, LABEL_TAXICAB))
00913         input->norm = ERROR_NORM_TAXICAB;
00914     else
00915     {
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 [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.
-----------------	---------------------------------------

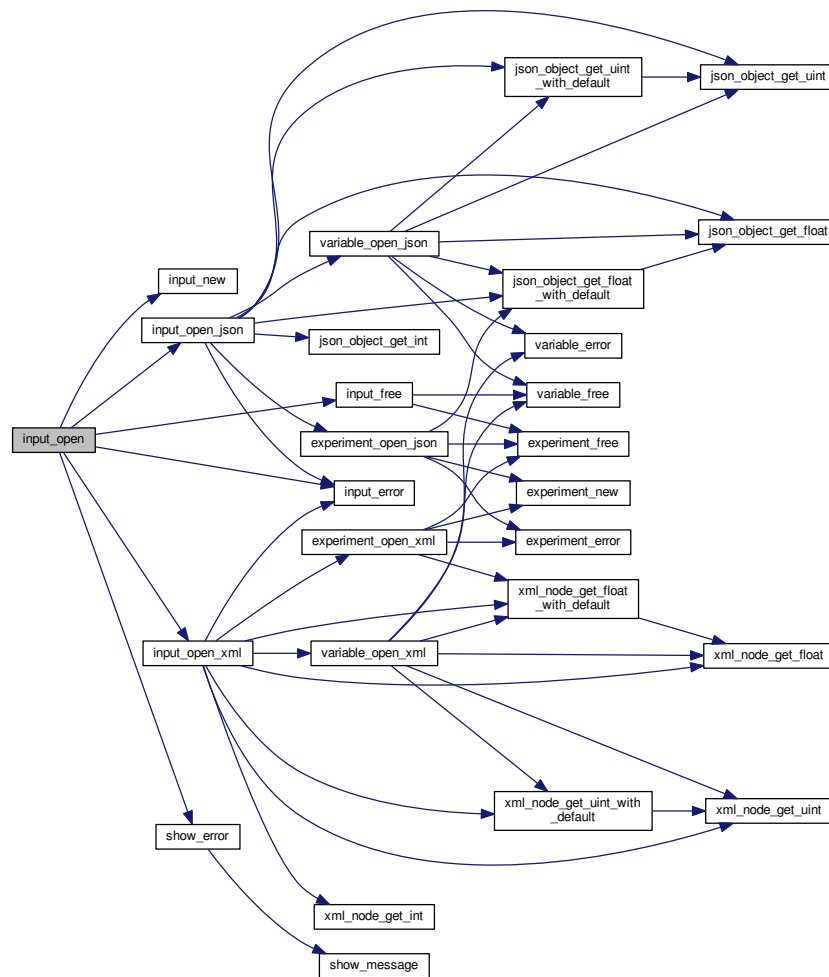
Returns

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
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 );
00635         if (error_code)
00636         {
00637             input_error (gettext ("Bad simulations number"));
00638             goto exit_on_error;
00639         }
00640     }
00641     else if (!strcmp (buffer, LABEL_SWEEP))
00642         input->algorithm = ALGORITHM_SWEEP;
00643     else if (!strcmp (buffer, LABEL_GENETIC))
00644         {

```

```

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     {
00659         input_error (gettext ("No population number"));
00660         goto exit_on_error;
00661     }
00662
00663     // Obtaining generations
00664     if (json_object_get_member (object, LABEL_NGENERATIONS))
00665     {
00666         input->niterations
00667         = json_object_get_uint (object,
00668 LABEL_NGENERATIONS, &error_code);
00669         if (error_code || !input->niterations)
00670         {
00671             input_error (gettext ("Invalid generations number"));
00672             goto exit_on_error;
00673         }
00674     }
00675     else
00676     {
00677         input_error (gettext ("No generations number"));
00678         goto exit_on_error;
00679     }
00680
00681     // Obtaining mutation probability
00682     if (json_object_get_member (object, LABEL_MUTATION))
00683     {
00684         input->mutation_ratio
00685         = json_object_get_float (object, LABEL_MUTATION, &error_code
00686 );
00687         if (error_code || input->mutation_ratio < 0.
00688             || input->mutation_ratio >= 1.)
00689         {
00690             input_error (gettext ("Invalid mutation probability"));
00691             goto exit_on_error;
00692         }
00693     }
00694     else
00695     {
00696         input_error (gettext ("No mutation probability"));
00697         goto exit_on_error;
00698     }
00699
00700     // Obtaining reproduction probability
00701     if (json_object_get_member (object, LABEL_REPRODUCTION))
00702     {
00703         input->reproduction_ratio
00704         = json_object_get_float (object,
00705 LABEL_REPRODUCTION, &error_code);
00706         if (error_code || input->reproduction_ratio < 0.
00707             || input->reproduction_ratio >= 1.0)
00708         {
00709             input_error (gettext ("Invalid reproduction probability"));
00710             goto exit_on_error;
00711         }
00712     }
00713     else
00714     {
00715         input_error (gettext ("No reproduction probability"));
00716         goto exit_on_error;
00717     }
00718
00719     // Obtaining adaptation probability
00720     if (json_object_get_member (object, LABEL_ADAPTATION))
00721     {
00722         input->adaptation_ratio
00723         = json_object_get_float (object,
00724 LABEL_ADAPTATION, &error_code);
00725         if (error_code || input->adaptation_ratio < 0.
00726             || input->adaptation_ratio >= 1.)
00727         {
00728             input_error (gettext ("Invalid adaptation probability"));
00729             goto exit_on_error;
00730         }
00731     }
00732     else
00733     {
00734         input_error (gettext ("No adaptation probability"));
00735         goto exit_on_error;
00736     }

```



```

00725         }
00726     }
00727     else
00728     {
00729         input_error (gettext ("No adaptation probability"));
00730         goto exit_on_error;
00731     }
00732
00733     // Checking survivals
00734     i = input->mutation_ratio * input->nsimulations;
00735     i += input->reproduction_ratio * input->
nsimulations;
00736     i += input->adaptation_ratio * input->
nsimulations;
00737     if (i > input->nsimulations - 2)
00738     {
00739         input_error
00740             (gettext
00741              ("No enough survival entities to reproduce the population"));
00742         goto exit_on_error;
00743     }
00744 }
00745 else
00746 {
00747     input_error (gettext ("Unknown algorithm"));
00748     goto exit_on_error;
00749 }
00750
00751 if (input->algorithm == ALGORITHM_MONTE_CARLO
00752     || input->algorithm == ALGORITHM_SWEEP)
00753 {
00754     // 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
00766     // Obtaining best number
00767     input->nbest
00768         = json_object_get_uint_with_default (object,
00769 LABEL_NBEST, 1,
00770                                             &error_code);
00771     if (error_code || !input->nbest)
00772     {
00773         input_error (gettext ("Invalid best number"));
00774         goto exit_on_error;
00775     }
00776
00777     // Obtaining tolerance
00778     input->tolerance
00779         = json_object_get_float_with_default (object,
00780 LABEL_TOLERANCE, 0.,
00781                                             &error_code);
00782     if (error_code || input->tolerance < 0.)
00783     {
00784         input_error (gettext ("Invalid tolerance"));
00785         goto exit_on_error;
00786     }
00787
00788     // Getting direction search method parameters
00789     if (json_object_get_member (object, LABEL_NSTEPS))
00790     {
00791         input->nsteps
00792             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00793         if (error_code || !input->nsteps)
00794         {
00795             input_error (gettext ("Invalid steps number"));
00796             goto exit_on_error;
00797         }
00798         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00799         if (!strcmp (buffer, LABEL_COORDINATES))
00800             input->direction = DIRECTION_METHOD_COORDINATES;
00801         else if (!strcmp (buffer, LABEL_RANDOM))
00802             input->direction = DIRECTION_METHOD_RANDOM;
00803         input->nestimates
00804             = json_object_get_uint (object,
00805 LABEL_NESTIMATES, &error_code);
00806         if (error_code || !input->nestimates)
00807         {

```

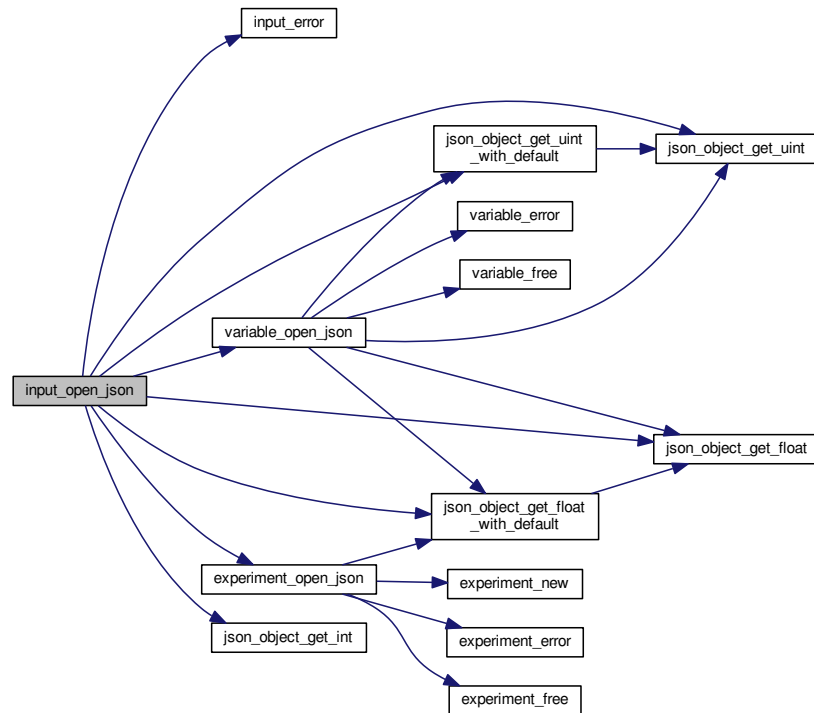
```

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,
00818     LABEL_RELAXATION,
00819     DEFAULT_RELAXATION,
00820     &error_code);
00821     if (error_code || input->relaxation < 0. || input->
00822     relaxation > 2.)
00823     {
00824         input_error (gettext ("Invalid relaxation parameter"));
00825         goto exit_on_error;
00826     }
00827     else
00828     {
00829         input->nsteps = 0;
00830     }
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     // Reading the experimental data
00842     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00843     n = json_array_get_length (array);
00844     input->experiment = (Experiment *) g_malloc (n * sizeof (
00845     Experiment));
00846     for (i = 0; i < n; ++i)
00847     {
00848         #if DEBUG_INPUT
00849         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00850         input->nexperiments);
00851         #endif
00852         child = json_array_get_element (array, i);
00853         if (!input->nexperiments)
00854         {
00855             if (!experiment_open_json (input->experiment, child, 0))
00856                 goto exit_on_error;
00857         }
00858         else
00859         {
00860             if (!experiment_open_json (input->experiment +
00861             input->nexperiments,
00862             child, input->experiment->
00863             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);
00879     n = json_array_get_length (array);
00880     input->variable = (Variable *) g_malloc (n * sizeof (
00881     Variable));
00882     for (i = 0; i < n; ++i)
00883     {
00884         #if DEBUG_INPUT
00885         fprintf (stderr, "input_open_json: nvariables=%u\n", input->
00886         nvariables);
00887         #endif
00888         child = json_array_get_element (array, i);
00889         if (!variable_open_json (input->variable +
00890         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 *)
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

```

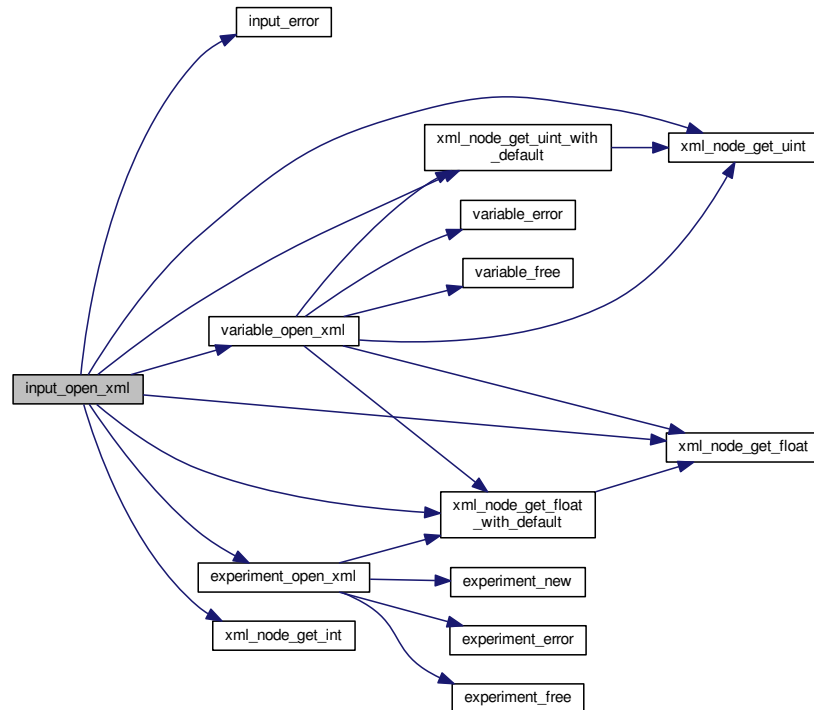


```

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

```

Here is the call graph for this function:



5.10 input.h

```

00001  /*
00002  MPCOTool:
00003  The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004  calibrations or optimizations of empirical parameters.
00005
00006  AUTHORS: Javier Burguete and Borja Latorre.
00007
00008  Copyright 2012-2016, AUTHORS.
00009
00010  Redistribution and use in source and binary forms, with or without modification,
00011  are permitted provided that the following conditions are met:
00012
00013      1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016      2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020  THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021  WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023  SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026  BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028  IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029  OF SUCH DAMAGE.
00030  */
00031
00032  #ifndef INPUT__H
00033  #define INPUT__H 1
00034
00035  enum DirectionMethod
00036  {

```

```

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](#).

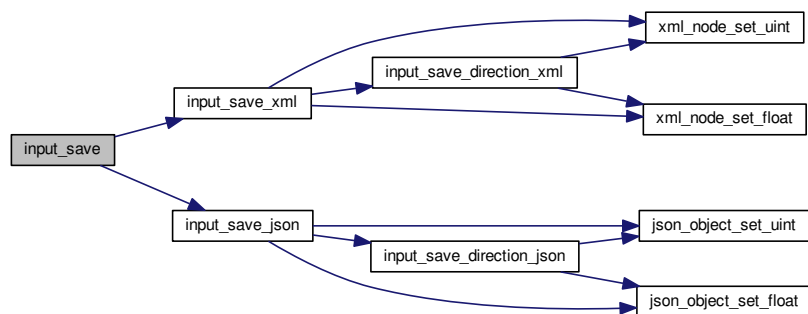
```
00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579
00580     #if DEBUG_INTERFACE
00581     fprintf (stderr, "input_save: start\n");
00582     #endif
00583
00584     // Getting the input file directory
00585     input->name = g_path_get_basename (filename);
00586     input->directory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

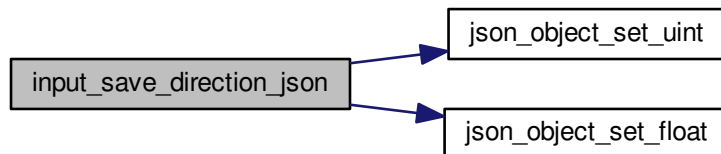
```

```

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          json_object_set_uint (object, LABEL_NESTIMATES,
00231          input->nestimates);
00232      }
00233  }
00234  #if DEBUG_INTERFACE
00235  fprintf (stderr, "input_save_direction_json: end\n");
00236  #endif

```

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)

```

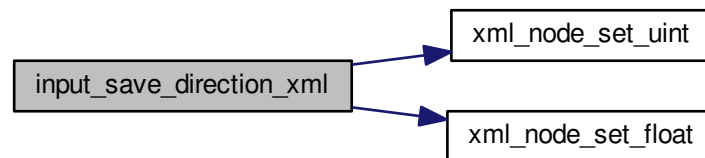


```

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);
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         #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);
00484         snprintf (buffer, 64, "%u", input->niterations);
00485         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00486         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00487         json_object_set_string_member (object, LABEL_MUTATION_RATIO, buffer);
00488         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00489         json_object_set_string_member (object, LABEL_REPRODUCTION_RATIO, buffer);
00490         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00491         json_object_set_string_member (object, LABEL_ADAPTATION_RATIO, buffer);
00492         break;
00493     }
00494     g_slice_free1 (64, buffer);
00495     if (input->threshold != 0.)
00496         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00497
00498     // Setting the experimental data
00499     array = json_array_new ();
00500     for (i = 0; i < input->nexperiments; ++i)
00501     {
00502         child = json_node_new (JSON_NODE_OBJECT);

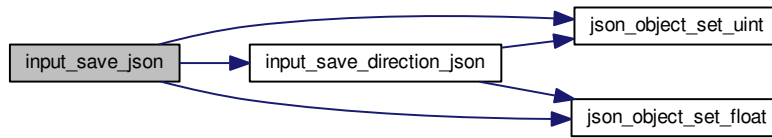
```

```

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].
template[j]);
00513     json_array_add_element (array, child);
00514 }
00515 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00516
00517 // Setting the variables data
00518 array = json_array_new ();
00519 for (i = 0; i < input->nvariables; ++i)
00520 {
00521     child = json_node_new (JSON_NODE_OBJECT);
00522     object = json_node_get_object (child);
00523     json_object_set_string_member (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,
LABEL_ABSOLUTE_MINIMUM,
                                input->variable[i].rangeminabs);
00529     json_object_set_float (object2, LABEL_MAXIMUM,
00530                            input->variable[i].rangemax);
00531     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00532         json_object_set_float (object2,
LABEL_ABSOLUTE_MAXIMUM,
                                input->variable[i].rangemaxabs);
00533     if (input->variable[i].precision !=
DEFAULT_PRECISION)
00534         json_object_set_uint (object2, LABEL_PRECISION,
                                input->variable[i].precision);
00535     if (input->algorithm == ALGORITHM_SWEEP)
00536         json_object_set_uint (object2, LABEL_NSWEEPS,
                                input->variable[i].nsweeps);
00537     else if (input->algorithm == ALGORITHM_GENETIC)
00538         json_object_set_uint (object2, LABEL_NBITS,
                                input->variable[i].nbits);
00539     if (input->nsteps)
00540         json_object_set_float (object, LABEL_STEP,
                                input->variable[i].step);
00541     json_array_add_element (array, child);
00542 }
00543 json_object_set_array_member (object, LABEL_VARIABLES, array);
00544
00545 // Saving the error norm
00546 switch (input->norm)
00547 {
00548     case ERROR_NORM_MAXIMUM:
00549         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00550         break;
00551     case ERROR_NORM_P:
00552         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00553         json_object_set_float (object, LABEL_P, input->
p);
00554         break;
00555     case ERROR_NORM_TAXICAB:
00556         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00557 }
00558
00559 #if DEBUG_INTERFACE
00560 fprintf (stderr, "input_save_json: end\n");
00561 #endif
00562 }

```

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)
```

```

00283     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
00284     input->seed);
00285
00286     // Setting the algorithm
00287     buffer = (char *) g_slice_alloc (64);
00288     switch (input->algorithm)
00289     {
00290     case ALGORITHM_MONTE_CARLO:
00291         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00292         (const xmlChar *) LABEL_MONTE_CARLO);
00293         snprintf (buffer, 64, "%u", input->nsimulations);
00294         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00295         (xmlChar *) buffer);
00296         snprintf (buffer, 64, "%u", input->niterations);
00297         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00298         (xmlChar *) buffer);
00299         snprintf (buffer, 64, "%.3lg", input->tolerance);
00300         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00301         snprintf (buffer, 64, "%u", input->nbest);
00302         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00303         input_save_direction_xml (node);
00304         break;
00305     case ALGORITHM_SWEEP:
00306         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00307         (const xmlChar *) LABEL_SWEEP);
00308         snprintf (buffer, 64, "%u", input->niterations);
00309         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00310         (xmlChar *) buffer);
00311         snprintf (buffer, 64, "%.3lg", input->tolerance);
00312         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00313         snprintf (buffer, 64, "%u", input->nbest);
00314         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00315         input_save_direction_xml (node);
00316         break;
00317     default:
00318         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00319         (const xmlChar *) LABEL_GENETIC);
00320         snprintf (buffer, 64, "%u", input->nsimulations);
00321         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00322         (xmlChar *) buffer);
00323         snprintf (buffer, 64, "%u", input->niterations);
00324         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00325         (xmlChar *) buffer);
00326         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00327         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00328         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00329         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00330         (xmlChar *) buffer);
00331         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00332         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00333         break;
00334     }
00335     g_slice_free1 (64, buffer);
00336     if (input->threshold != 0.)
00337         xml_node_set_float (node, (const xmlChar *)
00338         LABEL_THRESHOLD,
00339         input->threshold);
00340
00341     // Setting the experimental data
00342     for (i = 0; i < input->nexperiments; ++i)
00343     {
00344         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00345         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00346         (xmlChar *) input->experiment[i].name);
00347         if (input->experiment[i].weight != 1.)
00348             xml_node_set_float (child, (const xmlChar *)
00349             LABEL_WEIGHT,
00350             input->experiment[i].weight);
00351         for (j = 0; j < input->experiment->ninputs; ++j)
00352             xmlSetProp (child, (const xmlChar *) template[j],
00353             (xmlChar *) input->experiment[i].template[j]);
00354     }
00355
00356     // Setting the variables data
00357     for (i = 0; i < input->nvariables; ++i)
00358     {
00359         child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00360         xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00361         (xmlChar *) input->variable[i].name);
00362         xml_node_set_float (child, (const xmlChar *)
00363         LABEL_MINIMUM,
00364         input->variable[i].rangemin);
00365         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00366             xml_node_set_float (child, (const xmlChar *)
00367             LABEL_ABSOLUTE_MINIMUM,
00368             input->variable[i].rangeminabs);
00369         xml_node_set_float (child, (const xmlChar *)

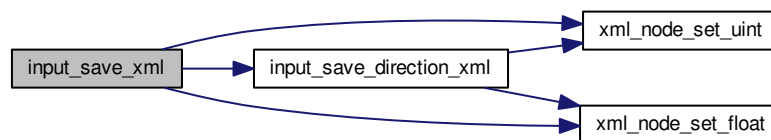
```

```

    LABEL_MAXIMUM,
00365         input->variable[i].rangemax);
00366     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00367         xml_node_set_float (child, (const xmlChar *)
    LABEL_ABSOLUTE_MAXIMUM,
00368         input->variable[i].rangemaxabs);
00369     if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00370         xml_node_set_uint (child, (const xmlChar *)
    LABEL_PRECISION,
00371         input->variable[i].precision);
00372     if (input->algorithm == ALGORITHM_SWEEP)
00373         xml_node_set_uint (child, (const xmlChar *)
    LABEL_NSWEEPS,
00374         input->variable[i].nsweeps);
00375     else if (input->algorithm == ALGORITHM_GENETIC)
00376         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00377         input->variable[i].nbits);
00378     if (input->nsteps)
00379         xml_node_set_float (child, (const xmlChar *)
    LABEL_STEP,
00380         input->variable[i].step);
00381 }
00382
00383 // Saving the error norm
00384 switch (input->norm)
00385 {
00386     case ERROR_NORM_MAXIMUM:
00387         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00388             (const xmlChar *) LABEL_MAXIMUM);
00389         break;
00390     case ERROR_NORM_P:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392             (const xmlChar *) LABEL_P);
00393         xml_node_set_float (node, (const xmlChar *) LABEL_P,
    input->p);
00394         break;
00395     case ERROR_NORM_TAXICAB:
00396         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397             (const xmlChar *) LABEL_TAXICAB);
00398 }
00399
00400 #if DEBUG_INTERFACE
00401     fprintf (stderr, "input_save: end\n");
00402 #endif
00403 }

```

Here is the call graph for this function:



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     #endif
00737     return i;
00738 }
```

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     #endif
00757     return i;
00758 }
```

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);
00773     #if DEBUG_INTERFACE
00774     fprintf (stderr, "window_get_norm: %u\n", i);
00775     fprintf (stderr, "window_get_norm: end\n");
00776     #endif
00777     return i;
00778 }
```

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
01911         case ALGORITHM_SWEEP:
01912             gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01913             gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01914             gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01915             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_direction),
input->nsteps);
01916             if (input->nsteps)
01917             {
01918                 gtk_toggle_button_set_active
(GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01919                 gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01920                 gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01921                 switch (input->direction)
01922                 {
01923                     case DIRECTION_METHOD_RANDOM:
01924                         gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01925
01926                 }
01927             }
01928     }
01929 }

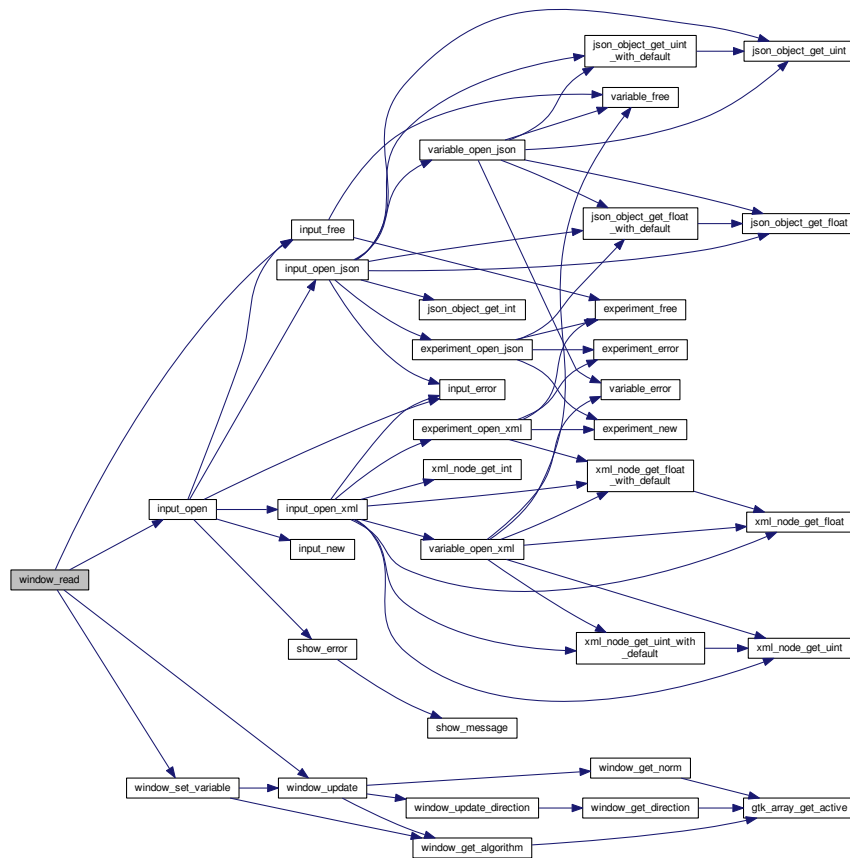
```

```

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->
mutation_ratio);
01945         gtk_spin_button_set_value (window->spin_reproduction,
01946                                   input->reproduction_ratio);
01947         gtk_spin_button_set_value (window->spin_adaptation,
01948                                   input->adaptation_ratio);
01949     }
01950     gtk_toggle_button_set_active
01951     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01952     gtk_spin_button_set_value (window->spin_p, input->p);
01953     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01954     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01955     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,
01964                             window->id_experiment);
01965     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01966     g_signal_handler_block (window->combo_variable, window->
id_variable);
01967     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01968     gtk_combo_box_text_remove_all (window->combo_variable);
01969     for (i = 0; i < input->nvariables; ++i)
01970         gtk_combo_box_text_append_text (window->combo_variable,
01971                                         input->variable[i].name);
01972     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01973     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01974     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01975     window_set_variable ();
01976     window_update ();
01977     #if DEBUG_INTERFACE
01978     fprintf (stderr, "window_read: end\n");
01979     #endif
01980     return 1;
01981 }

```

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     // Setting the algorithm
00901     switch (window_get_algorithm ())
00902     {
00903     case ALGORITHM_MONTE_CARLO:
00904         input->algorithm = ALGORITHM_MONTE_CARLO;
00905         input->nsimulations
00906             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00907         input->niterations
00908             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00909         input->tolerance = gtk_spin_button_get_value (window->
00910 spin_tolerance);
00911         input->nbest = gtk_spin_button_get_value_as_int (window->
00912 spin_bests);
00913         window_save_direction ();
00914         break;
00915     case ALGORITHM_SWEEP:
00916         input->algorithm = ALGORITHM_SWEEP;
00917         input->niterations
00918             = gtk_spin_button_get_value_as_int (window->spin_iterations);

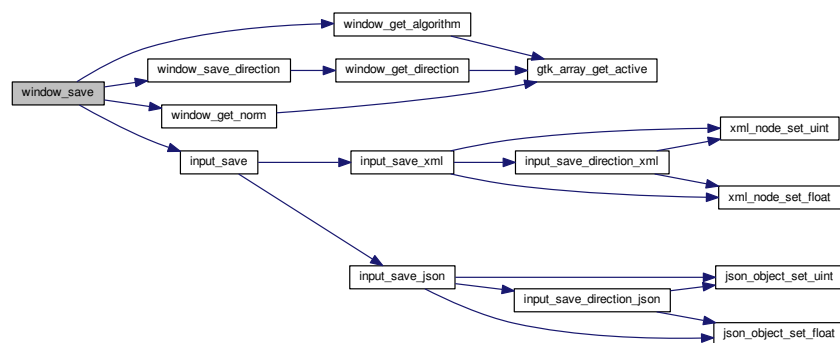
```

```

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
= gtk_spin_button_get_value_as_int (window->spin_population);
00923     input->niterations
= gtk_spin_button_get_value_as_int (window->spin_generations);
00924     input->mutation_ratio
= gtk_spin_button_get_value (window->spin_mutation);
00925     input->reproduction_ratio
= gtk_spin_button_get_value (window->spin_reproduction);
00926     input->adaptation_ratio
= gtk_spin_button_get_value (window->spin_adaptation);
00927     break;
00928     }
00929     input->norm = window_get_norm ();
00930     input->p = gtk_spin_button_get_value (window->spin_p);
00931     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00932
00933     // Saving the XML file
00934     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00935     input_save (buffer);
00936
00937     // Closing and freeing memory
00938     g_free (buffer);
00939     gtk_widget_destroy (GTK_WIDGET (dlg));
00940 #if DEBUG_INTERFACE
00941     fprintf (stderr, "window_save: end\n");
00942 #endif
00943     return 1;
00944 }
00945
00946 // Closing and freeing memory
00947 gtk_widget_destroy (GTK_WIDGET (dlg));
00948 #if DEBUG_INTERFACE
00949     fprintf (stderr, "window_save: end\n");
00950 #endif
00951     return 0;
00952 }

```

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>
```

```

00050 #ifdef G_OS_WIN32
00051 #include <windows.h>
00052 #endif
00053 #if HAVE_MPI
00054 #include <mpi.h>
00055 #endif
00056 #include <gio/gio.h>
00057 #include <gtk/gtk.h>
00058 #include "genetic/genetic.h"
00059 #include "utils.h"
00060 #include "experiment.h"
00061 #include "variable.h"
00062 #include "input.h"
00063 #include "optimize.h"
00064 #include "interface.h"
00065
00066 #define DEBUG_INTERFACE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define INPUT_FILE "test-ga-win.xml"
00074 #else
00075 #define INPUT_FILE "test-ga.xml"
00076 #endif
00077
00078 const char *logo[] = {
00079     "32 32 3 1",
00080     "    c None",
00081     ".    c #0000FF",
00082     "+    c #FF0000",
00083     "                                ",
00084     "                                ",
00085     "                                ",
00086     "    .    .    .    .    ",
00087     "    .    .    .    .    ",
00088     "    .    .    .    .    ",
00089     "    .    .    .    .    ",
00090     "    .    .    +++    .    ",
00091     "    .    .    +++++    .    ",
00092     "    .    .    +++++    .    ",
00093     "    .    .    +++++    .    ",
00094     "    +++    .    +++    +++    ",
00095     "    +++++    .    .    +++++    ",
00096     "    +++++    .    .    +++++    ",
00097     "    +++++    .    .    +++++    ",
00098     "    +++    .    .    +++    ",
00099     "    .    .    .    .    ",
00100     "    .    +++    .    .    ",
00101     "    .    +++++    .    .    ",
00102     "    .    +++++    .    .    ",
00103     "    .    +++++    .    .    ",
00104     "    .    +++    .    .    ",
00105     "    .    .    .    .    ",
00106     "    .    .    .    .    ",
00107     "    .    .    .    .    ",
00108     "    .    .    .    .    ",
00109     "    .    .    .    .    ",
00110     "    .    .    .    .    ",
00111     "    .    .    .    .    ",
00112     "                                ",
00113     "                                ",
00114     "                                ",
00115 };
00116
00117 /*
00118 const char * logo[] = {
00119     "32 32 3 1",
00120     "    c #FFFFFFFF",
00121     ".    c #00000000",
00122     "X    c #FFF0000000",
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     }
00230     #if DEBUG_INTERFACE
00231         fprintf (stderr, "input_save_direction_json: end\n");
00232     #endif
00233 }
00234

```



```

00241 void
00242 input_save_xml (xmlDoc * doc)
00243 {
00244     unsigned int i, j;
00245     char *buffer;
00246     xmlNode *node, *child;
00247     GFile *file, *file2;
00248
00249 #if DEBUG_INTERFACE
00250     fprintf (stderr, "input_save_xml: start\n");
00251 #endif
00252
00253     // Setting root XML node
00254     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00255     xmlDocSetRootElement (doc, node);
00256
00257     // Adding properties to the root XML node
00258     if (xmlStrcmp
00259         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00260         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00261                     (xmlChar *) input->result);
00262     if (xmlStrcmp
00263         ((const xmlChar *) input->variables, (const xmlChar *)
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,
00296                     (xmlChar *) buffer);
00297         snprintf (buffer, 64, "%u", input->niterations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                     (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%.3lg", input->tolerance);
00301         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_direction_xml (node);
00305         break;
00306     case ALGORITHM_SWEEP:
00307         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                     (const xmlChar *) LABEL_SWEEP);
00309         snprintf (buffer, 64, "%u", input->niterations);
00310         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                     (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%.3lg", input->tolerance);
00313         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00314         snprintf (buffer, 64, "%u", input->nbest);
00315         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00316         input_save_direction_xml (node);
00317         break;
00318     default:
00319         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00320                     (const xmlChar *) LABEL_GENETIC);
00321         snprintf (buffer, 64, "%u", input->nsimulations);
00322         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00323                     (xmlChar *) buffer);
00324         snprintf (buffer, 64, "%u", input->niterations);
00325         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00326                     (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);

```

```

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)
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 // Saving the error norm
00391 switch (input->norm)
00392 {
00393     case ERROR_NORM_MAXIMUM:
00394         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00395                     (const xmlChar *) LABEL_MAXIMUM);
00396         break;
00397     case ERROR_NORM_P:
00398         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00399                     (const xmlChar *) LABEL_P);
00400         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00401                             input->p);
00402         break;
00403     case ERROR_NORM_TAXICAB:
00404         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00405                     (const xmlChar *) LABEL_TAXICAB);
00406         break;
00407 }
00408 #if DEBUG_INTERFACE
00409 fprintf (stderr, "input_save: end\n");

```

```

00402 #endif
00403 }
00404
00411 void
00412 input_save_json (JsonGenerator * generator)
00413 {
00414     unsigned int i, j;
00415     char *buffer;
00416     JsonNode *node, *child;
00417     JsonObject *object, *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     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

```

```

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
00554 fprintf (stderr, "input_save_json: end\n");
00555 #endif
00556 }
00557
00558 void
00559 input_save (char *filename)

```

```

00576 {
00577     xmlDoc *doc;
00578     JsonGenerator *generator;
00579
00580 #if DEBUG_INTERFACE
00581     fprintf (stderr, "input_save: start\n");
00582 #endif
00583
00584     // Getting the input file directory
00585     input->name = g_path_get_basename (filename);
00586     input->directory = g_path_get_dirname (filename);
00587
00588     if (input->type == INPUT_TYPE_XML)
00589     {
00590         // Opening the input file
00591         doc = xmlNewDoc ((const xmlChar *) "1.0");
00592         input_save_xml (doc);
00593
00594         // Saving the XML file
00595         xmlSaveFormatFile (filename, doc, 1);
00596
00597         // Freeing memory
00598         xmlFreeDoc (doc);
00599     }
00600     else
00601     {
00602         // Opening the input file
00603         generator = json_generator_new ();
00604         json_generator_set_pretty (generator, TRUE);
00605         input_save_json (generator);
00606
00607         // Saving the JSON file
00608         json_generator_to_file (generator, filename, NULL);
00609
00610         // Freeing memory
00611         g_object_unref (generator);
00612     }
00613
00614 #if DEBUG_INTERFACE
00615     fprintf (stderr, "input_save: end\n");
00616 #endif
00617 }
00618
00623 void
00624 options_new ()
00625 {
00626 #if DEBUG_INTERFACE
00627     fprintf (stderr, "options_new: start\n");
00628 #endif
00629     options->label_seed = (GtkLabel *)
00630         gtk_label_new (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;

```

```

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

```



```

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 }
00958
00959 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
01016 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),
input->nexperiments > 1);
01202 gtk_widget_set_sensitive
01203 (GTK_WIDGET (window->button_remove_variable), input->
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->
id_input[i]);
01213     gtk_toggle_button_set_active
01214 (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
01215     g_signal_handler_unblock
01216 (window->button_template[i], window->id_input[i]);
01217     g_signal_handler_unblock
01218 (window->check_template[i], window->id_template[i]);
01219 }
01220 if (i > 0)

```

```

01221     {
01222         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01223         gtk_widget_set_sensitive
01224             (GTK_WIDGET (window->button_template[i - 1]),
01225              gtk_toggle_button_get_active
01226                  GTK_TOGGLE_BUTTON (window->check_template[i - 1]));
01227     }
01228     if (i < MAX_NINPUTS)
01229     {
01230         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01231         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01232         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     window_update ();
01296     #if DEBUG_INTERFACE
01297     fprintf (stderr, "window_set_algorithm: end\n");
01298     #endif
01299 }
01300
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
01331 gtk_file_chooser_set_filename
01332 (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01333 g_free (buffer2);
01334 g_signal_handler_unblock
01335 (window->button_template[j], window->id_input[j]);
01336 }
01337 #if DEBUG_INTERFACE
01338 fprintf (stderr, "window_set_experiment: end\n");
01339 #endif
01340 }
01341
01342 void
01343 window_remove_experiment ()
01344 {
01345 unsigned int i, j;
01346 #if DEBUG_INTERFACE
01347 fprintf (stderr, "window_remove_experiment: start\n");
01348 #endif
01349 i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01350 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01351 gtk_combo_box_text_remove (window->combo_experiment, i);
01352 g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01353 experiment_free (input->experiment + i, input->
type);
01354 --input->nexperiments;
01355 for (j = i; j < input->nexperiments; ++j)
01356 memcpy (input->experiment + j, input->experiment + j + 1,
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
(input->experiment, (input->nexperiments + 1) * sizeof (
Experiment));

```

```

01398     for (j = input->nexperiments - 1; j > i; --j)
01399         memcpy (input->experiment + j + 1, input->experiment + j,
01400             sizeof (Experiment));
01401     input->experiment[j + 1].weight = input->experiment[j].
weight;
01402     input->experiment[j + 1].ninputs = input->
experiment[j].ninputs;
01403     if (input->type == INPUT_TYPE_XML)
01404     {
01405         input->experiment[j + 1].name
01406             = (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
01434 void
01435 window_name_experiment ()
01436 {
01437     unsigned int i;
01438     char *buffer;
01439     GFile *file1, *file2;
01440     #if DEBUG_INTERFACE
01441     fprintf (stderr, "window_name_experiment: start\n");
01442     #endif
01443     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01444     file1
01445         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01446     file2 = g_file_new_for_path (input->directory);
01447     buffer = g_file_get_relative_path (file2, file1);
01448     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01449     gtk_combo_box_text_remove (window->combo_experiment, i);
01450     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01451     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01452     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01453     g_free (buffer);
01454     g_object_unref (file2);
01455     g_object_unref (file1);
01456     #if DEBUG_INTERFACE
01457     fprintf (stderr, "window_name_experiment: end\n");
01458     #endif
01459 }
01460
01461 void
01462 window_weight_experiment ()
01463 {
01464     unsigned int i;
01465     #if DEBUG_INTERFACE
01466     fprintf (stderr, "window_weight_experiment: start\n");
01467     #endif
01468     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01469     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01470     #if DEBUG_INTERFACE
01471     fprintf (stderr, "window_weight_experiment: end\n");
01472     #endif
01473 }
01474
01475
01476

```

```

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 }

```

```

01581     else
01582     {
01583         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01584         gtk_toggle_button_set_active
01585             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01586     }
01587     gtk_spin_button_set_value (window->spin_precision,
01588                             input->variable[i].precision);
01589     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
01590     if (input->nsteps)
01591         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].
nsweeps);
01601         #if DEBUG_INTERFACE
01602             fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01603                 input->variable[i].nsweeps);
01604         #endif
01605         break;
01606         case ALGORITHM_GENETIC:
01607             gtk_spin_button_set_value (window->spin_bits,
01608                                     (gdouble) input->variable[i].nbits);
01609         #if DEBUG_INTERFACE
01610             fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01611                 input->variable[i].nbits);
01612         #endif
01613         break;
01614     }
01615     window_update ();
01616     #if DEBUG_INTERFACE
01617         fprintf (stderr, "window_set_variable: end\n");
01618     #endif
01619 }
01620
01625 void
01626 window_remove_variable ()
01627 {
01628     unsigned int i, j;
01629     #if DEBUG_INTERFACE
01630         fprintf (stderr, "window_remove_variable: start\n");
01631     #endif
01632     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01633     g_signal_handler_block (window->combo_variable, window->
id_variable);
01634     gtk_combo_box_text_remove (window->combo_variable, i);
01635     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01636     xmlFree (input->variable[i].name);
01637     --input->nvariables;
01638     for (j = i; j < input->nvariables; ++j)
01639         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01640     j = input->nvariables - 1;
01641     if (i > j)
01642         i = j;
01643     g_signal_handler_block (window->entry_variable, window->
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->
id_variable_label);
01646     window_update ();
01647     #if DEBUG_INTERFACE
01648         fprintf (stderr, "window_remove_variable: end\n");
01649     #endif
01650 }
01651
01656 void
01657 window_add_variable ()
01658 {
01659     unsigned int i, j;
01660     #if DEBUG_INTERFACE
01661         fprintf (stderr, "window_add_variable: start\n");
01662     #endif
01663     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01664     g_signal_handler_block (window->combo_variable, window->
id_variable);
01665     gtk_combo_box_text_insert_text (window->combo_variable, i,
01666                                     input->variable[i].name);

```



```

01667 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01668 input->variable = (Variable *) g_realloc
01669 (input->variable, (input->nvariables + 1) * sizeof (
Variable));
01670 for (j = input->nvariables - 1; j > i; --j)
01671 memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01672 memcpy (input->variable + j + 1, input->variable + j, sizeof (
Variable));
01673 if (input->type == INPUT_TYPE_XML)
01674 input->variable[j + 1].name
01675 = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01676 else
01677 input->variable[j + 1].name = g_strdup (input->
variable[j].name);
01678 ++input->nvariables;
01679 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01680 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01681 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01682 window_update ();
01683 #if DEBUG_INTERFACE
01684 fprintf (stderr, "window_add_variable: end\n");
01685 #endif
01686 }
01687
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 }
01736
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
01777 void
01778 window_rangeminabs_variable ()
01779 {
01780     unsigned int i;
01781     #if DEBUG_INTERFACE
01782         fprintf (stderr, "window_rangeminabs_variable: start\n");
01783     #endif
01784     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01785     input->variable[i].rangeminabs
        = gtk_spin_button_get_value (window->spin_minabs);
01786     #if DEBUG_INTERFACE
01787         fprintf (stderr, "window_rangeminabs_variable: end\n");
01788     #endif
01789 }
01790
01791
01796 void
01797 window_rangemaxabs_variable ()
01798 {
01799     unsigned int i;
01800     #if DEBUG_INTERFACE
01801         fprintf (stderr, "window_rangemaxabs_variable: start\n");
01802     #endif
01803     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01804     input->variable[i].rangemaxabs
        = gtk_spin_button_get_value (window->spin_maxabs);
01805     #if DEBUG_INTERFACE
01806         fprintf (stderr, "window_rangemaxabs_variable: end\n");
01807     #endif
01808 }
01809
01810
01815 void
01816 window_step_variable ()
01817 {
01818     unsigned int i;
01819     #if DEBUG_INTERFACE
01820         fprintf (stderr, "window_step_variable: start\n");
01821     #endif
01822     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01823     input->variable[i].step = gtk_spin_button_get_value (window->
        spin_step);
01824     #if DEBUG_INTERFACE
01825         fprintf (stderr, "window_step_variable: end\n");
01826     #endif
01827 }
01828
01833 void
01834 window_update_variable ()
01835 {
01836     int i;
01837     #if DEBUG_INTERFACE
01838         fprintf (stderr, "window_update_variable: start\n");
01839     #endif
01840     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01841     if (i < 0)
01842         i = 0;
01843     switch (window_get_algorithm ())
01844     {
01845         case ALGORITHM_SWEEP:
01846             input->variable[i].nsweeps
                = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01847             #if DEBUG_INTERFACE
01848                 fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
                    input->variable[i].nsweeps);
01849             #endif
01850             break;
01851         case ALGORITHM_GENETIC:
01852             input->variable[i].nbits
                = gtk_spin_button_get_value_as_int (window->spin_bits);
01853             #if DEBUG_INTERFACE
01854                 fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
                    input->variable[i].nbits);
01855             #endif
01856         }
01857 }
01858

```

```

01859 #endif
01860 }
01861 #if DEBUG_INTERFACE
01862 fprintf (stderr, "window_update_variable: end\n");
01863 #endif
01864 }
01865
01866 int
01873 window_read (char *filename)
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_best, (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         break;
01927     default:
01928         gtk_spin_button_set_value (window->spin_population,
(gdouble) input->nsimulations);
01929         gtk_spin_button_set_value (window->spin_generations,
(gdouble) input->niterations);
01930         gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01931         gtk_spin_button_set_value (window->spin_reproduction,

```

```

01946         input->reproduction_ratio);
01947         gtk_spin_button_set_value (window->spin_adaptation,
01948         input->adaptation_ratio);
01949     }
01950     gtk_toggle_button_set_active
01951     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01952     gtk_spin_button_set_value (window->spin_p, input->p);
01953     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01954     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01955     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->
id_experiment);
01964     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01965     g_signal_handler_block (window->combo_variable, window->
id_variable);
01966     g_signal_handler_block (window->entry_variable, window->
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);
01971     g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01972     g_signal_handler_unblock (window->combo_variable, window->
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
01983 void
01984 window_open ()
01985 {
01986     GtkFileChooserDialog *dlg;
01987     GtkFileFilter *filter;
01988     char *buffer, *directory, *name;
01989
01990 #if DEBUG_INTERFACE
01991     fprintf (stderr, "window_open: start\n");
01992 #endif
01993
01994     // Saving a backup of the current input file
01995     directory = g_strdup (input->directory);
01996     name = g_strdup (input->name);
01997
01998     // Opening dialog
01999     dlg = (GtkFileChooserDialog *)
02000         gtk_file_chooser_dialog_new (gettext ("Open input file"),
02001         window->window,
02002         GTK_FILE_CHOOSER_ACTION_OPEN,
02003         gettext ("_Cancel"), GTK_RESPONSE_CANCEL,
02004         gettext ("_OK"), GTK_RESPONSE_OK, NULL);
02005
02006     // Adding XML filter
02007     filter = (GtkFileFilter *) gtk_file_filter_new ();
02008     gtk_file_filter_set_name (filter, "XML");
02009     gtk_file_filter_add_pattern (filter, "*.xml");
02010     gtk_file_filter_add_pattern (filter, "*.XML");
02011     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02012
02013     // Adding JSON filter
02014     filter = (GtkFileFilter *) gtk_file_filter_new ();
02015     gtk_file_filter_set_name (filter, "JSON");
02016     gtk_file_filter_add_pattern (filter, "*.json");
02017     gtk_file_filter_add_pattern (filter, "*.JSON");
02018     gtk_file_filter_add_pattern (filter, "*.js");
02019     gtk_file_filter_add_pattern (filter, "*.JS");
02020     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02021
02022     // If OK saving
02023     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02024     {
02025
02026

```

```

02030         // Traying to open the input file
02031         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02032         if (!window_read (buffer))
02033         {
02034             #if DEBUG_INTERFACE
02035                 fprintf (stderr, "window_open: error reading input file\n");
02036             #endif
02037             g_free (buffer);
02038
02039             // Reading backup file on error
02040             buffer = g_build_filename (directory, name, NULL);
02041             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[NNORMS] = { "L2", "L", "Lp", "L1" };
02094     char *tip_norm[NNORMS] = {
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"));
02119     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 (*)(
window_save,
02128                               NULL));
02129
02130 // Creating the run button
02131 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_hexpand (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",
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->
02219 label_simulator),
02220                  0, 0, 1, 1);
02221 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02222 button_simulator),
02223                  1, 0, 1, 1);
02224 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02225 check_evaluator),
02226                  0, 1, 1, 1);
02227 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02228 button_evaluator),
02229                  1, 1, 1, 1);
02230 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02231 label_result),
02232                  0, 2, 1, 1);
02233 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02234 entry_result),
02235                  1, 2, 1, 1);
02236 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02237 label_variables),
02238                  0, 3, 1, 1);
02239 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02240 entry_variables),
02241                  1, 3, 1, 1);
02242
02243 // Creating the algorithm properties
02244 window->label_simulations = (GtkLabel *) gtk_label_new
02245 (gettext ("Simulations number"));
02246 window->spin_simulations
02247 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02248 gtk_widget_set_tooltip_text
02249 (GTK_WIDGET (window->spin_simulations),
02250  gettext ("Number of simulations to perform for each iteration"));
02251 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02252 window->label_iterations = (GtkLabel *)
02253 gtk_label_new (gettext ("Iterations number"));
02254 window->spin_iterations
02255 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02256 gtk_widget_set_tooltip_text
02257 (GTK_WIDGET (window->spin_iterations),
02258  gettext ("Number of iterations"));
02259 g_signal_connect
02260 (window->spin_iterations, "value-changed", window_update, NULL);
02261 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02262 window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
02263 window->spin_tolerance
02264 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02265 gtk_widget_set_tooltip_text
02266 (GTK_WIDGET (window->spin_tolerance),
02267  gettext ("Tolerance to set the variable interval on the next iteration"));
02268 window->label_bests = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
02269 window->spin_bests
02270 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02271 gtk_widget_set_tooltip_text
02272 (GTK_WIDGET (window->spin_bests),
02273  gettext ("Number of best simulations used to set the variable interval "
02274           "on the next iteration"));
02275 window->label_population
02276 = (GtkLabel *) gtk_label_new (gettext ("Population number"));
02277 window->spin_population
02278 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02279 gtk_widget_set_tooltip_text
02280 (GTK_WIDGET (window->spin_population),
02281  gettext ("Number of population for the genetic algorithm"));
02282 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02283 window->label_generations
02284 = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
02285 window->spin_generations
02286 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02287 gtk_widget_set_tooltip_text
02288 (GTK_WIDGET (window->spin_generations),
02289  gettext ("Number of generations for the genetic algorithm"));
02290 window->label_mutation
02291 = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
02292 window->spin_mutation
02293 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);

```

```

02285 gtk_widget_set_tooltip_text
02286     (GTK_WIDGET (window->spin_mutation),
02287      gettext ("Ratio of mutation for the genetic algorithm"));
02288 window->label_reproduction
02289     = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
02290 window->spin_reproduction
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02292 gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_reproduction),
02294      gettext ("Ratio of reproduction for the genetic algorithm"));
02295 window->label_adaptation
02296     = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
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);

```



```

02365
02366 // Creating the array of algorithms
02367 window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02368 window->button_algorithm[0] = (GtkRadioButton *)
02369     gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02370 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02371     tip_algorithm[0]);
02372 gtk_grid_attach (window->grid_algorithm,
02373     GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02374 g_signal_connect (window->button_algorithm[0], "clicked",
02375     window_set_algorithm, NULL);
02376 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->
02447     label_threshold),
02448     0, NALGORITHMS + 11, 1, 1);
02449 gtk_grid_attach (window->grid_algorithm,
02450     GTK_WIDGET (window->scrolled_threshold), 1,
    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,
NULL);
02658 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02659 gtk_grid_attach (window->grid_experiment,
02660 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02661 gtk_grid_attach (window->grid_experiment,
02662 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02663 gtk_grid_attach (window->grid_experiment,
02664 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02665 gtk_grid_attach (window->grid_experiment,
02666 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02667 gtk_grid_attach (window->grid_experiment,
02668 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02669 gtk_grid_attach (window->grid_experiment,
02670 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02671 gtk_grid_attach (window->grid_experiment,
02672 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02673 for (i = 0; i < MAX_NINPUTS; ++i)
02674 {
02675     snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02676     window->check_template[i] = (GtkCheckButton *)
02677     gtk_check_button_new_with_label (buffer3);
02678     window->id_template[i]
02679     = g_signal_connect (window->check_template[i], "toggled",
02680     window_inputs_experiment, NULL);
02681     gtk_grid_attach (window->grid_experiment,
02682     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02683     window->button_template[i] = (GtkFileChooserButton *)
02684     gtk_file_chooser_button_new (gettext ("Input template"),
02685     GTK_FILE_CHOOSER_ACTION_OPEN);
02686     gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->button_template[i]),
02688     gettext ("Experimental input template file"));
02689     window->id_input[i]
02690     = g_signal_connect_swapped (window->button_template[i],
02691     "selection-changed",
02692     (void (*)(void *)) window_template_experiment,
02693     (void *) (size_t) i);
02694     gtk_grid_attach (window->grid_experiment,
02695     GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02696 }
02697 window->frame_experiment
02698 = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02699 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02700 GTK_WIDGET (window->grid_experiment));
02701
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));

```

```

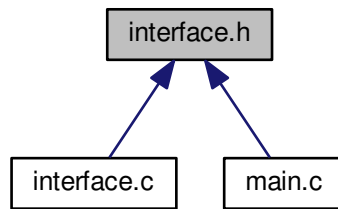
02707 window->button_norm[0] = (GtkRadioButton *)
02708     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02709 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02710     tip_norm[0]);
02711 gtk_grid_attach (window->grid_norm,
02712     GTK_WIDGET (window->button_norm[0]), 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",
02724         window_update, NULL);
02725 }
02726 window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02727 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02728 window->spin_p = (GtkSpinButton *)
02729     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02730 gtk_widget_set_tooltip_text
02731     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02732 window->scrolled_p
02733     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02734 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02735     GTK_WIDGET (window->spin_p));
02736 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02737 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02738 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02739     1, 2, 1, 2);
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->
02752     grid));
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 // Showing the window
02757 gtk_widget_show_all (GTK_WIDGET (window->window));
02758 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02759 #if GTK_MINOR_VERSION >= 16
02760 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02761 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02762 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02763 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02764 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02765 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02766 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02767 #endif
02768 // Reading initial example
02769 input_new ();
02770 buffer2 = g_get_current_dir ();
02771 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02772 g_free (buffer2);
02773 window_read (buffer);
02774 g_free (buffer);
02775 #if DEBUG_INTERFACE
02776 fprintf (stderr, "window_new: start\n");
02777 #endif
02778 }

```

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.
-----------------	------------------

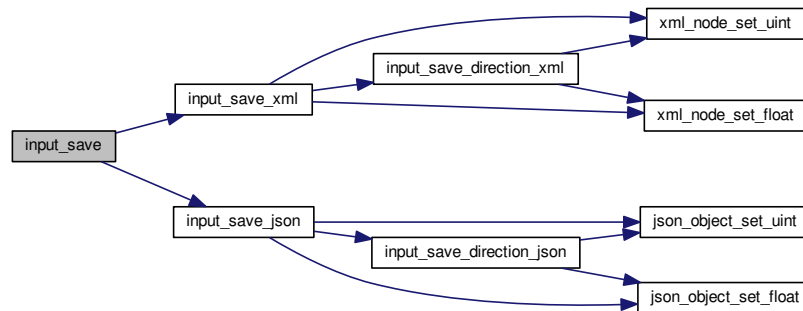
Definition at line 575 of file [interface.c](#).

```

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);
00733     #if DEBUG_INTERFACE
00734         fprintf (stderr, "window_get_algorithm: %u\n", i);
00735         fprintf (stderr, "window_get_algorithm: end\n");
00736     #endif
00737     return i;
00738 }
  
```

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);
01900
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     }
  
```

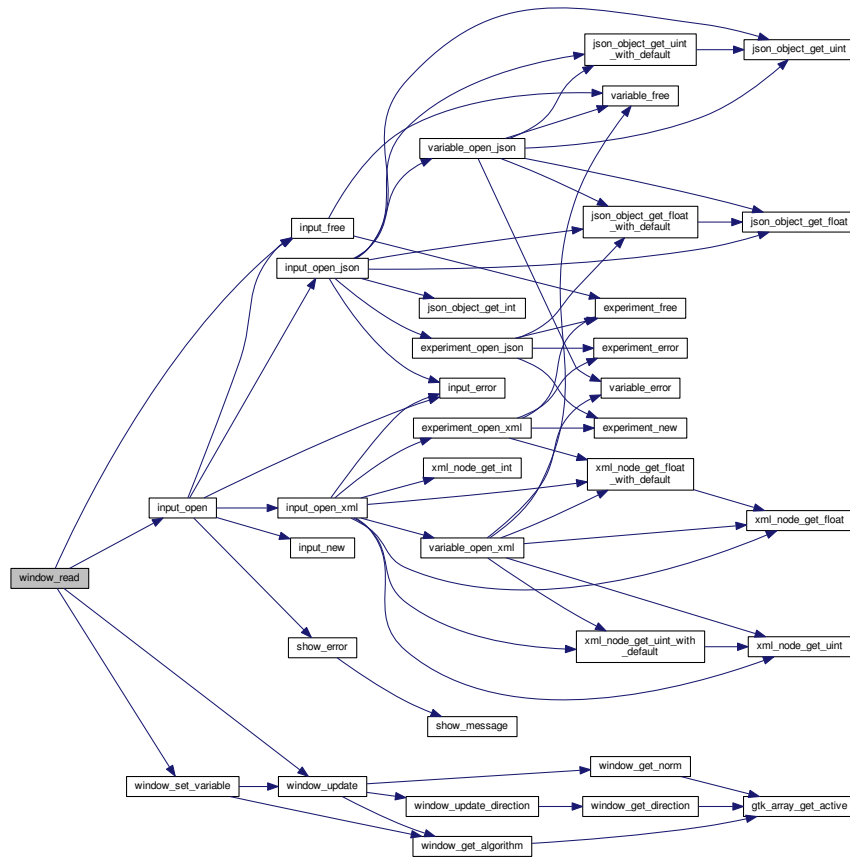
```

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

```

```
01981 }
```

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
00907             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00908         input->niterations
00909             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00910         input->tolerance = gtk_spin_button_get_value (window->
00911         spin_tolerance);
00912         input->nbest = gtk_spin_button_get_value_as_int (window->
00913         spin_bests);
00914         window_save_direction ();
00915         break;

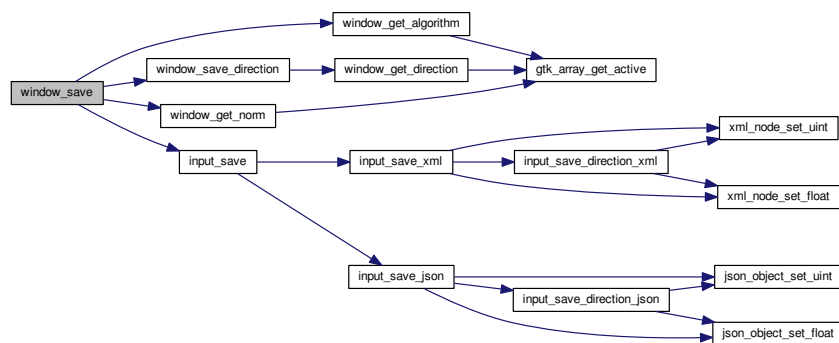
```

```

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
00041 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00042
00048 typedef struct
00049 {
00050     GtkDialog *dialog;
00051     GtkGrid *grid;
00052     GtkLabel *label_seed;
00054     GtkSpinButton *spin_seed;
00056     GtkLabel *label_threads;
```

```
00057     GtkSpinButton *spin_threads;
00058     GtkLabel *label_direction;
00059     GtkSpinButton *spin_direction;
00061 } Options;
00062
00067 typedef struct
00068 {
00069     GtkWidget *dialog;
00070     GtkLabel *label;
00071     GtkSpinner *spinner;
00072     GtkGrid *grid;
00073 } 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   GtkWidget *spin_minabs;
00173   GtkWidget *scrolled_minabs;
00174   GtkWidget *check_maxabs;
00175   GtkWidget *spin_maxabs;
00176   GtkWidget *scrolled_maxabs;
00177   GtkWidget *label_precision;
00178   GtkWidget *spin_precision;
00179   GtkWidget *label_sweeps;
00180   GtkWidget *spin_sweeps;
00181   GtkWidget *label_bits;
00182   GtkWidget *spin_bits;
00183   GtkWidget *label_step;
00184   GtkWidget *spin_step;
00185   GtkWidget *scrolled_step;
00186   GtkWidget *frame_experiment;
00187   GtkWidget *grid_experiment;
00188   GtkWidget *combo_experiment;
00189   GtkWidget *button_add_experiment;
00190   GtkWidget *button_remove_experiment;
00191   GtkWidget *label_experiment;
00192   GtkWidget *button_experiment;
00193   GtkWidget *label_weight;
00194   GtkWidget *spin_weight;
00195   GtkWidget *check_template[MAX_NINPUTS];
00196   GtkWidget *button_template[MAX_NINPUTS];
00197   GdkPixbuf *logo;
00198   Experiment *experiment;
00199   Variable *variable;
00200   char *application_directory;
00201   gulong id_experiment;
00202   gulong id_experiment_name;
00203   gulong id_variable;
00204   gulong id_variable_label;
00205   gulong id_template[MAX_NINPUTS];
00206   gulong id_input[MAX_NINPUTS];
00207   unsigned int n_experiments;
00208   unsigned int n_variables;
00209 } Window;
00210
00211 // Global variables
00212 extern const char *logo[];
00213 extern Options options[1];
00214 extern Running running[1];
00215 extern Window window[1];
00216
00217 // Inline functions
00218 #if GTK_MAJOR_VERSION <= 3 && GTK_MINOR_VERSION < 10
00219 static inline GtkWidget *
00220 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00221 {
00222   GtkWidget *button;
00223   GtkWidget *image;
00224   button = (GtkWidget *) gtk_button_new ();
00225   image = (GtkWidget *) gtk_image_new_from_icon_name (name, size);
00226   gtk_button_set_image (button, GTK_WIDGET (image));
00227   return button;
00228 }
00229 #endif
00230
00231 // Public functions
00232 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00233 void input_save (char *filename);
00234 void options_new ();
00235 void running_new ();
00236 unsigned int window_get_algorithm ();
00237 unsigned int window_get_direction ();
00238 unsigned int window_get_norm ();
00239 void window_save_direction ();
00240 int window_save ();
00241 void window_run ();
00242 void window_help ();
00243 void window_update_direction ();
00244 void window_update ();
00245 void window_set_algorithm ();
00246 void window_set_experiment ();
00247 void window_remove_experiment ();
00248 void window_add_experiment ();
00249 void window_name_experiment ();
00250 void window_weight_experiment ();
00251 void window_inputs_experiment ();
00252 void window_template_experiment (void *data);
00253 void window_set_variable ();
00254 void window_remove_variable ();
00255 void window_add_variable ();
00256 void window_label_variable ();
00257 void window_precision_variable ();
00258 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"

```

```

00064 #include "input.h"
00065 #include "optimize.h"
00066 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069
00070 #define DEBUG_MAIN 0
00071
00072
00081 int
00082 main (int argn, char **argc)
00083 {
00084     #if HAVE_GTK
00085         char *buffer;
00086     #endif
00087
00088     // Starting pseudo-random numbers generator
00089     #if DEBUG_MAIN
00090         fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00091     #endif
00092     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00093
00094     // Allowing spaces in the XML data file
00095     #if DEBUG_MAIN
00096         fprintf (stderr, "main: allowing spaces in the XML data file\n");
00097     #endif
00098     xmlKeepBlanksDefault (0);
00099
00100     // Starting MPI
00101     #if HAVE_MPI
00102     #if DEBUG_MAIN
00103         fprintf (stderr, "main: starting MPI\n");
00104     #endif
00105     MPI_Init (&argn, &argc);
00106     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00107     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00108     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00109     #else
00110         ntasks = 1;
00111     #endif
00112
00113     // Resetting result and variables file names
00114     #if DEBUG_MAIN
00115         fprintf (stderr, "main: resetting result and variables file names\n");
00116     #endif
00117     input->result = input->variables = NULL;
00118
00119     #if HAVE_GTK
00120
00121     // Getting threads number and pseudo-random numbers generator seed
00122     nthreads_direction = nthreads = cores_number ();
00123     optimize->seed = DEFAULT_RANDOM_SEED;
00124
00125     // Setting local language and international floating point numbers notation
00126     setlocale (LC_ALL, "");
00127     setlocale (LC_NUMERIC, "C");
00128     window->application_directory = g_get_current_dir ();
00129     buffer = g_build_filename (window->application_directory,
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

```

```

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             else
00494                 break;
00495         }
00496     }
00497     #if DEBUG_OPTIMIZE
00498         fprintf (stderr, "optimize_best: end\n");
00499     #endif
00500 }
```

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](#).

```

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 }

```

5.17.2.3 optimize_direction_sequential()

```

void optimize_direction_sequential (
    unsigned int simulation )

```

Function to estimate the direction search sequentially.

Parameters

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

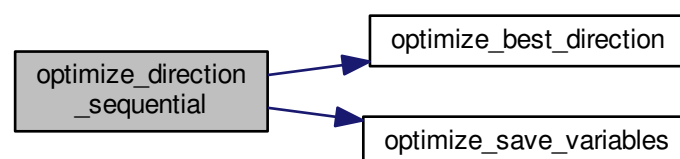
Definition at line 818 of file [optimize.c](#).

```

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->
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     }
00841     #if DEBUG_OPTIMIZE
00842         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00843     #endif
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.
-------------	----------------

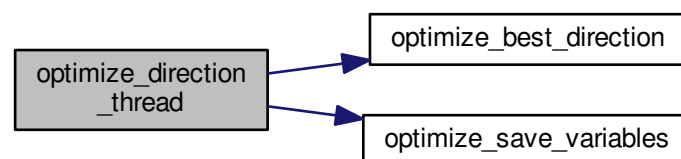
Returns

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];
```

```

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 }

```

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

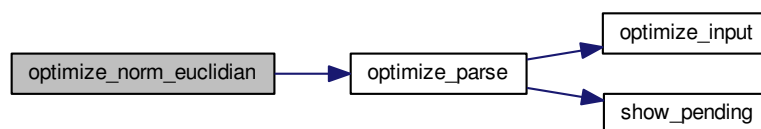
Euclidian error norm.

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

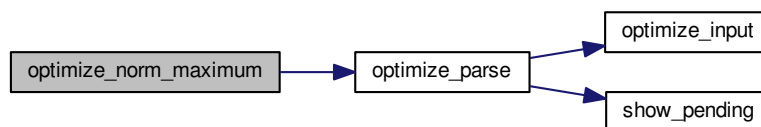
Maximum error norm.

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](#).

```

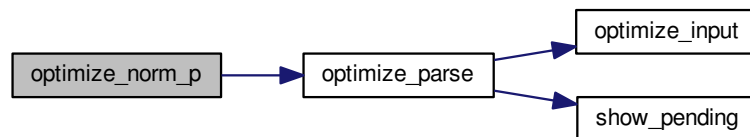
00354 {
00355     double e, ei;
00356     unsigned int i;
```

```

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.

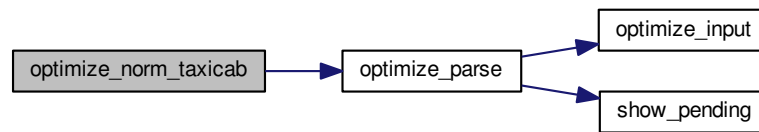
Definition at line 382 of file [optimize.c](#).

```

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]);
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     // Performing the simulation

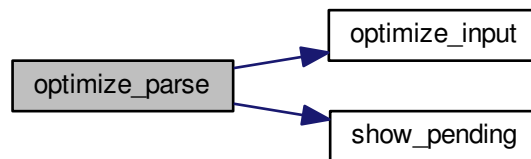
```

```

00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225             buffer4, input[0], input[1], input[2], input[3], input[4], 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         fprintf (optimize->file_variables, "%.14le\n", error);
00449         #if DEBUG_OPTIMIZE
00450         fprintf (stderr, "optimize_save_variables: end\n");
00451         #endif
00452 }
```

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.
-------------------	--------------------

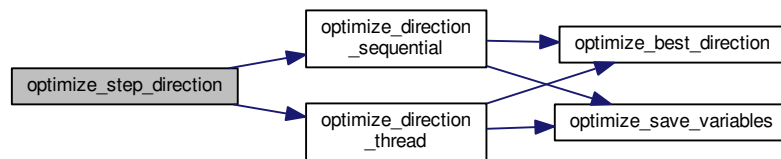
Definition at line 961 of file `optimize.c`.

```

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 *) 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

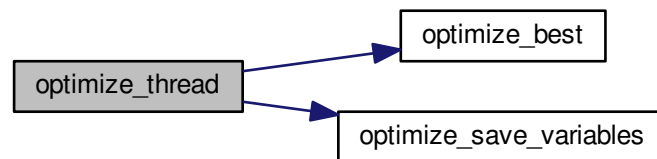
NULL

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 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 <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"

```

```

00064 #include "optimize.h"
00065
00066 #define DEBUG_OPTIMIZE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define RM "del"
00074 #else
00075 #define RM "rm"
00076 #endif
00077
00078 int ntasks;
00079 unsigned int nthreads;
00080 unsigned int nthreads_direction;
00082 GMutex mutex[1];
00083 void (*optimize_algorithm) ();
00085 double (*optimize_estimate_direction) (unsigned int variable,
00086                                         unsigned int estimate);
00088 double (*optimize_norm) (unsigned int simulation);
00090 Optimize optimize[1];
00091
00103 void
00104 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00105 {
00106     unsigned int i;
00107     char buffer[32], value[32], *buffer2, *buffer3, *content;
00108     FILE *file;
00109     gsize length;
00110     GRegex *regex;
00111
00112     #if DEBUG_OPTIMIZE
00113         fprintf (stderr, "optimize_input: start\n");
00114     #endif
00115
00116     // Checking the file
00117     if (!template)
00118         goto optimize_input_end;
00119
00120     // Opening template
00121     content = g_mapped_file_get_contents (template);
00122     length = g_mapped_file_get_length (template);
00123     #if DEBUG_OPTIMIZE
00124         fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00125     #endif
00126     file = g_fopen (input, "w");
00127
00128     // Parsing template
00129     for (i = 0; i < optimize->nvariables; ++i)
00130     {
00131         #if DEBUG_OPTIMIZE
00132             fprintf (stderr, "optimize_input: variable=%u\n", i);
00133         #endif
00134         snprintf (buffer, 32, "@variable%u@", i + 1);
00135         regex = g_regex_new (buffer, 0, 0, NULL);
00136         if (i == 0)
00137         {
00138             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00139                                                 optimize->label[i], 0, NULL);
00140         #if DEBUG_OPTIMIZE
00141             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00142         #endif
00143         }
00144         else
00145         {
00146             length = strlen (buffer3);
00147             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00148                                                 optimize->label[i], 0, NULL);
00149             g_free (buffer3);
00150         }
00151         g_regex_unref (regex);
00152         length = strlen (buffer2);
00153         snprintf (buffer, 32, "@value%u@", i + 1);
00154         regex = g_regex_new (buffer, 0, 0, NULL);
00155         snprintf (value, 32, format[optimize->precision[i]],
00156                 optimize->value[simulation * optimize->nvariables + i]);
00157
00158         #if DEBUG_OPTIMIZE
00159             fprintf (stderr, "optimize_input: value=%s\n", value);
00160         #endif
00161         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00162                                           0, NULL);
00163         g_free (buffer2);
00164         g_regex_unref (regex);
00165     }
00166
00167     // Saving input file
00168     fwrite (buffer3, strlen (buffer3), sizeof (char), file);

```

```

00169     g_free (buffer3);
00170     fclose (file);
00171
00172 optimize_input_end:
00173 #if DEBUG_OPTIMIZE
00174     fprintf (stderr, "optimize_input: end\n");
00175 #endif
00176     return;
00177 }
00178
00179 double
00190 optimize_parse (unsigned int simulation, unsigned int experiment)
00191 {
00192     unsigned int i;
00193     double e;
00194     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00195         *buffer3, *buffer4;
00196     FILE *file_result;
00197
00198 #if DEBUG_OPTIMIZE
00199     fprintf (stderr, "optimize_parse: start\n");
00200     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", 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->file[i][experiment]);
00212     }
00213     for (; i < MAX_NINPUTS; ++i)
00214         strcpy (&input[i][0], "");
00215 #if DEBUG_OPTIMIZE
00216     fprintf (stderr, "optimize_parse: parsing end\n");
00217 #endif
00218
00219     // Performing the simulation
00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225             buffer4, input[0], input[1], input[2], input[3], input[4], 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 }
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->
00416             error_old[0]);
00417     for (i = 0; i < optimize->nvariables; ++i)
00418     {
00419         snprintf (buffer, 512, "%s = %s\n",
00420                 optimize->label[i], format[optimize->precision[i]]);
00421         printf (buffer, optimize->value_old[i]);
00422         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00423     }
00424     fflush (optimize->file_result);
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->
00486                     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
00499 void
00500 optimize_sequential ()
00501 {
00502     unsigned int i;
00503     double e;
00504 #if DEBUG_OPTIMIZE
00505     fprintf(stderr, "optimize_sequential: start\n");
00506     fprintf(stderr, "optimize_sequential: nstart=%u nend=%u\n",
00507         optimize->nstart, optimize->nend);
00508 #endif
00509     for (i = optimize->nstart; i < optimize->nend; ++i)
00510     {
00511         e = optimize_norm (i);
00512         optimize_best (i, e);
00513         optimize_save_variables (i, e);
00514         if (e < optimize->threshold)
00515         {
00516             optimize->stop = 1;
00517             break;
00518         }
00519 #if DEBUG_OPTIMIZE
00520     fprintf(stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00521 #endif
00522     }
00523 #if DEBUG_OPTIMIZE
00524     fprintf(stderr, "optimize_sequential: end\n");
00525 #endif
00526 }
00527
00528 void *
00529 optimize_thread (ParallelData * data)
00530 {
00531     unsigned int i, thread;
00532     double e;
00533 #if DEBUG_OPTIMIZE
00534     fprintf(stderr, "optimize_thread: start\n");
00535 #endif
00536     thread = data->thread;
00537 #if DEBUG_OPTIMIZE
00538     fprintf(stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00539         optimize->thread[thread], optimize->thread[thread + 1]);
00540 #endif
00541     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00542     {
00543         e = optimize_norm (i);
00544         g_mutex_lock (mutex);
00545         optimize_best (i, e);
00546         optimize_save_variables (i, e);
00547         if (e < optimize->threshold)
00548             optimize->stop = 1;
00549         g_mutex_unlock (mutex);
00550         if (optimize->stop)
00551             break;
00552 #if DEBUG_OPTIMIZE
00553     fprintf(stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00554 #endif
00555     }
00556 #if DEBUG_OPTIMIZE
00557     fprintf(stderr, "optimize_thread: end\n");
00558 #endif
00559     g_thread_exit (NULL);
00560     return NULL;
00561 }
00562
00563 void
00564 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00565     double *error_best)
00566 {
00567     unsigned int i, j, k, s[optimize->nbest];
00568     double e[optimize->nbest];
00569 #if DEBUG_OPTIMIZE
00570     fprintf(stderr, "optimize_merge: start\n");
00571 #endif
00572     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 += l * (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 (*) ) 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     {
00753         for (j = 0; j < optimize->nvariables; ++j)
00754             optimize->value[i * optimize->nvariables + j]
00755                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00756                   * (optimize->rangemax[j] - optimize->rangemin[j]);
00757         optimize->nsaveds = 0;
00758         if (nthreads <= 1)
00759             optimize_sequential ();
00760         else
00761         {
00762             for (i = 0; i < nthreads; ++i)
00763             {
00764                 data[i].thread = i;
00765                 thread[i] = g_thread_new (NULL, (void (*) ) optimize_thread, &data[i]);
00766             }
00767             for (i = 0; i < nthreads; ++i)
00768                 g_thread_join (thread[i]);
00769         }
00770 #if HAVE_MPI
00771         // Communicating tasks results
00772         optimize_synchronise ();
00773 #endif
00774 #if DEBUG_OPTIMIZE
00775         fprintf (stderr, "optimize_MonteCarlo: end\n");
00776 #endif
00777     }
00778 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 #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 }
00810
00811 void
00812 optimize_direction_sequential (unsigned int simulation)
00813 {
00814     unsigned int i, j;
00815     double e;
00816 #if DEBUG_OPTIMIZE
00817     fprintf (stderr, "optimize_direction_sequential: start\n");
00818     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00819             "nend_direction=%u\n",
00820             optimize->nstart_direction, optimize->nend_direction);
00821 #endif
00822     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00823     {
00824         j = simulation + i;
00825         e = optimize_norm (j);
00826         optimize_best_direction (j, e);
00827         optimize_save_variables (j, e);
00828         if (e < optimize->threshold)
00829         {
00830             optimize->stop = 1;
00831             break;
00832         }
00833 #if DEBUG_OPTIMIZE
00834         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00835 #endif
00836     }
00837 #if DEBUG_OPTIMIZE
00838     fprintf (stderr, "optimize_direction_sequential: end\n");
00839 #endif
00840 }
00841
00842 void *
00843 optimize_direction_thread (ParallelData * data)
00844 {
00845     unsigned int i, thread;
00846     double e;
00847 #if DEBUG_OPTIMIZE
00848     fprintf (stderr, "optimize_direction_thread: start\n");
00849 #endif
00850     thread = data->thread;
00851 #if DEBUG_OPTIMIZE
00852     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00853             thread,
00854             optimize->thread_direction[thread],
00855             optimize->thread_direction[thread + 1]);
00856 #endif
00857     for (i = optimize->thread_direction[thread];
00858          i < optimize->thread_direction[thread + 1]; ++i)
00859     {
00860         e = optimize_norm (i);
00861         g_mutex_lock (mutex);
00862         optimize_best_direction (i, e);
00863         optimize_save_variables (i, e);
00864         if (e < optimize->threshold)
00865             optimize->stop = 1;
00866         g_mutex_unlock (mutex);
00867         if (optimize->stop)
00868             break;
00869 #if DEBUG_OPTIMIZE
00870         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00871 #endif
00872     }
00873 #if DEBUG_OPTIMIZE
00874     fprintf (stderr, "optimize_direction_thread: end\n");
00875 #endif
00876     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     }
01013     for (i = 0; i < nthreads_direction; ++i)
01014     {
01015         data[i].thread = i;
01016         thread[i] = g_thread_new
01017             (NULL, (void (*)(void *)) optimize_direction_thread, &data[i]);
01018     }
01019     for (i = 0; i < nthreads_direction; ++i)
01020     {
01021         g_thread_join (thread[i]);
01022     }
01023     #if DEBUG_OPTIMIZE
01024     fprintf (stderr, "optimize_step_direction: end\n");
01025     #endif
01026 }
01027
01028 void
01029 optimize_direction ()
01030 {
01031     unsigned int i, j, k, b, s, adjust;
01032     #if DEBUG_OPTIMIZE
01033     fprintf (stderr, "optimize_direction: start\n");
01034     #endif
01035     for (i = 0; i < optimize->nvariables; ++i)
01036     {
01037         optimize->direction[i] = 0.;
01038         b = optimize->simulation_best[0] * optimize->nvariables;
01039         s = optimize->nsimulations;
01040         adjust = 1;
01041         for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01042         {
01043             #if DEBUG_OPTIMIZE
01044             fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01045                 i, optimize->simulation_best[0]);
01046             #endif
01047             optimize_step_direction (s);
01048             k = optimize->simulation_best[0] * optimize->nvariables;
01049             #if DEBUG_OPTIMIZE
01050             fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01051                 i, optimize->simulation_best[0]);
01052             #endif
01053             if (k == b)
01054             {
01055                 if (adjust)
01056                 {
01057                     for (j = 0; j < optimize->nvariables; ++j)
01058                     {
01059                         optimize->step[j] *= 0.5;
01060                     }
01061                     for (j = 0; j < optimize->nvariables; ++j)
01062                     {
01063                         optimize->direction[j] = 0.;
01064                     }
01065                     adjust = 1;
01066                 }
01067                 else
01068                 {
01069                     for (j = 0; j < optimize->nvariables; ++j)
01070                     {
01071                         #if DEBUG_OPTIMIZE
01072                         fprintf (stderr,
01073                             "optimize_direction: best=%u old=%u\n",
01074                             j, optimize->value[k + j], j, optimize->value[b + j]);
01075                         #endif
01076                         optimize->direction[j]
01077                         = (1. - optimize->relaxation) * optimize->direction[j]
01078                         + optimize->relaxation
01079                         * (optimize->value[k + j] - optimize->value[b + j]);
01080                         #if DEBUG_OPTIMIZE
01081                         fprintf (stderr, "optimize_direction: direction%u=%u\n",
01082                             j, optimize->direction[j]);
01083                         #endif
01084                     }
01085                     adjust = 0;
01086                 }
01087             }
01088             #if DEBUG_OPTIMIZE
01089             fprintf (stderr, "optimize_direction: end\n");
01090             #endif
01091         }
01092     }
01093     double
01094     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
01148     genetic_algorithm_default (optimize->nvariables,
01149         optimize->genetic_variable,
01150         optimize->nsimulations,
01151         optimize->niterations,
01152         optimize->mutation_ratio,
01153         optimize->reproduction_ratio,
01154         optimize->adaptation_ratio,
01155         optimize->seed,
01156         optimize->threshold,
01157         &optimize_genetic_objective,
01158         &best_genome, &best_variable, &best_objective);
01159     #if DEBUG_OPTIMIZE
01160     fprintf (stderr, "optimize_genetic: the best\n");
01161     #endif
01162     optimize->error_old = (double *) g_malloc (sizeof (double));
01163     optimize->value_old
01164         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01165     optimize->error_old[0] = best_objective;
01166     memcpy (optimize->value_old, best_variable,
01167         optimize->nvariables * sizeof (double));
01168     g_free (best_genome);
01169     g_free (best_variable);
01170     optimize_print ();
01171     #if DEBUG_OPTIMIZE
01172     fprintf (stderr, "optimize_genetic: end\n");
01173     #endif
01174 }
01175
01179 void
01180 optimize_save_old ()
01181 {
01182     unsigned int i, j;
01183     #if DEBUG_OPTIMIZE
01184     fprintf (stderr, "optimize_save_old: start\n");
01185     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01186     #endif
01187     memcpy (optimize->error_old, optimize->error_best,
01188         optimize->nbest * sizeof (double));
01189     for (i = 0; i < optimize->nbest; ++i)
01190     {
01191         j = optimize->simulation_best[i];

```

```

01192 #if DEBUG_OPTIMIZE
01193     fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01194 #endif
01195     memcpy (optimize->value_old + i * optimize->nvariables,
01196             optimize->value + j * optimize->nvariables,
01197             optimize->nvariables * sizeof (double));
01198 }
01199 #if DEBUG_OPTIMIZE
01200     for (i = 0; i < optimize->nvariables; ++i)
01201         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01202                 i, optimize->value_old[i]);
01203     fprintf (stderr, "optimize_save_old: end\n");
01204 #endif
01205 }
01206
01212 void
01213 optimize_merge_old ()
01214 {
01215     unsigned int i, j, k;
01216     double v[optimize->nbest * optimize->nvariables], e[optimize->
nbest],
01217          *enew, *eold;
01218     #if DEBUG_OPTIMIZE
01219         fprintf (stderr, "optimize_merge_old: start\n");
01220     #endif
01221     anew = optimize->error_best;
01222     eold = optimize->error_old;
01223     i = j = k = 0;
01224     do
01225     {
01226         if (*enew < *eold)
01227         {
01228             memcpy (v + k * optimize->nvariables,
01229                     optimize->value
01230                     + optimize->simulation_best[i] * optimize->
nvariables,
01231                     optimize->nvariables * sizeof (double));
01232             e[k] = *enew;
01233             ++k;
01234             ++enew;
01235             ++i;
01236         }
01237         else
01238         {
01239             memcpy (v + k * optimize->nvariables,
01240                     optimize->value_old + j * optimize->nvariables,
01241                     optimize->nvariables * sizeof (double));
01242             e[k] = *eold;
01243             ++k;
01244             ++eold;
01245             ++j;
01246         }
01247     }
01248     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]
= optimize->value_old[j];
01279             }
01280             for (i = 0; ++i < optimize->nbest; )
01281             {
01282                 for (j = 0; j < optimize->nvariables; ++j)
01283                 {
01284                     optimize->rangemin[j]
= fmin (optimize->rangemin[j],
01285

```

```

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
01342 void
01343 optimize_step ()
01344 {
01345     #if DEBUG_OPTIMIZE
01346     fprintf (stderr, "optimize_step: start\n");
01347     #endif
01348     optimize_algorithm ();
01349     if (optimize->nsteps)
01350         optimize_direction ();
01351     #if DEBUG_OPTIMIZE
01352     fprintf (stderr, "optimize_step: end\n");
01353     #endif
01354 }
01355
01356 void
01357 optimize_iterate ()
01358 {
01359     unsigned int i;
01360     #if DEBUG_OPTIMIZE
01361     fprintf (stderr, "optimize_iterate: start\n");
01362     #endif
01363     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01364     optimize->value_old = (double *)
01365         g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01366     optimize_step ();
01367     optimize_save_old ();
01368     optimize_refine ();
01369     optimize_print ();
01370     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01371     {
01372         optimize_step ();
01373         optimize_merge_old ();
01374     }

```

```

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 =
01496 optimize_estimate_direction_coordinates;
01496         break;
01497         default:
01498             optimize->nestimates = input->nestimates;
01499             optimize_estimate_direction =
01500 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].
01530         name;
01531     optimize->weight[i] = input->experiment[i].weight;
01532     #if DEBUG_OPTIMIZE
01533     fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",
01534         optimize->experiment[i], optimize->weight[i]);
01535     #endif
01536     for (j = 0; j < input->experiment->ninputs; ++j)
01537     {
01538         #if DEBUG_OPTIMIZE
01539         fprintf (stderr, "optimize_open: template%u\n", j + 1);
01540         #endif
01541         optimize->file[j][i]
01542             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
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 *)
g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01618         for (i = 0; i < optimize->nvariables; ++i)
01619         {
01620 #if DEBUG_OPTIMIZE
01621             fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01622                 i, optimize->rangemin[i], optimize->rangemax[i],
01623                 optimize->nbits[i]);
01624 #endif
01625             optimize->genetic_variable[i].minimum = optimize->
rangemin[i];
01626             optimize->genetic_variable[i].maximum = optimize->
rangemax[i];
01627             optimize->genetic_variable[i].nbits = optimize->nbits[i];
01628         }
01629     }
01630 #if DEBUG_OPTIMIZE
01631     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01632         optimize->nvariables, optimize->nsimulations);
01633 #endif
01634     optimize->value = (double *)
g_malloc ((optimize->nsimulations
+ optimize->nestimates * optimize->nsteps)
* optimize->nvariables * sizeof (double));
01639
01640     // Calculating simulations to perform for each task
01641 #if HAVE_MPI

```

```

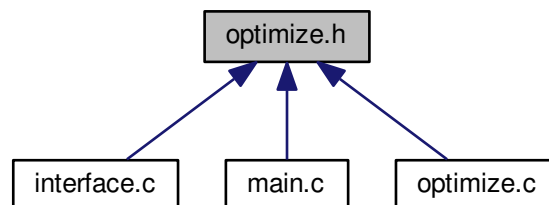
01642 #if DEBUG_OPTIMIZE
01643     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01644             optimize->mpi_rank, ntasks);
01645 #endif
01646     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
    ntasks;
01647     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
    ntasks;
01648     if (optimize->nsteps)
01649     {
01650         optimize->nstart_direction
01651             = optimize->mpi_rank * optimize->nestimates / ntasks;
01652         optimize->nend_direction
01653             = (1 + optimize->mpi_rank) * optimize->nestimates /
    ntasks;
01654     }
01655 #else
01656     optimize->nstart = 0;
01657     optimize->nend = optimize->nsimulations;
01658     if (optimize->nsteps)
01659     {
01660         optimize->nstart_direction = 0;
01661         optimize->nend_direction = optimize->nestimates;
01662     }
01663 #endif
01664 #if DEBUG_OPTIMIZE
01665     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01666             optimize->nend);
01667 #endif
01668 // Calculating simulations to perform for each thread
01669 optimize->thread
01670     = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01671 for (i = 0; i <= nthreads; ++i)
01672 {
01673     optimize->thread[i] = optimize->nstart
01674         + i * (optimize->nend - optimize->nstart) / nthreads;
01675 #if DEBUG_OPTIMIZE
01676     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01677             optimize->thread[i]);
01678 #endif
01679 }
01680 if (optimize->nsteps)
01681     optimize->thread_direction = (unsigned int *)
01682         alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01683 // Opening result files
01684 optimize->file_result = g_fopen (optimize->result, "w");
01685 optimize->file_variables = g_fopen (optimize->variables, "w");
01686 // Performing the algorithm
01687 switch (optimize->algorithm)
01688 {
01689     // Genetic algorithm
01690     case ALGORITHM_GENETIC:
01691         optimize_genetic ();
01692         break;
01693     // Iterative algorithm
01694     default:
01695         optimize_iterate ();
01696 }
01697 // Getting calculation time
01698 t = g_date_time_new_now (tz);
01699 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01700 g_date_time_unref (t);
01701 g_date_time_unref (t0);
01702 g_time_zone_unref (tz);
01703 printf ("%s = %.6lg s\n",
01704         gettext ("Calculation time"), optimize->calculation_time);
01705 fprintf (optimize->file_result, "%s = %.6lg s\n",
01706         gettext ("Calculation time"), optimize->calculation_time);
01707 // Closing result files
01708 fclose (optimize->file_variables);
01709 fclose (optimize->file_result);
01710 #if DEBUG_OPTIMIZE
01711     fprintf (stderr, "optimize_open: end\n");
01712 #endif
01713 }

```

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](#).

```

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 }

```

5.19.2.3 optimize_direction_sequential()

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

Function to estimate the direction search sequentially.

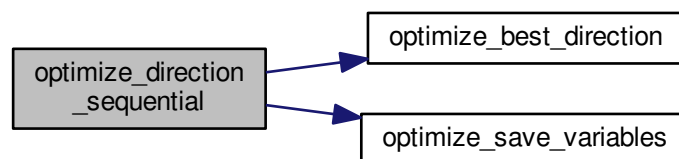
Parameters

<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->
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:



5.19.2.4 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

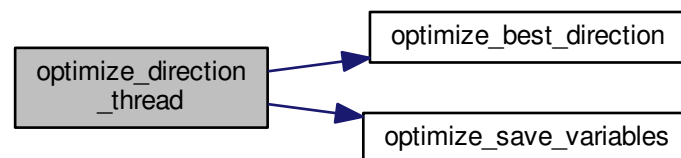
Definition at line 856 of file [optimize.c](#).

```

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->
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 }

```

5.19.2.9 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

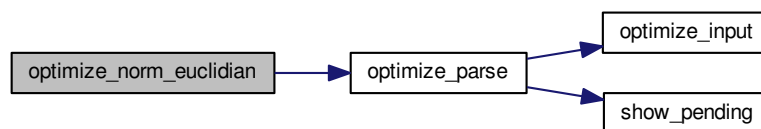
Euclidian error norm.

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.

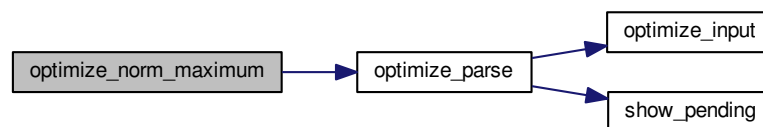
Definition at line 325 of file [optimize.c](#).

```

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;

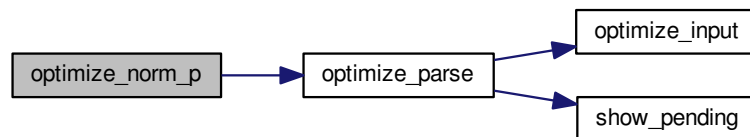
```

```

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.

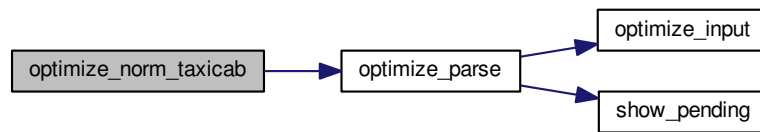
Definition at line 382 of file [optimize.c](#).

```

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     // Performing the simulation

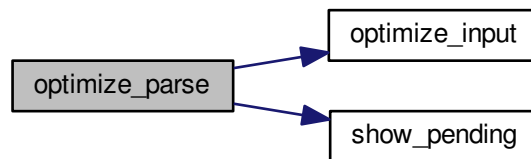
```

```

00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225             buffer4, input[0], input[1], input[2], input[3], input[4], 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:



5.19.2.15 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->
00447                               nvariables + i]);
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.
-------------------	--------------------

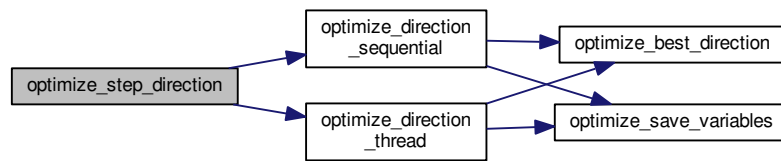
Definition at line 961 of file `optimize.c`.

```

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

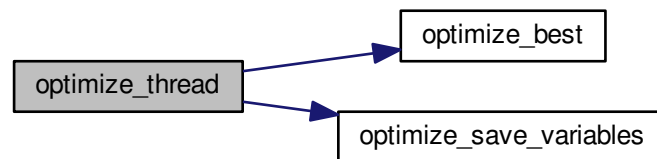
NULL

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 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 #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;
  
```

```

00072 double *error_old;
00074 unsigned int *precision;
00075 unsigned int *nsweeps;
00076 unsigned int *nbits;
00078 unsigned int *thread;
00080 unsigned int *thread_direction;
00083 unsigned int *simulation_best;
00084 double tolerance;
00085 double mutation_ratio;
00086 double reproduction_ratio;
00087 double adaptation_ratio;
00088 double relaxation;
00089 double calculation_time;
00090 double p;
00091 double threshold;
00092 unsigned long int seed;
00094 unsigned int nvariables;
00095 unsigned int nexperiments;
00096 unsigned int ninputs;
00097 unsigned int nsimulations;
00098 unsigned int nsteps;
00100 unsigned int nestimates;
00102 unsigned int algorithm;
00103 unsigned int nstart;
00104 unsigned int nend;
00105 unsigned int nstart_direction;
00107 unsigned int nend_direction;
00109 unsigned int niterations;
00110 unsigned int nbest;
00111 unsigned int nsaveds;
00112 unsigned int stop;
00113 #if HAVE_MPI
00114 int mpi_rank;
00115 #endif
00116 } Optimize;
00117
00122 typedef struct
00123 {
00124     unsigned int thread;
00125 } ParallelData;
00126
00127 // Global variables
00128 extern int ntasks;
00129 extern unsigned int nthreads;
00130 extern unsigned int nthreads_direction;
00131 extern GMutex mutex[1];
00132 extern void (*optimize_algorithm) ();
00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134                                              unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
00137
00138 // Public functions
00139 void optimize_input (unsigned int simulation, char *input,
00140                    GMappedFile * template);
00141 double optimize_parse (unsigned int simulation, unsigned int experiment);
00142 double optimize_norm_euclidian (unsigned int simulation);
00143 double optimize_norm_maximum (unsigned int simulation);
00144 double optimize_norm_p (unsigned int simulation);
00145 double optimize_norm_taxicab (unsigned int simulation);
00146 void optimize_print ();
00147 void optimize_save_variables (unsigned int simulation, double error);
00148 void optimize_best (unsigned int simulation, double value);
00149 void optimize_sequential ();
00150 void *optimize_thread (ParallelData * data);
00151 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00152                    double *error_best);
00153 #if HAVE_MPI
00154 void optimize_synchronize ();
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
00164                                              variable,
00165                                              unsigned int estimate);
00166 void optimize_step_direction (unsigned int simulation);
00167 void optimize_direction ();
00168 double optimize_genetic_objective (Entity * entity);
00169 void optimize_genetic ();
00170 void optimize_save_old ();
00171 void optimize_merge_old ();
00172 void optimize_refine ();

```

```

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

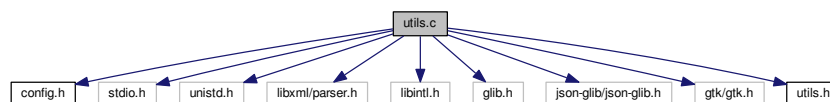
Source file to define some useful functions.

```

#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.

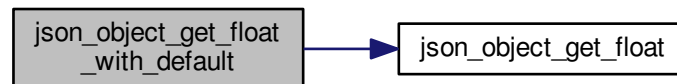
Returns

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.

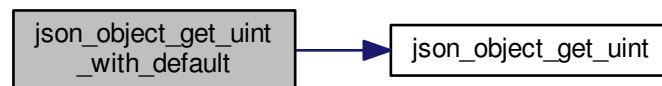
Returns

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

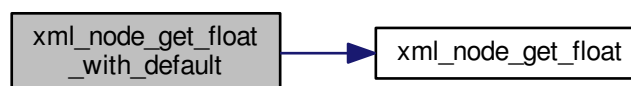
Floating point number value.

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.

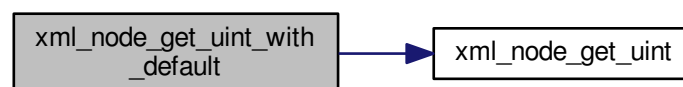
Returns

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 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 <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
```

```

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 }
00106
00107 void
00114 show_error (char *msg)
00115 {
00116     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00117 }
00118
00119 int
00132 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
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 }
00149
00150 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
00181 unsigned int
00197 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00198                                 unsigned int default_value, int *error_code)
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 }
00210
00211 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
00242     double
00257 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00259                                 double default_value, int *error_code)
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 }
00271
00272 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)

```

```

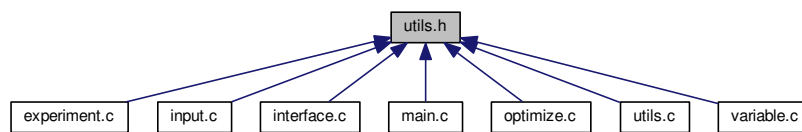
00569     if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570         break;
00571     return i;
00572 }
00573
00574 #endif

```

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 GtkWidget.
<i>n</i>	Number of GtkWidget.

Returns

Active GtkWidget.

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

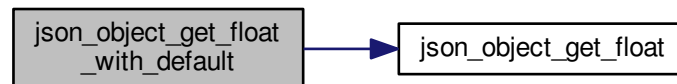
Floating point number value.

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.

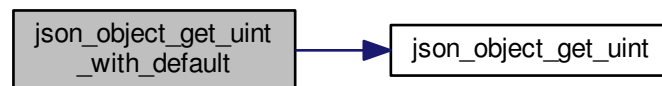
Returns

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.

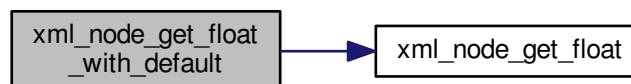
Returns

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.

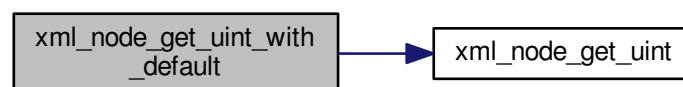
Returns

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
00032 #ifndef UTILS__H
00033 #define UTILS__H 1
00034
00035 #if HAVE_GTK
00036 #define ERROR_TYPE GTK_MESSAGE_ERROR
00037 #define INFO_TYPE GTK_MESSAGE_INFO
00038 extern GtkWidget *main_window;
00039 #else
00040 #define ERROR_TYPE 0
00041 #define INFO_TYPE 0
00042 #endif
00043
00044 extern char *error_message;
00045
00046 // Public functions
00047 void show_pending ();
00048 void show_message (char *title, char *msg, int type);
00049 void show_error (char *msg);
00050 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00051 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00052                                int *error_code);
00053 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00054                                             const xmlChar * prop,
00055                                             unsigned int default_value,
00056                                             int *error_code);
00057 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00058                           int *error_code);
00059 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00060                                         double default_value, int *error_code);
00061 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00062 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00063                        unsigned int value);
00064 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00065 int json_object_get_int (JsonObject * object, const char *prop,
00066                        int *error_code);
00067 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00068                                   int *error_code);
00069 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

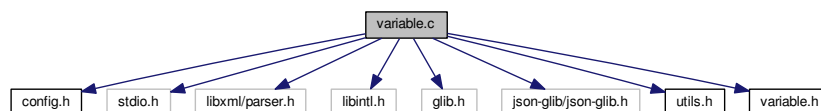
Source file to define the variable data.

```

#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,
```

```

    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

```

```

00357     = json_object_get_float_with_default (object,
00358     LABEL_ABSOLUTE_MAXIMUM,
00359     G_MAXDOUBLE, &error_code);
00359     if (error_code)
00360     {
00361         variable_error (variable, gettext ("bad absolute maximum"));
00362         goto exit_on_error;
00363     }
00364     if (variable->rangemax > variable->rangemaxabs)
00365     {
00366         variable_error (variable, gettext ("maximum range not allowed"));
00367         goto exit_on_error;
00368     }
00369     if (variable->rangemax < variable->rangemin)
00370     {
00371         variable_error (variable, gettext ("bad range"));
00372         goto exit_on_error;
00373     }
00374 }
00375 else
00376 {
00377     variable_error (variable, gettext ("no maximum range"));
00378     goto exit_on_error;
00379 }
00380 variable->precision
00381 = json_object_get_uint_with_default (object,
00382 LABEL_PRECISION,
00383     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,
00431 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 #endif

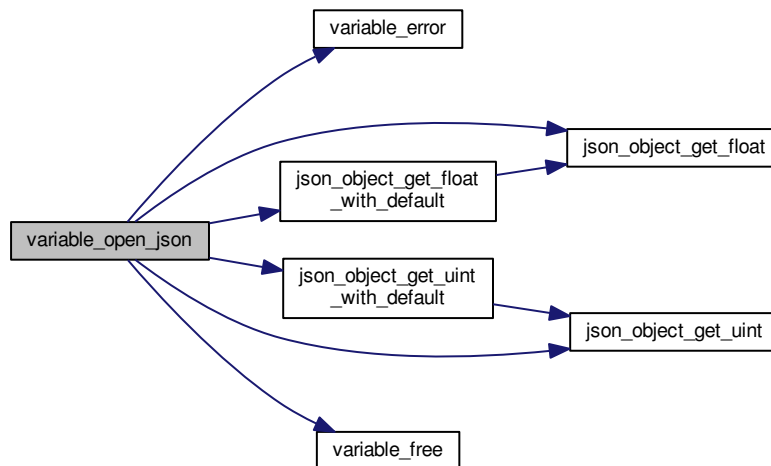
```

```

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 *)
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,
00194             &error_code);
00195         if (error_code)
00196         {
00197             variable_error (variable, gettext ("bad absolute maximum"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemax > variable->rangemaxabs)
00201         {
00202             variable_error (variable, gettext ("maximum range not allowed"));
00203             goto exit_on_error;
00204         }
00205         if (variable->rangemax < variable->rangemin)
00206         {
00207             variable_error (variable, gettext ("bad range"));
00208             goto exit_on_error;
00209         }
00210     }
00211     else
00212     {
00213         variable_error (variable, gettext ("no maximum range"));
00214         goto exit_on_error;
00215     }
00216     variable->precision
00217         = xml_node_get_uint_with_default (node, (const xmlChar *)
00218 LABEL_PRECISION,
00219                                     DEFAULT_PRECISION, &error_code);
00220     if (error_code || variable->precision >= NPRECISIONS)
00221     {
00222         variable_error (variable, gettext ("bad precision"));
00223         goto exit_on_error;
00224     }

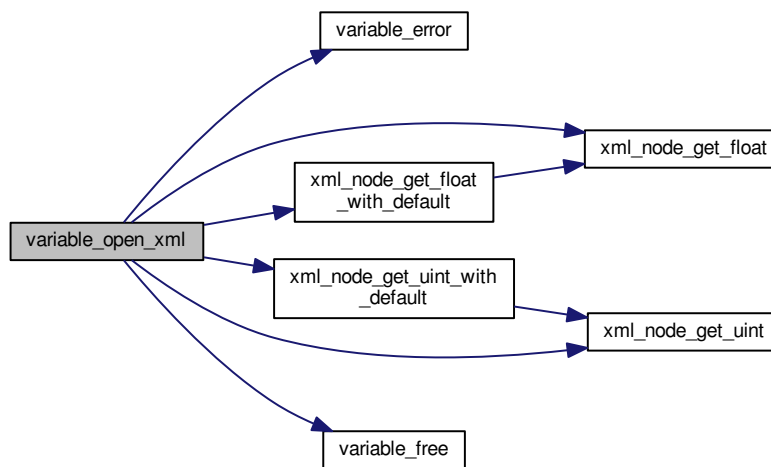
```

```

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     #if DEBUG_VARIABLE
00275     fprintf (stderr, "variable_open_xml: end\n");
00276     #endif
00277     return 1;
00278 exit_on_error:
00279     variable_free (variable, INPUT_TYPE_XML);
00280     #if DEBUG_VARIABLE
00281     fprintf (stderr, "variable_open_xml: end\n");
00282     #endif
00283     return 0;
00284 }
00285 }

```


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 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 "variable.h"
00041
00042 #define DEBUG_VARIABLE 0
00043
00044 const char *format[NPRECISIONS] = {
00045     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00046     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
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
00058         fprintf (stderr, "variable_new: start\n");
00059     #endif
00060     variable->name = NULL;
00061     #if DEBUG_VARIABLE
00062         fprintf (stderr, "variable_new: end\n");
00063     #endif
00064 }
00065
00066 void
00067 variable_free (Variable * variable, unsigned int type)
00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_free: start\n");
00071     #endif
00072     if (type == INPUT_TYPE_XML)
00073         xmlFree (variable->name);
00074     else
00075         g_free (variable->name);
00076     #if DEBUG_VARIABLE
00077         fprintf (stderr, "variable_free: end\n");
00078     #endif
00079 }
00080
00081 void
00082 variable_error (Variable * variable, char *message)
00083 {
00084     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);
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,
00194                 &error_code);
00195         if (error_code)
00196         {
00197             variable_error (variable, gettext ("bad absolute maximum"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemax > variable->rangemaxabs)
00201         {
00202             variable_error (variable, gettext ("maximum range not allowed"));
00203             goto exit_on_error;
00204         }
00205         if (variable->rangemax < variable->rangemin)
00206         {
00207             variable_error (variable, gettext ("bad range"));
00208             goto exit_on_error;
00209         }
00210     }
00211     else
00212     {
00213         variable_error (variable, gettext ("no maximum range"));
00214     }

```

```

00212     goto exit_on_error;
00213 }
00214 variable->precision
00215 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
00216                                     DEFAULT_PRECISION, &error_code);
00217 if (error_code || variable->precision >= NPRECISIONS)
00218 {
00219     variable_error (variable, gettext ("bad precision"));
00220     goto exit_on_error;
00221 }
00222 if (algorithm == ALGORITHM_SWEEP)
00223 {
00224     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225     {
00226         variable->nsweeps
00227         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NSWEEPS,
00228                             &error_code);
00229         if (error_code || !variable->nsweeps)
00230         {
00231             variable_error (variable, gettext ("bad sweeps"));
00232             goto exit_on_error;
00233         }
00234     }
00235     else
00236     {
00237         variable_error (variable, gettext ("no sweeps number"));
00238         goto exit_on_error;
00239     }
00240 #if DEBUG_VARIABLE
00241     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {
00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NBITS,
00251                             &error_code);
00252         if (error_code || !variable->nbits)
00253         {
00254             variable_error (variable, gettext ("invalid bits number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         variable_error (variable, gettext ("no bits number"));
00261         goto exit_on_error;
00262     }
00263 }
00264 else if (nsteps)
00265 {
00266     variable->step
00267     = xml_node_get_float (node, (const xmlChar *)
LABEL_STEP, &error_code);
00268     if (error_code || variable->step < 0.)
00269     {
00270         variable_error (variable, gettext ("bad step size"));
00271         goto exit_on_error;
00272     }
00273 }
00274
00275 #if DEBUG_VARIABLE
00276     fprintf (stderr, "variable_open_xml: end\n");
00277 #endif
00278     return 1;
00279 exit_on_error:
00280     variable_free (variable, INPUT_TYPE_XML);
00281 #if DEBUG_VARIABLE
00282     fprintf (stderr, "variable_open_xml: end\n");
00283 #endif
00284     return 0;
00285 }
00286
00301 int
00302 variable_open_json (Variable * variable, JsonNode * node,
00303                     unsigned int algorithm, unsigned int nsteps)
00304 {
00305     JsonObject *object;
00306     const char *label;
00307     int error_code;
00308 #if DEBUG_VARIABLE

```

```

00309     fprintf (stderr, "variable_open_json: start\n");
00310 #endif
00311     object = json_node_get_object (node);
00312     label = json_object_get_string_member (object, LABEL_NAME);
00313     if (!label)
00314     {
00315         variable_error (variable, gettext ("no name"));
00316         goto exit_on_error;
00317     }
00318     variable->name = g_strdup (label);
00319     if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321         variable->rangemin
00322         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323         if (error_code)
00324         {
00325             variable_error (variable, gettext ("bad minimum"));
00326             goto exit_on_error;
00327         }
00328         variable->rangeminabs
00329         = json_object_get_float_with_default (object,
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     variable->precision
00383     = json_object_get_uint_with_default (object,
00384 LABEL_PRECISION,
00385                                         DEFAULT_PRECISION, &error_code);
00386     if (error_code || variable->precision >= NPRECISIONS)
00387     {
00388         variable_error (variable, gettext ("bad precision"));
00389         goto exit_on_error;
00390     }
00391     if (algorithm == ALGORITHM_SWEEP)
00392     {
00393         if (json_object_get_member (object, LABEL_NSWEEPS))
00394         {
00395             variable->nsweeps

```

```

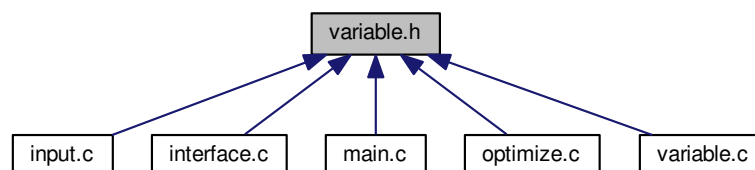
00393         = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00394     if (error_code || !variable->nsweeps)
00395     {
00396         variable_error (variable, gettext ("bad sweeps"));
00397         goto exit_on_error;
00398     }
00399 }
00400 else
00401 {
00402     variable_error (variable, 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,
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 }

```

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, xmlDoc *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.
- int [variable_open_json](#) ([Variable](#) *variable, cJSON *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 }

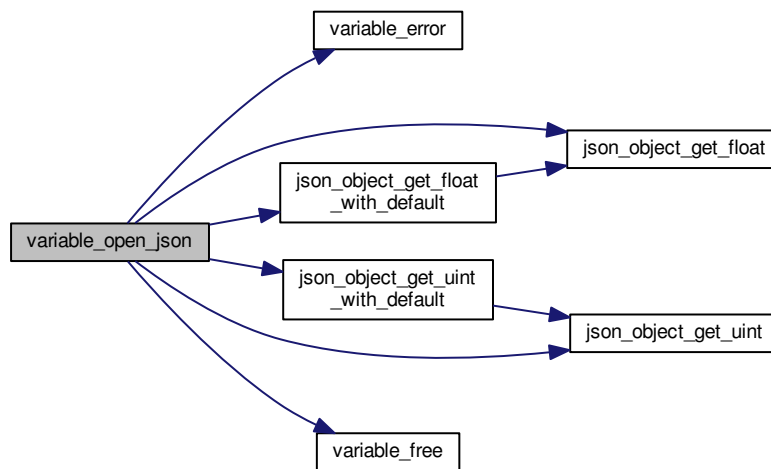
```

```

00380     variable->precision
00381     = json_object_get_uint_with_default (object,
00382     LABEL_PRECISION,
00383     DEFAULT_PRECISION, &error_code);
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,
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:



5.27.3.5 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>	<code>Variable</code> 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)

```

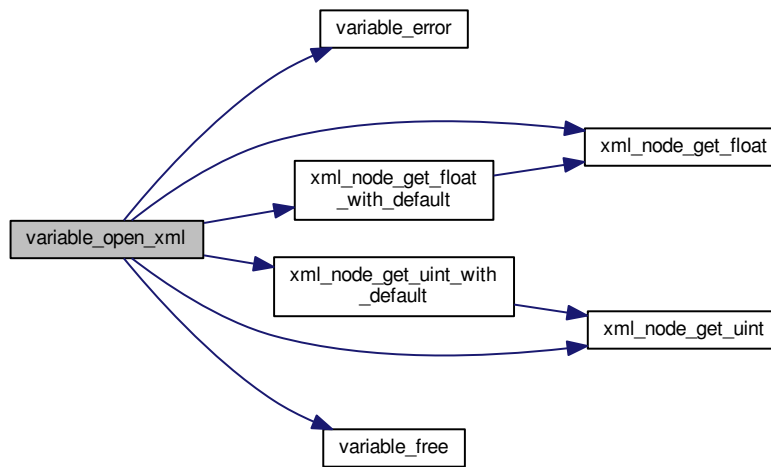
```

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,
00194         &error_code);
00195         if (error_code)
00196         {
00197             variable_error (variable, gettext ("bad absolute maximum"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemax > variable->rangemaxabs)
00201         {
00202             variable_error (variable, gettext ("maximum range not allowed"));
00203             goto exit_on_error;
00204         }
00205         if (variable->rangemax < variable->rangemin)
00206         {
00207             variable_error (variable, gettext ("bad range"));
00208             goto exit_on_error;
00209         }
00210     }
00211     else
00212     {
00213         variable_error (variable, gettext ("no maximum range"));
00214         goto exit_on_error;
00215     }
00216     variable->precision
00217     = xml_node_get_uint_with_default (node, (const xmlChar *)
00218 LABEL_PRECISION,
00219                                     DEFAULT_PRECISION, &error_code);
00220     if (error_code || variable->precision >= NPRECISIONS)
00221     {
00222         variable_error (variable, gettext ("bad precision"));
00223         goto exit_on_error;
00224     }
00225     if (algorithm == ALGORITHM_SWEEP)
00226     {
00227         if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00228         {
00229             variable->nsweeps
00230             = xml_node_get_uint (node, (const xmlChar *)
00231 LABEL_NSWEEPS,
00232                                 &error_code);
00233             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         if (error_code || !variable->nbits)
00254         {
00255             variable_error (variable, gettext ("invalid bits number"));
00256             goto exit_on_error;
00257         }
00258     }
00259     else
00260     {
00261         variable_error (variable, gettext ("no bits number"));
00262         goto exit_on_error;
00263     }
00264     else if (nsteps)
00265     {
00266         variable->step
00267         = xml_node_get_float (node, (const xmlChar *)
00268 LABEL_STEP, &error_code);
00269         if (error_code || variable->step < 0.)
00270         {
00271             variable_error (variable, gettext ("bad step size"));
00272             goto exit_on_error;
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;
  
```

```
00061 double rangeminabs;
00062 double rangemaxabs;
00063 double step;
00064 unsigned int precision;
00065 unsigned int nsweeps;
00066 unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable, unsigned int type);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open_xml (Variable * variable, xmlNode * node,
00077                      unsigned int algorithm, unsigned int nsteps);
00078 int variable_open_json (Variable * variable, JsonNode * node,
00079                      unsigned int algorithm, unsigned int nsteps);
00080
00081 #endif
```


Index

Algorithm
variable.h, 269

config.h, 29
INPUT_TYPE, 32

cores_number
utils.c, 224
utils.h, 242

DirectionMethod
input.h, 81

ErrorNorm
input.h, 81

Experiment, 15
experiment.c, 34
experiment_error, 35
experiment_free, 36
experiment_new, 36
experiment_open_json, 37
experiment_open_xml, 39
template, 41

experiment.h, 45
experiment_error, 46
experiment_free, 46
experiment_new, 48
experiment_open_json, 48
experiment_open_xml, 50

experiment_error
experiment.c, 35
experiment.h, 46

experiment_free
experiment.c, 36
experiment.h, 46

experiment_new
experiment.c, 36
experiment.h, 48

experiment_open_json
experiment.c, 37
experiment.h, 48

experiment_open_xml
experiment.c, 39
experiment.h, 50

format
variable.c, 263

gtk_array_get_active
interface.h, 150
utils.c, 224
utils.h, 242

INPUT_TYPE
config.h, 32

Input, 16

input.c, 53
input_error, 54
input_open, 55
input_open_json, 56
input_open_xml, 62

input.h, 80
DirectionMethod, 81
ErrorNorm, 81
input_error, 82
input_open, 82
input_open_json, 84
input_open_xml, 90

input_error
input.c, 54
input.h, 82

input_open
input.c, 55
input.h, 82

input_open_json
input.c, 56
input.h, 84

input_open_xml
input.c, 62
input.h, 90

input_save
interface.c, 100
interface.h, 151

input_save_direction_json
interface.c, 101

input_save_direction_xml
interface.c, 102

input_save_json
interface.c, 103

input_save_xml
interface.c, 106

interface.c, 97
input_save, 100
input_save_direction_json, 101
input_save_direction_xml, 102
input_save_json, 103
input_save_xml, 106
window_get_algorithm, 108
window_get_direction, 109
window_get_norm, 110
window_read, 110
window_save, 113

- window_template_experiment, 115
- interface.h, 147
 - gtk_array_get_active, 150
 - input_save, 151
 - window_get_algorithm, 152
 - window_get_direction, 152
 - window_get_norm, 153
 - window_read, 154
 - window_save, 156
 - window_template_experiment, 158
- json_object_get_float
 - utils.c, 224
 - utils.h, 242
- json_object_get_float_with_default
 - utils.c, 225
 - utils.h, 243
- json_object_get_int
 - utils.c, 226
 - utils.h, 244
- json_object_get_uint
 - utils.c, 227
 - utils.h, 245
- json_object_get_uint_with_default
 - utils.c, 227
 - utils.h, 245
- json_object_set_float
 - utils.c, 228
 - utils.h, 246
- json_object_set_int
 - utils.c, 229
 - utils.h, 247
- json_object_set_uint
 - utils.c, 229
 - utils.h, 247
- main.c, 162
- Optimize, 17
 - thread_direction, 20
- optimize.c, 166
 - optimize_best, 168
 - optimize_best_direction, 169
 - optimize_direction_sequential, 170
 - optimize_direction_thread, 170
 - optimize_estimate_direction_coordinates, 171
 - optimize_estimate_direction_random, 172
 - optimize_genetic_objective, 173
 - optimize_input, 173
 - optimize_merge, 174
 - optimize_norm_euclidian, 175
 - optimize_norm_maximum, 176
 - optimize_norm_p, 177
 - optimize_norm_taxicab, 178
 - optimize_parse, 179
 - optimize_save_variables, 181
 - optimize_step_direction, 181
 - optimize_thread, 183
- optimize.h, 202
 - optimize_best, 204
 - optimize_best_direction, 205
 - optimize_direction_sequential, 205
 - optimize_direction_thread, 206
 - optimize_estimate_direction_coordinates, 207
 - optimize_estimate_direction_random, 208
 - optimize_genetic_objective, 208
 - optimize_input, 209
 - optimize_merge, 210
 - optimize_norm_euclidian, 211
 - optimize_norm_maximum, 212
 - optimize_norm_p, 213
 - optimize_norm_taxicab, 214
 - optimize_parse, 215
 - optimize_save_variables, 217
 - optimize_step_direction, 217
 - optimize_thread, 219
- optimize_best
 - optimize.c, 168
 - optimize.h, 204
- optimize_best_direction
 - optimize.c, 169
 - optimize.h, 205
- optimize_direction_sequential
 - optimize.c, 170
 - optimize.h, 205
- optimize_direction_thread
 - optimize.c, 170
 - optimize.h, 206
- optimize_estimate_direction_coordinates
 - optimize.c, 171
 - optimize.h, 207
- optimize_estimate_direction_random
 - optimize.c, 172
 - optimize.h, 208
- optimize_genetic_objective
 - optimize.c, 173
 - optimize.h, 208
- optimize_input
 - optimize.c, 173
 - optimize.h, 209
- optimize_merge
 - optimize.c, 174
 - optimize.h, 210
- optimize_norm_euclidian
 - optimize.c, 175
 - optimize.h, 211
- optimize_norm_maximum
 - optimize.c, 176
 - optimize.h, 212
- optimize_norm_p
 - optimize.c, 177
 - optimize.h, 213
- optimize_norm_taxicab
 - optimize.c, 178
 - optimize.h, 214
- optimize_parse
 - optimize.c, 179

- optimize.h, 215
- optimize_save_variables
 - optimize.c, 181
 - optimize.h, 217
- optimize_step_direction
 - optimize.c, 181
 - optimize.h, 217
- optimize_thread
 - optimize.c, 183
 - optimize.h, 219
- Options, 20
- ParallelData, 21
- precision
 - variable.c, 263
- Running, 21
- show_error
 - utils.c, 229
 - utils.h, 247
- show_message
 - utils.c, 230
 - utils.h, 248
- template
 - experiment.c, 41
- thread_direction
 - Optimize, 20
- utils.c, 222
 - cores_number, 224
 - gtk_array_get_active, 224
 - json_object_get_float, 224
 - json_object_get_float_with_default, 225
 - json_object_get_int, 226
 - json_object_get_uint, 227
 - json_object_get_uint_with_default, 227
 - json_object_set_float, 228
 - json_object_set_int, 229
 - json_object_set_uint, 229
 - show_error, 229
 - show_message, 230
 - xml_node_get_float, 231
 - xml_node_get_float_with_default, 231
 - xml_node_get_int, 232
 - xml_node_get_uint, 233
 - xml_node_get_uint_with_default, 233
 - xml_node_set_float, 234
 - xml_node_set_int, 235
 - xml_node_set_uint, 235
- utils.h, 240
 - cores_number, 242
 - gtk_array_get_active, 242
 - json_object_get_float, 242
 - json_object_get_float_with_default, 243
 - json_object_get_int, 244
 - json_object_get_uint, 245
 - json_object_get_uint_with_default, 245
 - json_object_set_float, 246
 - json_object_set_int, 247
 - json_object_set_uint, 247
 - show_error, 247
 - show_message, 248
 - xml_node_get_float, 249
 - xml_node_get_float_with_default, 249
 - xml_node_get_int, 250
 - xml_node_get_uint, 251
 - xml_node_get_uint_with_default, 251
 - xml_node_set_float, 252
 - xml_node_set_int, 253
 - xml_node_set_uint, 253
- Variable, 22
- variable.c, 255
 - format, 263
 - precision, 263
 - variable_error, 256
 - variable_free, 256
 - variable_new, 257
 - variable_open_json, 257
 - variable_open_xml, 260
- variable.h, 268
 - Algorithm, 269
 - variable_error, 270
 - variable_free, 270
 - variable_new, 271
 - variable_open_json, 271
 - variable_open_xml, 274
- variable_error
 - variable.c, 256
 - variable.h, 270
- variable_free
 - variable.c, 256
 - variable.h, 270
- variable_new
 - variable.c, 257
 - variable.h, 271
- variable_open_json
 - variable.c, 257
 - variable.h, 271
- variable_open_xml
 - variable.c, 260
 - variable.h, 274
- Window, 23
- window_get_algorithm
 - interface.c, 108
 - interface.h, 152
- window_get_direction
 - interface.c, 109
 - interface.h, 152
- window_get_norm
 - interface.c, 110
 - interface.h, 153
- window_read
 - interface.c, 110
 - interface.h, 154

- window_save
 - interface.c, [113](#)
 - interface.h, [156](#)
- window_template_experiment
 - interface.c, [115](#)
 - interface.h, [158](#)
- xml_node_get_float
 - utils.c, [231](#)
 - utils.h, [249](#)
- xml_node_get_float_with_default
 - utils.c, [231](#)
 - utils.h, [249](#)
- xml_node_get_int
 - utils.c, [232](#)
 - utils.h, [250](#)
- xml_node_get_uint
 - utils.c, [233](#)
 - utils.h, [251](#)
- xml_node_get_uint_with_default
 - utils.c, [233](#)
 - utils.h, [251](#)
- xml_node_set_float
 - utils.c, [234](#)
 - utils.h, [252](#)
- xml_node_set_int
 - utils.c, [235](#)
 - utils.h, [253](#)
- xml_node_set_uint
 - utils.c, [235](#)
 - utils.h, [253](#)