

MPCOTool

3.4.0

Generated by Doxygen 1.8.12



# Contents

<b>1</b>	<b>MPCOTool</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>11</b>
2.1	Data Structures . . . . .	11
<b>3</b>	<b>File Index</b>	<b>13</b>
3.1	File List . . . . .	13
<b>4</b>	<b>Data Structure Documentation</b>	<b>15</b>
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

<b>5</b>	<b>File Documentation</b>	<b>29</b>
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	<a href="#">input_open()</a>	55
5.7.2.3	<a href="#">input_open_json()</a>	56
5.7.2.4	<a href="#">input_open_xml()</a>	61
5.8	<a href="#">input.c</a>	67
5.9	<a href="#">input.h File Reference</a>	78
5.9.1	<a href="#">Detailed Description</a>	80
5.9.2	<a href="#">Enumeration Type Documentation</a>	80
5.9.2.1	<a href="#">DirectionMethod</a>	80
5.9.2.2	<a href="#">ErrorNorm</a>	80
5.9.3	<a href="#">Function Documentation</a>	81
5.9.3.1	<a href="#">input_error()</a>	81
5.9.3.2	<a href="#">input_open()</a>	81
5.9.3.3	<a href="#">input_open_json()</a>	82
5.9.3.4	<a href="#">input_open_xml()</a>	88
5.10	<a href="#">input.h</a>	93
5.11	<a href="#">interface.c File Reference</a>	95
5.11.1	<a href="#">Detailed Description</a>	97
5.11.2	<a href="#">Function Documentation</a>	97
5.11.2.1	<a href="#">input_save()</a>	97
5.11.2.2	<a href="#">input_save_direction_json()</a>	99
5.11.2.3	<a href="#">input_save_direction_xml()</a>	99
5.11.2.4	<a href="#">input_save_json()</a>	100
5.11.2.5	<a href="#">input_save_xml()</a>	103
5.11.2.6	<a href="#">window_get_algorithm()</a>	105
5.11.2.7	<a href="#">window_get_direction()</a>	106
5.11.2.8	<a href="#">window_get_norm()</a>	107
5.11.2.9	<a href="#">window_new()</a>	107
5.11.2.10	<a href="#">window_read()</a>	116
5.11.2.11	<a href="#">window_save()</a>	118
5.11.2.12	<a href="#">window_template_experiment()</a>	120

5.12	interface.c	121
5.13	interface.h File Reference	153
5.13.1	Detailed Description	155
5.13.2	Function Documentation	155
5.13.2.1	gtk_array_get_active()	155
5.13.2.2	input_save()	156
5.13.2.3	window_get_algorithm()	157
5.13.2.4	window_get_direction()	158
5.13.2.5	window_get_norm()	158
5.13.2.6	window_new()	159
5.13.2.7	window_read()	168
5.13.2.8	window_save()	170
5.13.2.9	window_template_experiment()	172
5.14	interface.h	172
5.15	main.c File Reference	175
5.15.1	Detailed Description	176
5.15.2	Function Documentation	176
5.15.2.1	main()	176
5.16	main.c	179
5.17	optimize.c File Reference	182
5.17.1	Detailed Description	185
5.17.2	Function Documentation	185
5.17.2.1	optimize_best()	185
5.17.2.2	optimize_best_direction()	186
5.17.2.3	optimize_direction_sequential()	186
5.17.2.4	optimize_direction_thread()	187
5.17.2.5	optimize_estimate_direction_coordinates()	188
5.17.2.6	optimize_estimate_direction_random()	189
5.17.2.7	optimize_genetic_objective()	189
5.17.2.8	optimize_input()	190

5.17.2.9	<code>optimize_merge()</code>	191
5.17.2.10	<code>optimize_norm_euclidian()</code>	192
5.17.2.11	<code>optimize_norm_maximum()</code>	193
5.17.2.12	<code>optimize_norm_p()</code>	194
5.17.2.13	<code>optimize_norm_taxicab()</code>	195
5.17.2.14	<code>optimize_parse()</code>	195
5.17.2.15	<code>optimize_save_variables()</code>	197
5.17.2.16	<code>optimize_step_direction()</code>	198
5.17.2.17	<code>optimize_thread()</code>	199
5.18	<code>optimize.c</code>	200
5.19	<code>optimize.h</code> File Reference	218
5.19.1	Detailed Description	220
5.19.2	Function Documentation	220
5.19.2.1	<code>optimize_best()</code>	220
5.19.2.2	<code>optimize_best_direction()</code>	221
5.19.2.3	<code>optimize_direction_sequential()</code>	222
5.19.2.4	<code>optimize_direction_thread()</code>	223
5.19.2.5	<code>optimize_estimate_direction_coordinates()</code>	224
5.19.2.6	<code>optimize_estimate_direction_random()</code>	225
5.19.2.7	<code>optimize_genetic_objective()</code>	225
5.19.2.8	<code>optimize_input()</code>	226
5.19.2.9	<code>optimize_merge()</code>	227
5.19.2.10	<code>optimize_norm_euclidian()</code>	228
5.19.2.11	<code>optimize_norm_maximum()</code>	229
5.19.2.12	<code>optimize_norm_p()</code>	230
5.19.2.13	<code>optimize_norm_taxicab()</code>	230
5.19.2.14	<code>optimize_parse()</code>	231
5.19.2.15	<code>optimize_save_variables()</code>	233
5.19.2.16	<code>optimize_step_direction()</code>	233
5.19.2.17	<code>optimize_thread()</code>	235

5.20	<code>optimize.h</code>	236
5.21	<code>utils.c</code> File Reference	238
5.21.1	Detailed Description	239
5.21.2	Function Documentation	240
5.21.2.1	<code>cores_number()</code>	240
5.21.2.2	<code>gtk_array_get_active()</code>	240
5.21.2.3	<code>json_object_get_float()</code>	241
5.21.2.4	<code>json_object_get_float_with_default()</code>	241
5.21.2.5	<code>json_object_get_int()</code>	242
5.21.2.6	<code>json_object_get_uint()</code>	243
5.21.2.7	<code>json_object_get_uint_with_default()</code>	243
5.21.2.8	<code>json_object_set_float()</code>	244
5.21.2.9	<code>json_object_set_int()</code>	245
5.21.2.10	<code>json_object_set_uint()</code>	245
5.21.2.11	<code>show_error()</code>	246
5.21.2.12	<code>show_message()</code>	246
5.21.2.13	<code>xml_node_get_float()</code>	247
5.21.2.14	<code>xml_node_get_float_with_default()</code>	247
5.21.2.15	<code>xml_node_get_int()</code>	248
5.21.2.16	<code>xml_node_get_uint()</code>	249
5.21.2.17	<code>xml_node_get_uint_with_default()</code>	250
5.21.2.18	<code>xml_node_set_float()</code>	250
5.21.2.19	<code>xml_node_set_int()</code>	251
5.21.2.20	<code>xml_node_set_uint()</code>	251
5.22	<code>utils.c</code>	252
5.23	<code>utils.h</code> File Reference	256
5.23.1	Detailed Description	257
5.23.2	Function Documentation	258
5.23.2.1	<code>cores_number()</code>	258
5.23.2.2	<code>gtk_array_get_active()</code>	258



5.23.2.3	<a href="#">json_object_get_float()</a>	259
5.23.2.4	<a href="#">json_object_get_float_with_default()</a>	259
5.23.2.5	<a href="#">json_object_get_int()</a>	260
5.23.2.6	<a href="#">json_object_get_uint()</a>	261
5.23.2.7	<a href="#">json_object_get_uint_with_default()</a>	261
5.23.2.8	<a href="#">json_object_set_float()</a>	262
5.23.2.9	<a href="#">json_object_set_int()</a>	263
5.23.2.10	<a href="#">json_object_set_uint()</a>	263
5.23.2.11	<a href="#">show_error()</a>	264
5.23.2.12	<a href="#">show_message()</a>	264
5.23.2.13	<a href="#">xml_node_get_float()</a>	265
5.23.2.14	<a href="#">xml_node_get_float_with_default()</a>	265
5.23.2.15	<a href="#">xml_node_get_int()</a>	266
5.23.2.16	<a href="#">xml_node_get_uint()</a>	267
5.23.2.17	<a href="#">xml_node_get_uint_with_default()</a>	268
5.23.2.18	<a href="#">xml_node_set_float()</a>	268
5.23.2.19	<a href="#">xml_node_set_int()</a>	269
5.23.2.20	<a href="#">xml_node_set_uint()</a>	269
5.24	<a href="#">utils.h</a>	270
5.25	<a href="#">variable.c File Reference</a>	271
5.25.1	<a href="#">Detailed Description</a>	272
5.25.2	<a href="#">Function Documentation</a>	272
5.25.2.1	<a href="#">variable_error()</a>	272
5.25.2.2	<a href="#">variable_free()</a>	273
5.25.2.3	<a href="#">variable_new()</a>	273
5.25.2.4	<a href="#">variable_open_json()</a>	274
5.25.2.5	<a href="#">variable_open_xml()</a>	276
5.25.3	<a href="#">Variable Documentation</a>	278
5.25.3.1	<a href="#">format</a>	279
5.25.3.2	<a href="#">precision</a>	279
5.26	<a href="#">variable.c</a>	279
5.27	<a href="#">variable.h File Reference</a>	284
5.27.1	<a href="#">Detailed Description</a>	285
5.27.2	<a href="#">Enumeration Type Documentation</a>	285
5.27.2.1	<a href="#">Algorithm</a>	285
5.27.3	<a href="#">Function Documentation</a>	286
5.27.3.1	<a href="#">variable_error()</a>	286
5.27.3.2	<a href="#">variable_free()</a>	286
5.27.3.3	<a href="#">variable_new()</a>	287
5.27.3.4	<a href="#">variable_open_json()</a>	287
5.27.3.5	<a href="#">variable_open_xml()</a>	290
5.28	<a href="#">variable.h</a>	292



# Chapter 1

## MPCOTool

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

### VERSIONS

- 3.4.0: Stable and recommended version.

### AUTHORS

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

- 3.4.0/configure.ac: configure generator.
- 3.4.0/Makefile.in: Makefile generator.
- 3.4.0/config.h.in: config header generator.
- 3.4.0/mpcotool.c: main source code.
- 3.4.0/mpcotool.h: main header code.
- 3.4.0/mpcotool.ico: icon file.
- 3.4.0/interface.h: interface header code.
- 3.4.0/build: script to build all.
- 3.4.0/logo.png: logo figure.
- 3.4.0/Doxyfile: configuration file to generate doxygen documentation.
- TODO: tasks to do.
- [README.md](#): this file.
- license.md: license file.
- tests/testX/\*: several tests to check the program working.
- locales/\*/LC\_MESSAGES/mpcotool.po: translation files.
- manuals/\*.eps: manual figures in EPS format.
- manuals/\*.png: manual figures in PNG format.
- manuals/\*.tex: documentation source files.
- applications/\*/\*: several practical application cases.
- check\_errors/\*.xml: several mistaken files to check error handling.

## BUILDING INSTRUCTIONS

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

Debian 8 (Linux, kFreeBSD or Hurd)

DragonFly BSD 4.6

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/3.4.0
$ ln -s ../../genetic/2.2.0 genetic
$ ln -s genetic/libgenetic.so
```

or in Windows systems:

```
$ ln -s genetic/libgenetic.dll
```

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_with_mpi
```

#### Linking as an external library

MPCOTool can also be used as an external library:

1. First copy the dynamic libraries (libmpcotool.so and libgenetic.so on Unix systems or libmpcotool.dll and libgenetic.dll on Windows systems) to your program directory.
2. Include the function header in your source code:

```
extern int mpcotool (int argn, char **argc);
```

3. Build the executable file with the linker and compiler flags:

```
$ gcc -L. -Wl,-rpath=. -lmpcotool -lgenetic ... `pkg-config --cflags --libs gsl glib-2.0 json-glib-1.0 ...`
```

4. Calling to this function is equivalent to command line order (see next chapter USER INSTRUCTIONS):

- argn: number of arguments
- argc[0]: "mpcotool"
- argc[1]: first command line argument.

...

- argc[argn-1]: last argument.

## 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/3.4.0`):

```
$ cd ../tests/test2
$ ln -s ../../genetic/2.2.0/libgenetic.so (or .dll on Windows systems)
$ cd ../test3
$ ln -s ../../genetic/2.2.0/libgenetic.so (or .dll on Windows systems)
$ cd ../test4
$ ln -s ../../genetic/2.2.0/libgenetic.so (or .dll on Windows systems)
```

2. Build all tests doing in the same terminal:

```
$ cd ../../3.4.0
$ make tests
```

## USER INSTRUCTIONS

Optional arguments are typed in square brackets.

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

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

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

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

- The syntax of the simulator has to be:

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

- The syntax of the program to evaluate the 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:

x (number of iterations)

- Moreover, both brute force algorithms can be coupled with a direction search method by using:

- *direction*: method to estimate the optimal direction. Two options are currently available:

- \* *coordinates*: coordinates descent method.

It increases the total number of simulations by:

(number of experiments) x (number of iterations) x (number of steps) x 2 x (number of variables)

- \* *random*: random method. It requires:

- \* *nestimates*: number of random checks to estimate the optimal direction.

It increases the total number of simulations by:

(number of experiments) x (number of iterations) x (number of steps) x (number of estimates)

Both methods require also:

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

and for each variable:

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

- **genetic**: Genetic algorithm. It requires the following parameters:

- *npopulation*: number of population.
- *ngenerations*: number of generations.
- *mutation*: mutation ratio.
- *reproduction*: reproduction ratio.
- *adaptation*: adaptation ratio.

and for each variable:

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

The total number of simulations to run is:

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

Implemented error 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",
      "templatel": "template_1_1",
      "template2": "template_1_2",
      ...
      "weight": "weight_1",
    },
    ...
    {
      "name": "data_file_N",
      "templatel": "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:

<a href="#">Experiment</a>	Struct to define the experiment data . . . . .	15
<a href="#">Input</a>	Struct to define the optimization input file . . . . .	16
<a href="#">Optimize</a>	Struct to define the optimization ation data . . . . .	17
<a href="#">Options</a>	Struct to define the options dialog . . . . .	20
<a href="#">ParallelData</a>	Struct to pass to the GThreads parallelized function . . . . .	21
<a href="#">Running</a>	Struct to define the running dialog . . . . .	21
<a href="#">Variable</a>	Struct to define the variable data . . . . .	22
<a href="#">Window</a>	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:

<a href="#">config.h</a>	Configuration header file . . . . .	29
<a href="#">experiment.c</a>	Source file to define the experiment data . . . . .	34
<a href="#">experiment.h</a>	Header file to define the experiment data . . . . .	45
<a href="#">input.c</a>	Source file to define the input functions . . . . .	53
<a href="#">input.h</a>	Header file to define the input functions . . . . .	78
<a href="#">interface.c</a>	Source file to define the graphical interface functions . . . . .	95
<a href="#">interface.h</a>	Header file to define the graphical interface functions . . . . .	153
<a href="#">main.c</a>	Main source file . . . . .	175
<a href="#">optimize.c</a>	Source file to define the optimization functions . . . . .	182
<a href="#">optimize.h</a>	Header file to define the optimization functions . . . . .	218
<a href="#">utils.c</a>	Source file to define some useful functions . . . . .	238
<a href="#">utils.h</a>	Header file to define some useful functions . . . . .	256
<a href="#">variable.c</a>	Source file to define the variable data . . . . .	271
<a href="#">variable.h</a>	Header file to define the variable data . . . . .	284





## 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:

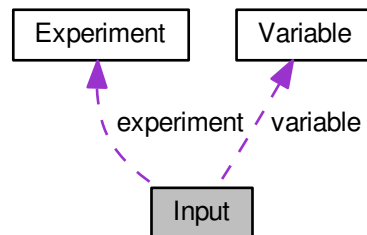
- [experiment.h](#)

## 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](#)

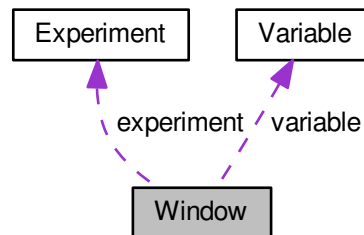


## 4.8 Window Struct Reference

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](#)



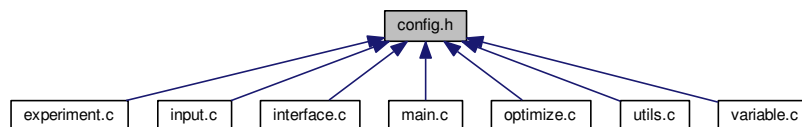
## Chapter 5

# File Documentation

### 5.1 config.h File Reference

Configuration header file.

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



### Macros

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

*Locales directory.*

- #define PROGRAM\_INTERFACE "mpcotool"  
*Name of the interface program.*
- #define LABEL\_ABSOLUTE\_MINIMUM "absolute\_minimum"  
*absolute minimum label.*
- #define LABEL\_ABSOLUTE\_MAXIMUM "absolute\_maximum"  
*absolute maximum label.*
- #define LABEL\_ADAPTATION "adaptation"  
*adaption label.*
- #define LABEL\_ALGORITHM "algorithm"  
*algoritm label.*
- #define LABEL\_OPTIMIZE "optimize"  
*optimize label.*
- #define LABEL\_COORDINATES "coordinates"  
*coordinates label.*
- #define LABEL\_DIRECTION "direction"  
*direction label.*
- #define LABEL\_EUCLIDIAN "euclidian"  
*euclidian label.*
- #define LABEL\_EVALUATOR "evaluator"  
*evaluator label.*
- #define LABEL\_EXPERIMENT "experiment"  
*experiment label.*
- #define LABEL\_EXPERIMENTS "experiments"  
*experiment label.*
- #define LABEL\_GENETIC "genetic"  
*genetic label.*
- #define LABEL\_MINIMUM "minimum"  
*minimum label.*
- #define LABEL\_MAXIMUM "maximum"  
*maximum label.*
- #define LABEL\_MONTE\_CARLO "Monte-Carlo"  
*Monte-Carlo label.*
- #define LABEL\_MUTATION "mutation"  
*mutation label.*
- #define LABEL\_NAME "name"  
*name label.*
- #define LABEL\_NBEST "nbest"  
*nbest label.*
- #define LABEL\_NBITS "nbits"  
*nbits label.*
- #define LABEL\_NESTIMATES "nestimates"  
*nestimates label.*
- #define LABEL\_NGENERATIONS "ngenerations"  
*ngenerations label.*
- #define LABEL\_NITERATIONS "niterations"  
*niterations label.*
- #define LABEL\_NORM "norm"  
*norm label.*
- #define LABEL\_NPOPULATION "npopulation"  
*npopulation label.*



- #define LABEL\_NSIMULATIONS "nsimulations"  
*nsimulations label.*
- #define LABEL\_NSTEPS "nsteps"  
*nsteps label.*
- #define LABEL\_NSWEEPS "nsweeps"  
*nsweeps label.*
- #define LABEL\_P "p"  
*p label.*
- #define LABEL\_PRECISION "precision"  
*precision label.*
- #define LABEL\_RANDOM "random"  
*random label.*
- #define LABEL\_RELAXATION "relaxation"  
*relaxation label.*
- #define LABEL\_REPRODUCTION "reproduction"  
*reproduction label.*
- #define LABEL\_RESULT\_FILE "result\_file"  
*result\_file label.*
- #define LABEL\_SIMULATOR "simulator"  
*simulator label.*
- #define LABEL\_SEED "seed"  
*seed label.*
- #define LABEL\_STEP "step"  
*step label.*
- #define LABEL\_SWEEP "sweep"  
*sweep label.*
- #define LABEL\_TAXICAB "taxicab"  
*taxicab label.*
- #define LABEL\_TEMPLATE1 "template1"  
*template1 label.*
- #define LABEL\_TEMPLATE2 "template2"  
*template2 label.*
- #define LABEL\_TEMPLATE3 "template3"  
*template3 label.*
- #define LABEL\_TEMPLATE4 "template4"  
*template4 label.*
- #define LABEL\_TEMPLATE5 "template5"  
*template5 label.*
- #define LABEL\_TEMPLATE6 "template6"  
*template6 label.*
- #define LABEL\_TEMPLATE7 "template7"  
*template7 label.*
- #define LABEL\_TEMPLATE8 "template8"  
*template8 label.*
- #define LABEL\_THRESHOLD "threshold"  
*threshold label.*
- #define LABEL\_TOLERANCE "tolerance"  
*tolerance label.*
- #define LABEL\_VARIABLE "variable"  
*variable label.*
- #define LABEL\_VARIABLES "variables"

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

## Enumerations

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

### 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 128 of file [config.h](#).

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

## 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 // Gettext simplification
00037 #define _(string) (gettext(string))
00038
00039 // Array sizes
00040
00041 #define MAX_NINPUTS 8
00042 #define NALGORITHMS 3
00043 #define NDIRECTIONS 2
00044 #define NNORMS 4
00045 #define NPRECISIONS 15
00046
00047 // Default choices
00048 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00049 #define DEFAULT_RANDOM_SEED 7007
00050 #define DEFAULT_RELAXATION 1.
00051
00052 // Interface labels
00053
00054 #define LOCALE_DIR "locales"
00055 #define PROGRAM_INTERFACE "mpcotool"
00056
00057 // Labels
00058
00059 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00060 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00061 #define LABEL_ADAPTATION "adaptation"
00062 #define LABEL_ALGORITHM "algorithm"
00063 #define LABEL_OPTIMIZE "optimize"
00064 #define LABEL_COORDINATES "coordinates"
00065 #define LABEL_DIRECTION "direction"
00066 #define LABEL_EUCLIDIAN "euclidian"
00067 #define LABEL_EVALUATOR "evaluator"
00068 #define LABEL_EXPERIMENT "experiment"
00069 #define LABEL_EXPERIMENTS "experiments"
00070 #define LABEL_GENETIC "genetic"
00071 #define LABEL_MINIMUM "minimum"
00072 #define LABEL_MAXIMUM "maximum"
00073 #define LABEL_MONTE_CARLO "Monte-Carlo"
00074 #define LABEL_MUTATION "mutation"
00075 #define LABEL_NAME "name"
00076 #define LABEL_NBEST "nbest"
00077 #define LABEL_NBITS "nbits"
00078 #define LABEL_NESTIMATES "nestimates"
00079 #define LABEL_NGENERATIONS "ngenerations"
00080 #define LABEL_NITERATIONS "niterations"
00081 #define LABEL_NORM "norm"
00082 #define LABEL_NPOPULATION "npopulation"
00083 #define LABEL_NSIMULATIONS "nsimulations"
00084 #define LABEL_NSTEPS "nsteps"
00085 #define LABEL_NSWEEPS "nsweeps"
00086 #define LABEL_P "p"
00087 #define LABEL_PRECISION "precision"

```

```

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

```

### 5.3 experiment.c File Reference

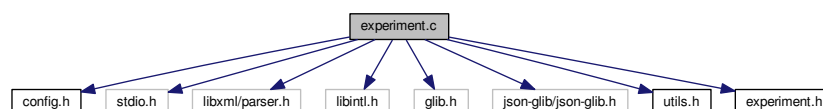
Source file to define the experiment data.

```

#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>	<a href="#">Experiment</a> 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", _("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128                 experiment->name, 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>	<a href="#">Experiment</a> 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>	<a href="#">Experiment</a> 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>	<a href="#">Experiment</a> 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 254 of file [experiment.c](#).

```

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

```

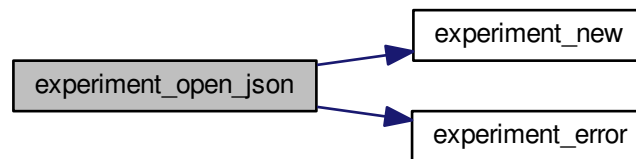
```

00284     experiment->weight
00285     = json_object_get_float_with_default (object,
00286     LABEL_WEIGHT, 1.,
00287     &error_code);
00288     if (error_code)
00289     {
00289         experiment_error (experiment, _("bad weight"));
00290         goto exit_on_error;
00291     }
00292 #if DEBUG_EXPERIMENT
00293     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294 #endif
00295     name = json_object_get_string_member (object, template[0]);
00296     if (name)
00297     {
00298 #if DEBUG_EXPERIMENT
00299         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300         name, template[0]);
00301 #endif
00302         ++experiment->ninputs;
00303     }
00304     else
00305     {
00306         experiment_error (experiment, _("no template"));
00307         goto exit_on_error;
00308     }
00309     experiment->template[0] = g_strdup (name);
00310     for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312 #if DEBUG_EXPERIMENT
00313         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314 #endif
00315         if (json_object_get_member (object, template[i]))
00316         {
00317             if (ninputs && ninputs <= i)
00318             {
00319                 experiment_error (experiment, _("bad templates number"));
00320                 goto exit_on_error;
00321             }
00322             name = json_object_get_string_member (object, template[i]);
00323 #if DEBUG_EXPERIMENT
00324             fprintf (stderr,
00325             "experiment_open_json: experiment=%s template%u=%s\n",
00326             experiment->nexperiments, name, template[i]);
00327 #endif
00328             experiment->template[i] = g_strdup (name);
00329             ++experiment->ninputs;
00330         }
00331         else if (ninputs && ninputs > i)
00332         {
00333             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334             experiment_error (experiment, buffer);
00335             goto exit_on_error;
00336         }
00337         else
00338             break;
00339     }
00340 #if DEBUG_EXPERIMENT
00341     fprintf (stderr, "experiment_open_json: end\n");
00342 #endif
00343     return 1;
00344 }
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>	<a href="#">Experiment</a> 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](#).

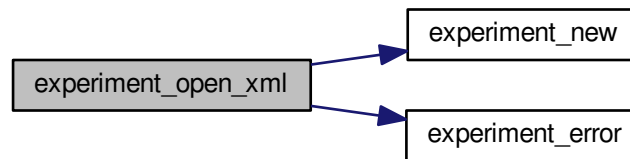
```
00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     =
```

```

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

```

Here is the call graph for this function:



### 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", _("Experiment"), message);
00098     else
00099         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00100                 experiment->name, 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];
00109     int error_code;
00110     unsigned int i;
00111     #if DEBUG_EXPERIMENT

```

```

00153     fprintf (stderr, "experiment_open_xml: start\n");
00154 #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166 #if DEBUG_EXPERIMENT
00167     fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168 #endif
00169     experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
00172     LABEL_WEIGHT, 1.,
00173     &error_code);
00174     if (error_code)
00175     {
00176         experiment_error (experiment, _("bad weight"));
00177         goto exit_on_error;
00178     }
00179 #if DEBUG_EXPERIMENT
00180     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00181 #endif
00182     experiment->template[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00184     if (experiment->template[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187             fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00188             experiment->name, template[0]);
00189         #endif
00190         ++experiment->ninputs;
00191     }
00192     else
00193     {
00194         experiment_error (experiment, _("no template"));
00195         goto exit_on_error;
00196     }
00197     for (i = 1; i < MAX_NINPUTS; ++i)
00198     {
00199         #if DEBUG_EXPERIMENT
00200             fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) template[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->template[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00211             #if DEBUG_EXPERIMENT
00212                 fprintf (stderr,
00213                 "experiment_open_xml: experiment=%s template%u=%s\n",
00214                 experiment->name, experiment->nexperiments,
00215                 experiment->template[i]);
00216             #endif
00217             ++experiment->ninputs;
00218         }
00219         else if (ninputs && ninputs > i)
00220         {
00221             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00222             experiment_error (experiment, buffer);
00223             goto exit_on_error;
00224         }
00225         else
00226             break;
00227     }
00228 #if DEBUG_EXPERIMENT
00229     fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231     return 1;
00232
00233 exit_on_error:
00234     experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236     fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238     return 0;

```

```

00239 }
00240
00253 int
00254 experiment_open_json (Experiment * experiment, JsonNode * node,
00255                      unsigned int ninputs)
00256 {
00257     char buffer[64];
00258     JsonObject *object;
00259     const char *name;
00260     int error_code;
00261     unsigned int i;
00262
00263     #if DEBUG_EXPERIMENT
00264     fprintf (stderr, "experiment_open_json: start\n");
00265     #endif
00266
00267     // Resetting experiment data
00268     experiment_new (experiment);
00269
00270     // Getting JSON object
00271     object = json_node_get_object (node);
00272
00273     // Reading the experimental data
00274     name = json_object_get_string_member (object, LABEL_NAME);
00275     if (!name)
00276     {
00277         experiment_error (experiment, _("no data file name"));
00278         goto exit_on_error;
00279     }
00280     experiment->name = g_strdup (name);
00281     #if DEBUG_EXPERIMENT
00282     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00283     #endif
00284     experiment->weight
00285     = json_object_get_float_with_default (object,
00286     LABEL_WEIGHT, 1.,
00287     &error_code);
00288     if (error_code)
00289     {
00289         experiment_error (experiment, _("bad weight"));
00290         goto exit_on_error;
00291     }
00292     #if DEBUG_EXPERIMENT
00293     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00294     #endif
00295     name = json_object_get_string_member (object, template[0]);
00296     if (name)
00297     {
00298         #if DEBUG_EXPERIMENT
00299         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00300         name, template[0]);
00301         #endif
00302         ++experiment->ninputs;
00303     }
00304     else
00305     {
00306         experiment_error (experiment, _("no template"));
00307         goto exit_on_error;
00308     }
00309     experiment->template[0] = g_strdup (name);
00310     for (i = 1; i < MAX_NINPUTS; ++i)
00311     {
00312         #if DEBUG_EXPERIMENT
00313         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314         #endif
00315         if (json_object_get_member (object, template[i]))
00316         {
00317             if (ninputs && ninputs <= i)
00318             {
00319                 experiment_error (experiment, _("bad templates number"));
00320                 goto exit_on_error;
00321             }
00322             name = json_object_get_string_member (object, template[i]);
00323             #if DEBUG_EXPERIMENT
00324             fprintf (stderr,
00325             "experiment_open_json: experiment=%s template%u=%s\n",
00326             experiment->nexperiments, name, template[i]);
00327             #endif
00328             experiment->template[i] = g_strdup (name);
00329             ++experiment->ninputs;
00330         }
00331         else if (ninputs && ninputs > i)
00332         {
00333             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334             experiment_error (experiment, buffer);
00335             goto exit_on_error;
00336         }
00337     }

```

```

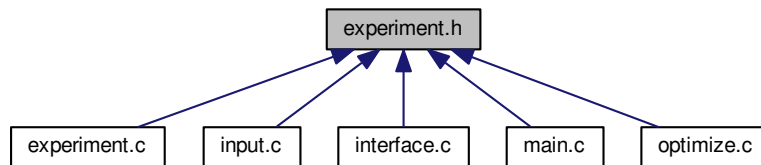
00337         else
00338             break;
00339     }
00340
00341     #if DEBUG_EXPERIMENT
00342     fprintf (stderr, "experiment_open_json: end\n");
00343     #endif
00344     return 1;
00345
00346 exit_on_error:
00347     experiment_free (experiment, INPUT_TYPE_JSON);
00348     #if DEBUG_EXPERIMENT
00349     fprintf (stderr, "experiment_open_json: end\n");
00350     #endif
00351     return 0;
00352 }

```

## 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 XML node.

## Variables

- `const char * template [MAX\_NINPUTS]`  
*Array of `xmlChar` strings with template labels.*

### 5.5.1 Detailed Description

Header file to define the experiment data.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.5.2 Function Documentation

#### 5.5.2.1 `experiment_error()`

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

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

#### Parameters

<i><a href="#">experiment</a></i>	<a href="#">Experiment</a> struct.
<i><a href="#">message</a></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", _("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", _("Experiment"),
00128                 experiment->name, 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>	<a href="#">Experiment</a> 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>	<a href="#">Experiment</a> 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>	<a href="#">Experiment</a> 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 254 of file [experiment.c](#).

```

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

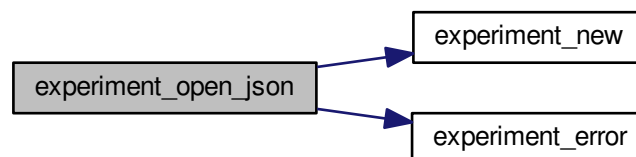
```

```

00313     fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00314 #endif
00315     if (json_object_get_member (object, template[i]))
00316     {
00317         if (ninputs && ninputs <= i)
00318         {
00319             experiment_error (experiment, _("bad templates number"));
00320             goto exit_on_error;
00321         }
00322         name = json_object_get_string_member (object, template[i]);
00323 #if DEBUG_EXPERIMENT
00324         fprintf (stderr,
00325             "experiment_open_json: experiment=%s template%u=%s\n",
00326             experiment->nexperiments, name, template[i]);
00327 #endif
00328         experiment->template[i] = g_strdup (name);
00329         ++experiment->ninputs;
00330     }
00331     else if (ninputs && ninputs > i)
00332     {
00333         snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00334         experiment_error (experiment, buffer);
00335         goto exit_on_error;
00336     }
00337     else
00338         break;
00339 }
00340
00341 #if DEBUG_EXPERIMENT
00342 fprintf (stderr, "experiment_open_json: end\n");
00343 #endif
00344 return 1;
00345
00346 exit_on_error:
00347     experiment_free (experiment, INPUT_TYPE_JSON);
00348 #if DEBUG_EXPERIMENT
00349 fprintf (stderr, "experiment_open_json: end\n");
00350 #endif
00351 return 0;
00352 }

```

Here is the call graph for this function:



### 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>	<a href="#">Experiment</a> 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, _("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     =
00171     xml_node_get_float_with_default (node, (const xmlChar *)
00172     LABEL_WEIGHT, 1.,
00173     &error_code);
00174     if (error_code)
00175     {
00176         experiment_error (experiment, _("bad weight"));
00177         goto exit_on_error;
00178     }
00179     #if DEBUG_EXPERIMENT
00180         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00181     #endif
00182     experiment->template[0]
00183     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00184     if (experiment->template[0])
00185     {
00186         #if DEBUG_EXPERIMENT
00187             fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00188             experiment->name, template[0]);
00189         #endif
00190         ++experiment->ninputs;
00191     }
00192     else
00193     {
00194         experiment_error (experiment, _("no template"));
00195         goto exit_on_error;
00196     }
00197     for (i = 1; i < MAX_NINPUTS; ++i)
00198     {
00199         #if DEBUG_EXPERIMENT
00200             fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00201         #endif
00202         if (xmlHasProp (node, (const xmlChar *) template[i]))
00203         {
00204             if (ninputs && ninputs <= i)
00205             {
00206                 experiment_error (experiment, _("bad templates number"));
00207                 goto exit_on_error;
00208             }
00209             experiment->template[i]
00210             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00211             #if DEBUG_EXPERIMENT
00212                 fprintf (stderr,
00213                 "experiment_open_xml: experiment=%s template%u=%s\n",
00214                 experiment->nexperiments, experiment->name,
00215                 experiment->template[i]);
00216             #endif
00217             ++experiment->ninputs;
00218         }
00219         else if (ninputs && ninputs > i)
00220         {
00221             snprintf (buffer, 64, "%s%u", _("no template"), i + 1);
00222             experiment_error (experiment, buffer);
00223             goto exit_on_error;
00224         }
00225     }

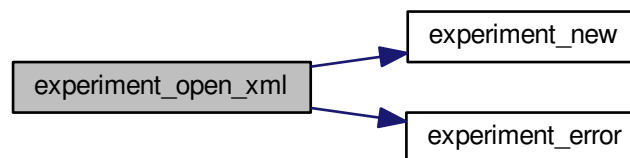
```

```

00224         else
00225             break;
00226     }
00227
00228 #if DEBUG_EXPERIMENT
00229     fprintf (stderr, "experiment_open_xml: end\n");
00230 #endif
00231     return 1;
00232
00233 exit_on_error:
00234     experiment_free (experiment, INPUT_TYPE_XML);
00235 #if DEBUG_EXPERIMENT
00236     fprintf (stderr, "experiment_open_xml: end\n");
00237 #endif
00238     return 0;
00239 }

```

Here is the call graph for this function:



## 5.6 experiment.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
00037     char *name;
00038     char *template[MAX_NINPUTS];

```

```

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", _("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.
-----------------	-----------------------

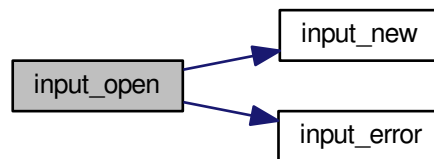
## Returns

1\_on\_success, 0\_on\_error.

Definition at line 952 of file [input.c](#).

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

Here is the call graph for this function:



### 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 562 of file [input.c](#).

```

00563 {
00564     JsonNode *node, *child;
00565     JsonObject *object;
00566     JsonArray *array;
00567     const char *buffer;
00568     int error_code;
00569     unsigned int i, n;
00570
00571     #if DEBUG_INPUT
00572     fprintf (stderr, "input_open_json: start\n");
00573     #endif
00574
00575     // Resetting input data
00576     input->type = INPUT_TYPE_JSON;
00577
00578     // Getting the root node
00579     #if DEBUG_INPUT
00580     fprintf (stderr, "input_open_json: getting the root node\n");
00581     #endif
00582     node = json_parser_get_root (parser);
00583     object = json_node_get_object (node);
00584
00585     // Getting result and variables file names
00586     if (!input->result)
00587     {
00588         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589         if (!buffer)
00590             buffer = result_name;
00591         input->result = g_strdup (buffer);
00592     }
  
```

```

00593     else
00594         input->result = g_strdup (result_name);
00595     if (!input->variables)
00596     {
00597         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598         if (!buffer)
00599             buffer = variables_name;
00600         input->variables = g_strdup (buffer);
00601     }
00602     else
00603         input->variables = g_strdup (variables_name);
00604
00605     // Opening simulator program name
00606     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607     if (!buffer)
00608     {
00609         input_error (_("Bad simulator program"));
00610         goto exit_on_error;
00611     }
00612     input->simulator = g_strdup (buffer);
00613
00614     // Opening evaluator program name
00615     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616     if (buffer)
00617         input->evaluator = g_strdup (buffer);
00618
00619     // Obtaining pseudo-random numbers generator seed
00620     input->seed
00621     = json_object_get_uint_with_default (object,
00622     LABEL_SEED,
00623     DEFAULT_RANDOM_SEED, &error_code);
00624     if (error_code)
00625     {
00626         input_error (_("Bad pseudo-random numbers generator seed"));
00627         goto exit_on_error;
00628     }
00629     // Opening algorithm
00630     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633         input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635         // Obtaining simulations number
00636         input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00638 );
00639         if (error_code)
00640         {
00641             input_error (_("Bad simulations number"));
00642             goto exit_on_error;
00643         }
00644     else if (!strcmp (buffer, LABEL_SWEEP))
00645         input->algorithm = ALGORITHM_SWEEP;
00646     else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648         input->algorithm = ALGORITHM_GENETIC;
00649
00650         // Obtaining population
00651         if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653             input->nsimulations
00654             = json_object_get_uint (object,
00655 LABEL_NPOPULATION, &error_code);
00656             if (error_code || input->nsimulations < 3)
00657             {
00658                 input_error (_("Invalid population number"));
00659                 goto exit_on_error;
00660             }
00661         else
00662         {
00663             input_error (_("No population number"));
00664             goto exit_on_error;
00665         }
00666
00667         // Obtaining generations
00668         if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670             input->niterations
00671             = json_object_get_uint (object,
00672 LABEL_NGENERATIONS, &error_code);
00673             if (error_code || !input->niterations)
00674             {
00675                 input_error (_("Invalid generations number"));
00676                 goto exit_on_error;

```

```

00676         }
00677     }
00678     else
00679     {
00680         input_error (_("No generations number"));
00681         goto exit_on_error;
00682     }
00683
00684     // Obtaining mutation probability
00685     if (json_object_get_member (object, LABEL_MUTATION))
00686     {
00687         input->mutation_ratio
00688         = json_object_get_float (object, LABEL_MUTATION, &error_code
00689 );
00689         if (error_code || input->mutation_ratio < 0.
00690             || input->mutation_ratio >= 1.)
00691         {
00692             input_error (_("Invalid mutation probability"));
00693             goto exit_on_error;
00694         }
00695     }
00696     else
00697     {
00698         input_error (_("No mutation probability"));
00699         goto exit_on_error;
00700     }
00701
00702     // Obtaining reproduction probability
00703     if (json_object_get_member (object, LABEL_REPRODUCTION))
00704     {
00705         input->reproduction_ratio
00706         = json_object_get_float (object,
00707 LABEL_REPRODUCTION, &error_code);
00707         if (error_code || input->reproduction_ratio < 0.
00708             || input->reproduction_ratio >= 1.0)
00709         {
00710             input_error (_("Invalid reproduction probability"));
00711             goto exit_on_error;
00712         }
00713     }
00714     else
00715     {
00716         input_error (_("No reproduction probability"));
00717         goto exit_on_error;
00718     }
00719
00720     // Obtaining adaptation probability
00721     if (json_object_get_member (object, LABEL_ADAPTATION))
00722     {
00723         input->adaptation_ratio
00724         = json_object_get_float (object,
00725 LABEL_ADAPTATION, &error_code);
00725         if (error_code || input->adaptation_ratio < 0.
00726             || input->adaptation_ratio >= 1.)
00727         {
00728             input_error (_("Invalid adaptation probability"));
00729             goto exit_on_error;
00730         }
00731     }
00732     else
00733     {
00734         input_error (_("No adaptation probability"));
00735         goto exit_on_error;
00736     }
00737
00738     // Checking survivals
00739     i = input->mutation_ratio * input->nsimulations;
00740     i += input->reproduction_ratio * input->
00741 nsimulations;
00741     i += input->adaptation_ratio * input->
00742 nsimulations;
00742     if (i > input->nsimulations - 2)
00743     {
00744         input_error
00745         (_("No enough survival entities to reproduce the population"));
00746         goto exit_on_error;
00747     }
00748 }
00749 else
00750 {
00751     input_error (_("Unknown algorithm"));
00752     goto exit_on_error;
00753 }
00754
00755 if (input->algorithm == ALGORITHM_MONTE_CARLO
00756     || input->algorithm == ALGORITHM_SWEEP)
00757 {

```

```

00758
00759 // Obtaining iterations number
00760 input->niterations
00761 = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00762 if (error_code == 1)
00763     input->niterations = 1;
00764 else if (error_code)
00765 {
00766     input_error (_("Bad iterations number"));
00767     goto exit_on_error;
00768 }
00769
00770 // Obtaining best number
00771 input->nbest
00772 = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
                                &error_code);
00773
00774 if (error_code || !input->nbest)
00775 {
00776     input_error (_("Invalid best number"));
00777     goto exit_on_error;
00778 }
00779
00780 // Obtaining tolerance
00781 input->tolerance
00782 = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
                                &error_code);
00783
00784 if (error_code || input->tolerance < 0.)
00785 {
00786     input_error (_("Invalid tolerance"));
00787     goto exit_on_error;
00788 }
00789
00790 // Getting direction search method parameters
00791 if (json_object_get_member (object, LABEL_NSTEPS))
00792 {
00793     input->nsteps
00794     = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00795     if (error_code)
00796     {
00797         input_error (_("Invalid steps number"));
00798         goto exit_on_error;
00799     }
00800     buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00801     if (!strcmp (buffer, LABEL_COORDINATES))
00802         input->direction = DIRECTION_METHOD_COORDINATES;
00803     else if (!strcmp (buffer, LABEL_RANDOM))
00804     {
00805         input->direction = DIRECTION_METHOD_RANDOM;
00806         input->nestimates
00807         =
00808         json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00809         if (error_code || !input->nestimates)
00810         {
00811             input_error (_("Invalid estimates number"));
00812             goto exit_on_error;
00813         }
00814     }
00815     else
00816     {
00817         input_error
00818         (_("Unknown method to estimate the direction search"));
00819         goto exit_on_error;
00820     }
00821     input->relaxation
00822     = json_object_get_float_with_default (object,
LABEL_RELAXATION,
                                DEFAULT_RELAXATION,
                                &error_code);
00823
00824     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00825     {
00826         input_error (_("Invalid relaxation parameter"));
00827         goto exit_on_error;
00828     }
00829 }
00830 else
00831     input->nsteps = 0;
00832
00833 // Obtaining the threshold
00834 input->threshold
00835 = json_object_get_float_with_default (object,
LABEL_THRESHOLD, 0.,
                                &error_code);
00836
00837

```

```

00838     if (error_code)
00839     {
00840         input_error (_("Invalid threshold"));
00841         goto exit_on_error;
00842     }
00843
00844     // Reading the experimental data
00845     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846     n = json_array_get_length (array);
00847     input->experiment = (Experiment *) g_malloc (n * sizeof (
00848         Experiment));
00849     for (i = 0; i < n; ++i)
00850     {
00851         #if DEBUG_INPUT
00852             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00853                 input->nexperiments);
00854         #endif
00855         child = json_array_get_element (array, i);
00856         if (!input->nexperiments)
00857         {
00858             if (!experiment_open_json (input->experiment, child, 0))
00859                 goto exit_on_error;
00860         }
00861         else
00862         {
00863             if (!experiment_open_json (input->experiment +
00864                 input->nexperiments,
00865                                     child, input->experiment->
00866                 ninputs))
00867                 goto exit_on_error;
00868             ++input->nexperiments;
00869         }
00870         #if DEBUG_INPUT
00871             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00872                 input->nexperiments);
00873         #endif
00874         if (!input->nexperiments)
00875         {
00876             input_error (_("No optimization experiments"));
00877             goto exit_on_error;
00878         }
00879
00880         // Reading the variables data
00881         array = json_object_get_array_member (object, LABEL_VARIABLES);
00882         n = json_array_get_length (array);
00883         input->variable = (Variable *) g_malloc (n * sizeof (
00884             Variable));
00885         for (i = 0; i < n; ++i)
00886         {
00887             #if DEBUG_INPUT
00888                 fprintf (stderr, "input_open_json: nvariables=%u\n", input->
00889                     nvariables);
00890             #endif
00891             child = json_array_get_element (array, i);
00892             if (!variable_open_json (input->variable +
00893                 input->nvariables, child,
00894                                     input->algorithm, input->
00895                 nsteps))
00896                 goto exit_on_error;
00897             ++input->nvariables;
00898         }
00899         if (!input->nvariables)
00900         {
00901             input_error (_("No optimization variables"));
00902             goto exit_on_error;
00903         }
00904
00905         // Obtaining the error norm
00906         if (json_object_get_member (object, LABEL_NORM))
00907         {
00908             buffer = json_object_get_string_member (object, LABEL_NORM);
00909             if (!strcmp (buffer, LABEL_EUCLIDIAN))
00910                 input->norm = ERROR_NORM_EUCLIDIAN;
00911             else if (!strcmp (buffer, LABEL_MAXIMUM))
00912                 input->norm = ERROR_NORM_MAXIMUM;
00913             else if (!strcmp (buffer, LABEL_P))
00914             {
00915                 input->norm = ERROR_NORM_P;
00916                 input->p = json_object_get_float (object,
00917                     LABEL_P, &error_code);
00918                 if (!error_code)
00919                 {
00920                     input_error (_("Bad P parameter"));
00921                     goto exit_on_error;
00922                 }
00923             }
00924         }
00925     }

```

```

00917         else if (!strcmp (buffer, LABEL_TAXICAB))
00918             input->norm = ERROR_NORM_TAXICAB;
00919         else
00920         {
00921             input_error (_("Unknown error norm"));
00922             goto exit_on_error;
00923         }
00924     }
00925     else
00926         input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928     // Closing the JSON document
00929     g_object_unref (parser);
00930
00931     #if DEBUG_INPUT
00932     fprintf (stderr, "input_open_json: end\n");
00933     #endif
00934     return 1;
00935
00936 exit_on_error:
00937     g_object_unref (parser);
00938     #if DEBUG_INPUT
00939     fprintf (stderr, "input_open_json: end\n");
00940     #endif
00941     return 0;
00942 }

```

Here is the call graph for this function:



#### 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 (_("Bad root XML node"));
00163     goto exit_on_error;
00164 }
00165
00166 // Getting result and variables file names
00167 if (!input->result)
00168 {
00169     input->result =
00170         (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171     if (!input->result)
00172         input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173 }
00174 if (!input->variables)
00175 {
00176     input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178     if (!input->variables)
00179         input->variables =
00180             (char *) xmlStrdup ((const xmlChar *) variables_name);
00181 }
00182
00183 // Opening simulator program name
00184 input->simulator =
00185     (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186 if (!input->simulator)
00187 {
00188     input_error (_("Bad simulator program"));
00189     goto exit_on_error;
00190 }
00191
00192 // Opening evaluator program name
00193 input->evaluator =
00194     (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196 // Obtaining pseudo-random numbers generator seed
00197 input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                DEFAULT_RANDOM_SEED, &error_code);
00199 if (error_code)
00200 {
00201     input_error (_("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 (_("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 (_("Invalid population number"));
00237             goto exit_on_error;
00238         }
00239     }
00240     else
00241     {
00242         input_error (_("No population number"));
00243         goto exit_on_error;
00244     }
00245
00246     // Obtaining generations
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248     {
00249         input->niterations
00250         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NGENERATIONS,
00251                             &error_code);
00252         if (error_code || !input->niterations)
00253         {
00254             input_error (_("Invalid generations number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         input_error (_("No generations number"));
00261         goto exit_on_error;
00262     }
00263
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268         = xml_node_get_float (node, (const xmlChar *)
LABEL_MUTATION,
00269                             &error_code);
00270         if (error_code || input->mutation_ratio < 0.
00271             || input->mutation_ratio >= 1.)
00272         {
00273             input_error (_("Invalid mutation probability"));
00274             goto exit_on_error;
00275         }
00276     }
00277     else
00278     {
00279         input_error (_("No mutation probability"));
00280         goto exit_on_error;
00281     }
00282
00283     // Obtaining reproduction probability
00284     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00285     {
00286         input->reproduction_ratio
00287         = xml_node_get_float (node, (const xmlChar *)
LABEL_REPRODUCTION,
00288                             &error_code);
00289         if (error_code || input->reproduction_ratio < 0.
00290             || input->reproduction_ratio >= 1.0)
00291         {
00292             input_error (_("Invalid reproduction probability"));
00293             goto exit_on_error;
00294         }
00295     }
00296     else
00297     {
00298         input_error (_("No reproduction probability"));
00299         goto exit_on_error;
00300     }
00301
00302     // Obtaining adaptation probability
00303     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304     {
00305         input->adaptation_ratio
00306         = xml_node_get_float (node, (const xmlChar *)
LABEL_ADAPTATION,
00307                             &error_code);
00308         if (error_code || input->adaptation_ratio < 0.
00309             || input->adaptation_ratio >= 1.)
00310         {
00311             input_error (_("Invalid adaptation probability"));

```

```

00312         goto exit_on_error;
00313     }
00314 }
00315 else
00316 {
00317     input_error (_("No adaptation probability"));
00318     goto exit_on_error;
00319 }
00320
00321 // Checking survivals
00322 i = input->mutation_ratio * input->nsimulations;
00323 i += input->reproduction_ratio * input->
nsimulations;
00324 i += input->adaptation_ratio * input->
nsimulations;
00325 if (i > input->nsimulations - 2)
00326 {
00327     input_error
00328         (_("No enough survival entities to reproduce the population"));
00329     goto exit_on_error;
00330 }
00331 }
00332 else
00333 {
00334     input_error (_("Unknown algorithm"));
00335     goto exit_on_error;
00336 }
00337 xmlFree (buffer);
00338 buffer = NULL;
00339
00340 if (input->algorithm == ALGORITHM_MONTE_CARLO
00341 || input->algorithm == ALGORITHM_SWEEP)
00342 {
00343
00344     // Obtaining iterations number
00345     input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
00347                             &error_code);
00348     if (error_code == 1)
00349         input->niterations = 1;
00350     else if (error_code)
00351     {
00352         input_error (_("Bad iterations number"));
00353         goto exit_on_error;
00354     }
00355
00356     // Obtaining best number
00357     input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00359                                         1, &error_code);
00360     if (error_code || !input->nbest)
00361     {
00362         input_error (_("Invalid best number"));
00363         goto exit_on_error;
00364     }
00365     if (input->nbest > input->nsimulations)
00366     {
00367         input_error (_("Best number higher than simulations number"));
00368         goto exit_on_error;
00369     }
00370
00371     // Obtaining tolerance
00372     input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                           (const xmlChar *) LABEL_TOLERANCE,
00375                                           0., &error_code);
00376     if (error_code || input->tolerance < 0.)
00377     {
00378         input_error (_("Invalid tolerance"));
00379         goto exit_on_error;
00380     }
00381
00382     // Getting direction search method parameters
00383     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384     {
00385         input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *)
LABEL_NSTEPS,
00387                               &error_code);
00388         if (error_code)
00389         {
00390             input_error (_("Invalid steps number"));
00391             goto exit_on_error;
00392         }
00393         buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);

```

```

00394         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395             input->direction = DIRECTION_METHOD_COORDINATES;
00396         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397         {
00398             input->direction = DIRECTION_METHOD_RANDOM;
00399             input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                                     &error_code);
00402             if (error_code || !input->nestimates)
00403             {
00404                 input_error (_("Invalid estimates number"));
00405                 goto exit_on_error;
00406             }
00407         }
00408         else
00409         {
00410             input_error
00411                 (_("Unknown method to estimate the direction search"));
00412             goto exit_on_error;
00413         }
00414         xmlFree (buffer);
00415         buffer = NULL;
00416         input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                                (const xmlChar *)
LABEL_RELAXATION,
00419                                                DEFAULT_RELAXATION,
00420                                                &error_code);
00421         if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00422         {
00423             input_error (_("Invalid relaxation parameter"));
00424             goto exit_on_error;
00425         }
00426     }
00427     else
00428         input->nsteps = 0;
00429 }
00430 // Obtaining the threshold
00431 input->threshold =
00432     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00433                                     0., &error_code);
00434     if (error_code)
00435     {
00436         input_error (_("Invalid threshold"));
00437         goto exit_on_error;
00438     }
00439 }
00440 // Reading the experimental data
00441 for (child = node->children; child; child = child->next)
00442 {
00443     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00444         break;
00445 #if DEBUG_INPUT
00446     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00447             input->nexperiments);
00448 #endif
00449     input->experiment = (Experiment *)
00450         g_realloc (input->experiment,
00451                   (1 + input->nexperiments) * sizeof (
Experiment));
00452     if (!input->nexperiments)
00453     {
00454         if (!experiment_open_xml (input->experiment, child, 0))
00455             goto exit_on_error;
00456     }
00457     else
00458     {
00459         if (!experiment_open_xml (input->experiment +
input->nexperiments,
00460                                 child, input->experiment->
ninputs))
00461             goto exit_on_error;
00462     }
00463     ++input->nexperiments;
00464 #if DEBUG_INPUT
00465     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00466             input->nexperiments);
00467 #endif
00468 }
00469 if (!input->nexperiments)
00470 {
00471     input_error (_("No optimization experiments"));
00472     goto exit_on_error;
00473 }
00474 }

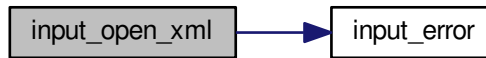
```

```

00475     buffer = NULL;
00476
00477     // Reading the variables data
00478     for (; child; child = child->next)
00479     {
00480 #if DEBUG_INPUT
00481         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00482 #endif
00483         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00484         {
00485             snprintf (buffer2, 64, "%s %u: %s",
00486                     _("Variable"), input->nvariables + 1, _("bad XML node"));
00487             input_error (buffer2);
00488             goto exit_on_error;
00489         }
00490         input->variable = (Variable *)
00491             g_realloc (input->variable,
00492                     (1 + input->nvariables) * sizeof (Variable));
00493         if (!variable_open_xml (input->variable +
00494                               input->nvariables, child,
00495                               input->algorithm, input->nsteps))
00496             goto exit_on_error;
00497         ++input->nvariables;
00498     }
00499     if (!input->nvariables)
00500     {
00501         input_error (_("No optimization variables"));
00502         goto exit_on_error;
00503     }
00504     buffer = NULL;
00505
00506     // Obtaining the error norm
00507     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00508     {
00509         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00510         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00511             input->norm = ERROR_NORM_EUCLIDIAN;
00512         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00513             input->norm = ERROR_NORM_MAXIMUM;
00514         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00515         {
00516             input->norm = ERROR_NORM_P;
00517             input->p
00518                 =
00519                 xml_node_get_float (node, (const xmlChar *)
00520                                     LABEL_P, &error_code);
00521             if (!error_code)
00522             {
00523                 input_error (_("Bad P parameter"));
00524                 goto exit_on_error;
00525             }
00526         }
00527         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00528             input->norm = ERROR_NORM_TAXICAB;
00529         else
00530         {
00531             input_error (_("Unknown error norm"));
00532             goto exit_on_error;
00533         }
00534         xmlFree (buffer);
00535     }
00536     else
00537         input->norm = ERROR_NORM_EUCLIDIAN;
00538
00539     // Closing the XML document
00540     xmlFreeDoc (doc);
00541
00542 #if DEBUG_INPUT
00543     fprintf (stderr, "input_open_xml: end\n");
00544 #endif
00545     return 1;
00546
00547 exit_on_error:
00548     xmlFree (buffer);
00549     xmlFreeDoc (doc);
00550 #if DEBUG_INPUT
00551     fprintf (stderr, "input_open_xml: end\n");
00552 #endif
00553     return 0;
00554 }

```

Here is the call graph for this function:



## 5.8 input.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "utils.h"
00042 #include "experiment.h"
00043 #include "variable.h"
00044 #include "input.h"
00045
00046 #define DEBUG_INPUT 0
00047
00048 Input input[1];
00049
00050 const char *result_name = "result";
00051 const char *variables_name = "variables";
00052
00053 void
00054 input_new ()
00055 {
00056     #if DEBUG_INPUT
00057         fprintf (stderr, "input_new: start\n");
00058     #endif
00059     input->nvariables = input->nexperiments = input->nsteps = 0;
00060     input->simulator = input->evaluator = input->directory = input->
00061     name = NULL;
00062     input->experiment = NULL;
  
```

```

00072     input->variable = NULL;
00073     #if DEBUG_INPUT
00074         fprintf (stderr, "input_new: end\n");
00075     #endif
00076 }
00077
00082 void
00083 input_free ()
00084 {
00085     unsigned int i;
00086     #if DEBUG_INPUT
00087         fprintf (stderr, "input_free: start\n");
00088     #endif
00089     g_free (input->name);
00090     g_free (input->directory);
00091     for (i = 0; i < input->nexperiments; ++i)
00092         experiment_free (input->experiment + i, input->type);
00093     for (i = 0; i < input->nvariables; ++i)
00094         variable_free (input->variable + i, input->type);
00095     g_free (input->experiment);
00096     g_free (input->variable);
00097     if (input->type == INPUT_TYPE_XML)
00098     {
00099         xmlFree (input->evaluator);
00100         xmlFree (input->simulator);
00101         xmlFree (input->result);
00102         xmlFree (input->variables);
00103     }
00104     else
00105     {
00106         g_free (input->evaluator);
00107         g_free (input->simulator);
00108         g_free (input->result);
00109         g_free (input->variables);
00110     }
00111     input->nexperiments = input->nvariables = input->nsteps = 0;
00112     #if DEBUG_INPUT
00113         fprintf (stderr, "input_free: end\n");
00114     #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", _("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
00130
00138 int
00139 input_open_xml (xmlDoc * doc)
00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157         fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (_("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173     }
00174     if (!input->variables)
00175     {

```

```

00176     input->variables =
00177         (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178     if (!input->variables)
00179         input->variables =
00180             (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (_("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed
00198         = xml_node_get_uint_with_default (node, (const xmlChar *)
00199     LABEL_SEED,
00200                                         DEFAULT_RANDOM_SEED, &error_code);
00201     if (error_code)
00202     {
00203         input_error (_("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 (_("Bad simulations number"));
00221             goto exit_on_error;
00222         }
00223     }
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00225     {
00226         input->algorithm = ALGORITHM_SWEEP;
00227     }
00228     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00229     {
00230         input->algorithm = ALGORITHM_GENETIC;
00231
00232         // Obtaining population
00233         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00234         {
00235             input->nsimulations
00236                 = xml_node_get_uint (node, (const xmlChar *)
00237     LABEL_NPOPULATION,
00238                                     &error_code);
00239             if (error_code || input->nsimulations < 3)
00240             {
00241                 input_error (_("Invalid population number"));
00242                 goto exit_on_error;
00243             }
00244         }
00245     }
00246     else
00247     {
00248         input_error (_("No population number"));
00249         goto exit_on_error;
00250     }
00251
00252     // Obtaining generations
00253     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00254     {
00255         input->niterations
00256             = xml_node_get_uint (node, (const xmlChar *)
00257     LABEL_NGENERATIONS,
00258                               &error_code);
00259         if (error_code || !input->niterations)
00260         {
00261             input_error (_("Invalid generations number"));
00262             goto exit_on_error;
00263         }
00264     }
00265     else

```

```

00259     {
00260         input_error (_("No generations number"));
00261         goto exit_on_error;
00262     }
00263
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268         = xml_node_get_float (node, (const xmlChar *)
00269 LABEL_MUTATION,
00270                             &error_code);
00271         if (error_code || input->mutation_ratio < 0.
00272             || input->mutation_ratio >= 1.)
00273         {
00274             input_error (_("Invalid mutation probability"));
00275             goto exit_on_error;
00276         }
00277     }
00278     else
00279     {
00280         input_error (_("No mutation probability"));
00281         goto exit_on_error;
00282     }
00283
00284     // Obtaining reproduction probability
00285     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286     {
00287         input->reproduction_ratio
00288         = xml_node_get_float (node, (const xmlChar *)
00289 LABEL_REPRODUCTION,
00290                             &error_code);
00291         if (error_code || input->reproduction_ratio < 0.
00292             || input->reproduction_ratio >= 1.0)
00293         {
00294             input_error (_("Invalid reproduction probability"));
00295             goto exit_on_error;
00296         }
00297     }
00298     else
00299     {
00300         input_error (_("No reproduction probability"));
00301         goto exit_on_error;
00302     }
00303
00304     // Obtaining adaptation probability
00305     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00306     {
00307         input->adaptation_ratio
00308         = xml_node_get_float (node, (const xmlChar *)
00309 LABEL_ADAPTATION,
00310                             &error_code);
00311         if (error_code || input->adaptation_ratio < 0.
00312             || input->adaptation_ratio >= 1.)
00313         {
00314             input_error (_("Invalid adaptation probability"));
00315             goto exit_on_error;
00316         }
00317     }
00318     else
00319     {
00320         input_error (_("No adaptation probability"));
00321         goto exit_on_error;
00322     }
00323
00324     // Checking survivals
00325     i = input->mutation_ratio * input->nsimulations;
00326     i += input->reproduction_ratio * input->nsimulations;
00327     i += input->adaptation_ratio * input->nsimulations;
00328     if (i > input->nsimulations - 2)
00329     {
00330         input_error
00331         (_("No enough survival entities to reproduce the population"));
00332         goto exit_on_error;
00333     }
00334     else
00335     {
00336         input_error (_("Unknown algorithm"));
00337         goto exit_on_error;
00338     }
00339     xmlFree (buffer);
00340     buffer = NULL;
00341
00342     if (input->algorithm == ALGORITHM_MONTE_CARLO
00343         || input->algorithm == ALGORITHM_SWEEP)
00344     {

```



```

00343
00344 // Obtaining iterations number
00345 input->niterations
00346 = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
00347                      &error_code);
00348 if (error_code == 1)
00349     input->niterations = 1;
00350 else if (error_code)
00351 {
00352     input_error (_("Bad iterations number"));
00353     goto exit_on_error;
00354 }
00355
00356 // Obtaining best number
00357 input->nbest
00358 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00359                                  1, &error_code);
00360 if (error_code || !input->nbest)
00361 {
00362     input_error (_("Invalid best number"));
00363     goto exit_on_error;
00364 }
00365 if (input->nbest > input->nsimulations)
00366 {
00367     input_error (_("Best number higher than simulations number"));
00368     goto exit_on_error;
00369 }
00370
00371 // Obtaining tolerance
00372 input->tolerance
00373 = xml_node_get_float_with_default (node,
00374                                   (const xmlChar *) LABEL_TOLERANCE,
00375                                   0., &error_code);
00376 if (error_code || input->tolerance < 0.)
00377 {
00378     input_error (_("Invalid tolerance"));
00379     goto exit_on_error;
00380 }
00381
00382 // Getting direction search method parameters
00383 if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384 {
00385     input->nsteps =
00386         xml_node_get_uint (node, (const xmlChar *)
LABEL_NSTEPS,
00387                             &error_code);
00388     if (error_code)
00389     {
00390         input_error (_("Invalid steps number"));
00391         goto exit_on_error;
00392     }
00393     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395         input->direction = DIRECTION_METHOD_COORDINATES;
00396     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397     {
00398         input->direction = DIRECTION_METHOD_RANDOM;
00399         input->nestimates
00400         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00401                             &error_code);
00402         if (error_code || !input->nestimates)
00403         {
00404             input_error (_("Invalid estimates number"));
00405             goto exit_on_error;
00406         }
00407     }
00408     else
00409     {
00410         input_error
00411             (_("Unknown method to estimate the direction search"));
00412         goto exit_on_error;
00413     }
00414     xmlFree (buffer);
00415     buffer = NULL;
00416     input->relaxation
00417     = xml_node_get_float_with_default (node,
00418                                       (const xmlChar *)
LABEL_RELAXATION,
00419                                       DEFAULT_RELAXATION,
00420                                       &error_code);
00421     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00422     {
00423         input_error (_("Invalid relaxation parameter"));
00424     }

```

```

00425         goto exit_on_error;
00426     }
00427 }
00428 else
00429     input->nsteps = 0;
00430 }
00431 // Obtaining the threshold
00432 input->threshold =
00433     xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00434                                     0., &error_code);
00435 if (error_code)
00436 {
00437     input_error (_("Invalid threshold"));
00438     goto exit_on_error;
00439 }
00440
00441 // Reading the experimental data
00442 for (child = node->children; child; child = child->next)
00443 {
00444     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00445         break;
00446 #if DEBUG_INPUT
00447     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00448             input->nexperiments);
00449 #endif
00450     input->experiment = (Experiment *)
00451         g_realloc (input->experiment,
00452                   (1 + input->nexperiments) * sizeof (Experiment));
00453     if (!input->nexperiments)
00454     {
00455         if (!experiment_open_xml (input->experiment, child, 0))
00456             goto exit_on_error;
00457     }
00458     else
00459     {
00460         if (!experiment_open_xml (input->experiment + input->
nexperiments,
00461                                 child, input->experiment->ninputs))
00462             goto exit_on_error;
00463     }
00464     ++input->nexperiments;
00465 #if DEBUG_INPUT
00466     fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00467             input->nexperiments);
00468 #endif
00469 }
00470 if (!input->nexperiments)
00471 {
00472     input_error (_("No optimization experiments"));
00473     goto exit_on_error;
00474 }
00475 buffer = NULL;
00476
00477 // Reading the variables data
00478 for (; child; child = child->next)
00479 {
00480 #if DEBUG_INPUT
00481     fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00482 #endif
00483     if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00484     {
00485         snprintf (buffer2, 64, "%s %u: %s",
00486                 _("Variable"), input->nvariables + 1, _("bad XML node"));
00487         input_error (buffer2);
00488         goto exit_on_error;
00489     }
00490     input->variable = (Variable *)
00491         g_realloc (input->variable,
00492                   (1 + input->nvariables) * sizeof (Variable));
00493     if (!variable_open_xml (input->variable + input->
nvariables, child,
00494                             input->algorithm, input->nsteps))
00495         goto exit_on_error;
00496     ++input->nvariables;
00497 }
00498 if (!input->nvariables)
00499 {
00500     input_error (_("No optimization variables"));
00501     goto exit_on_error;
00502 }
00503 buffer = NULL;
00504
00505 // Obtaining the error norm
00506 if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507 {
00508     buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);

```

```

00509     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510         input->norm = ERROR_NORM_EUCLIDIAN;
00511     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512         input->norm = ERROR_NORM_MAXIMUM;
00513     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514     {
00515         input->norm = ERROR_NORM_P;
00516         input->p
00517         =
00518         xml_node_get_float (node, (const xmlChar *)
00519 LABEL_P, &error_code);
00519         if (!error_code)
00520         {
00521             input_error (_("Bad P parameter"));
00522             goto exit_on_error;
00523         }
00524     }
00525     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526         input->norm = ERROR_NORM_TAXICAB;
00527     else
00528     {
00529         input_error (_("Unknown error norm"));
00530         goto exit_on_error;
00531     }
00532     xmlFree (buffer);
00533 }
00534 else
00535     input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537 // Closing the XML document
00538 xmlFreeDoc (doc);
00539
00540 #if DEBUG_INPUT
00541 fprintf (stderr, "input_open_xml: end\n");
00542 #endif
00543 return 1;
00544
00545 exit_on_error:
00546 xmlFree (buffer);
00547 xmlFreeDoc (doc);
00548 #if DEBUG_INPUT
00549 fprintf (stderr, "input_open_xml: end\n");
00550 #endif
00551 return 0;
00552 }
00553
00561 int
00562 input_open_json (JsonParser * parser)
00563 {
00564     JsonNode *node, *child;
00565     JsonObject *object;
00566     JsonArray *array;
00567     const char *buffer;
00568     int error_code;
00569     unsigned int i, n;
00570
00571 #if DEBUG_INPUT
00572 fprintf (stderr, "input_open_json: start\n");
00573 #endif
00574
00575 // Resetting input data
00576 input->type = INPUT_TYPE_JSON;
00577
00578 // Getting the root node
00579 #if DEBUG_INPUT
00580 fprintf (stderr, "input_open_json: getting the root node\n");
00581 #endif
00582 node = json_parser_get_root (parser);
00583 object = json_node_get_object (node);
00584
00585 // Getting result and variables file names
00586 if (!input->result)
00587 {
00588     buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00589     if (!buffer)
00590         buffer = result_name;
00591     input->result = g_strdup (buffer);
00592 }
00593 else
00594     input->result = g_strdup (result_name);
00595 if (!input->variables)
00596 {
00597     buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00598     if (!buffer)
00599         buffer = variables_name;
00600     input->variables = g_strdup (buffer);
00601 }

```

```

00602     else
00603         input->variables = g_strdup (variables_name);
00604
00605     // Opening simulator program name
00606     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00607     if (!buffer)
00608     {
00609         input_error (_("Bad simulator program"));
00610         goto exit_on_error;
00611     }
00612     input->simulator = g_strdup (buffer);
00613
00614     // Opening evaluator program name
00615     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00616     if (buffer)
00617         input->evaluator = g_strdup (buffer);
00618
00619     // Obtaining pseudo-random numbers generator seed
00620     input->seed
00621     = json_object_get_uint_with_default (object,
00622     LABEL_SEED,
00623                                         DEFAULT_RANDOM_SEED, &error_code);
00624     if (error_code)
00625     {
00626         input_error (_("Bad pseudo-random numbers generator seed"));
00627         goto exit_on_error;
00628     }
00629     // Opening algorithm
00630     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00631     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633         input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635         // Obtaining simulations number
00636         input->nsimulations
00637         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00638 );
00639         if (error_code)
00640         {
00641             input_error (_("Bad simulations number"));
00642             goto exit_on_error;
00643         }
00644     else if (!strcmp (buffer, LABEL_SWEEP))
00645         input->algorithm = ALGORITHM_SWEEP;
00646     else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648         input->algorithm = ALGORITHM_GENETIC;
00649
00650         // Obtaining population
00651         if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653             input->nsimulations
00654             = json_object_get_uint (object,
00655     LABEL_NPOPULATION, &error_code);
00656             if (error_code || input->nsimulations < 3)
00657             {
00658                 input_error (_("Invalid population number"));
00659                 goto exit_on_error;
00660             }
00661         else
00662         {
00663             input_error (_("No population number"));
00664             goto exit_on_error;
00665         }
00666
00667         // Obtaining generations
00668         if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670             input->niterations
00671             = json_object_get_uint (object,
00672     LABEL_NGENERATIONS, &error_code);
00673             if (error_code || !input->niterations)
00674             {
00675                 input_error (_("Invalid generations number"));
00676                 goto exit_on_error;
00677             }
00678         else
00679         {
00680             input_error (_("No generations number"));
00681             goto exit_on_error;
00682         }
00683
00684         // Obtaining mutation probability

```

```

00685     if (json_object_get_member (object, LABEL_MUTATION))
00686     {
00687         input->mutation_ratio
00688         = json_object_get_float (object, LABEL_MUTATION, &error_code
);
00689         if (error_code || input->mutation_ratio < 0.
00690             || input->mutation_ratio >= 1.)
00691         {
00692             input_error (_("Invalid mutation probability"));
00693             goto exit_on_error;
00694         }
00695     }
00696     else
00697     {
00698         input_error (_("No mutation probability"));
00699         goto exit_on_error;
00700     }
00701
00702     // Obtaining reproduction probability
00703     if (json_object_get_member (object, LABEL_REPRODUCTION))
00704     {
00705         input->reproduction_ratio
00706         = json_object_get_float (object,
LABEL_REPRODUCTION, &error_code);
00707         if (error_code || input->reproduction_ratio < 0.
00708             || input->reproduction_ratio >= 1.0)
00709         {
00710             input_error (_("Invalid reproduction probability"));
00711             goto exit_on_error;
00712         }
00713     }
00714     else
00715     {
00716         input_error (_("No reproduction probability"));
00717         goto exit_on_error;
00718     }
00719
00720     // Obtaining adaptation probability
00721     if (json_object_get_member (object, LABEL_ADAPTATION))
00722     {
00723         input->adaptation_ratio
00724         = json_object_get_float (object,
LABEL_ADAPTATION, &error_code);
00725         if (error_code || input->adaptation_ratio < 0.
00726             || input->adaptation_ratio >= 1.)
00727         {
00728             input_error (_("Invalid adaptation probability"));
00729             goto exit_on_error;
00730         }
00731     }
00732     else
00733     {
00734         input_error (_("No adaptation probability"));
00735         goto exit_on_error;
00736     }
00737
00738     // Checking survivals
00739     i = input->mutation_ratio * input->nsimulations;
00740     i += input->reproduction_ratio * input->nsimulations;
00741     i += input->adaptation_ratio * input->nsimulations;
00742     if (i > input->nsimulations - 2)
00743     {
00744         input_error
00745         (_("No enough survival entities to reproduce the population"));
00746         goto exit_on_error;
00747     }
00748 }
00749 else
00750 {
00751     input_error (_("Unknown algorithm"));
00752     goto exit_on_error;
00753 }
00754
00755 if (input->algorithm == ALGORITHM_MONTE_CARLO
00756     || input->algorithm == ALGORITHM_SWEEP)
00757 {
00758
00759     // Obtaining iterations number
00760     input->niterations
00761     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00762     if (error_code == 1)
00763         input->niterations = 1;
00764     else if (error_code)
00765     {
00766         input_error (_("Bad iterations number"));
00767         goto exit_on_error;

```

```

00768     }
00769
00770     // Obtaining best number
00771     input->nbest
00772     = json_object_get_uint_with_default (object,
00773     LABEL_NBEST, 1,
00774     &error_code);
00775     if (error_code || !input->nbest)
00776     {
00777         input_error (_("Invalid best number"));
00778         goto exit_on_error;
00779     }
00780
00781     // Obtaining tolerance
00782     input->tolerance
00783     = json_object_get_float_with_default (object,
00784     LABEL_TOLERANCE, 0.,
00785     &error_code);
00786     if (error_code || input->tolerance < 0.)
00787     {
00788         input_error (_("Invalid tolerance"));
00789         goto exit_on_error;
00790     }
00791
00792     // Getting direction search method parameters
00793     if (json_object_get_member (object, LABEL_NSTEPS))
00794     {
00795         input->nsteps
00796         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00797         if (error_code)
00798         {
00799             input_error (_("Invalid steps number"));
00800             goto exit_on_error;
00801         }
00802         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00803         if (!strcmp (buffer, LABEL_COORDINATES))
00804             input->direction = DIRECTION_METHOD_COORDINATES;
00805         else if (!strcmp (buffer, LABEL_RANDOM))
00806         {
00807             input->direction = DIRECTION_METHOD_RANDOM;
00808             input->nestimates
00809             =
00810             json_object_get_uint (object,
00811             LABEL_NESTIMATES, &error_code);
00812             if (error_code || !input->nestimates)
00813             {
00814                 input_error (_("Invalid estimates number"));
00815                 goto exit_on_error;
00816             }
00817         }
00818         else
00819         {
00820             input_error
00821             (_("Unknown method to estimate the direction search"));
00822             goto exit_on_error;
00823         }
00824         input->relaxation
00825         = json_object_get_float_with_default (object,
00826         LABEL_RELAXATION,
00827         DEFAULT_RELAXATION,
00828         &error_code);
00829         if (error_code || input->relaxation < 0. || input->
00830         relaxation > 2.)
00831         {
00832             input_error (_("Invalid relaxation parameter"));
00833             goto exit_on_error;
00834         }
00835     }
00836     else
00837     {
00838         input->nsteps = 0;
00839     }
00840
00841     // Obtaining the threshold
00842     input->threshold
00843     = json_object_get_float_with_default (object,
00844     LABEL_THRESHOLD, 0.,
00845     &error_code);
00846     if (error_code)
00847     {
00848         input_error (_("Invalid threshold"));
00849         goto exit_on_error;
00850     }
00851
00852     // Reading the experimental data
00853     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00854     n = json_array_get_length (array);
00855     input->experiment = (Experiment *) g_malloc (n * sizeof (
00856     Experiment));

```

```

00848     for (i = 0; i < n; ++i)
00849     {
00850 #if DEBUG_INPUT
00851         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00852                 input->nexperiments);
00853 #endif
00854         child = json_array_get_element (array, i);
00855         if (!input->nexperiments)
00856         {
00857             if (!experiment_open_json (input->experiment, child, 0))
00858                 goto exit_on_error;
00859         }
00860         else
00861         {
00862             if (!experiment_open_json (input->experiment + input->
nexperiments,
00863                                     child, input->experiment->ninputs))
00864                 goto exit_on_error;
00865         }
00866         ++input->nexperiments;
00867 #if DEBUG_INPUT
00868         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00869                 input->nexperiments);
00870 #endif
00871     }
00872     if (!input->nexperiments)
00873     {
00874         input_error (_("No optimization experiments"));
00875         goto exit_on_error;
00876     }
00877
00878     // Reading the variables data
00879     array = json_object_get_array_member (object, LABEL_VARIABLES);
00880     n = json_array_get_length (array);
00881     input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00882     for (i = 0; i < n; ++i)
00883     {
00884 #if DEBUG_INPUT
00885         fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00886 #endif
00887         child = json_array_get_element (array, i);
00888         if (!variable_open_json (input->variable + input->
nvariables, child,
00889                                 input->algorithm, input->nsteps))
00890             goto exit_on_error;
00891         ++input->nvariables;
00892     }
00893     if (!input->nvariables)
00894     {
00895         input_error (_("No optimization variables"));
00896         goto exit_on_error;
00897     }
00898
00899     // Obtaining the error norm
00900     if (json_object_get_member (object, LABEL_NORM))
00901     {
00902         buffer = json_object_get_string_member (object, LABEL_NORM);
00903         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00904             input->norm = ERROR_NORM_EUCLIDIAN;
00905         else if (!strcmp (buffer, LABEL_MAXIMUM))
00906             input->norm = ERROR_NORM_MAXIMUM;
00907         else if (!strcmp (buffer, LABEL_P))
00908         {
00909             input->norm = ERROR_NORM_P;
00910             input->p = json_object_get_float (object,
LABEL_P, &error_code);
00911             if (!error_code)
00912             {
00913                 input_error (_("Bad P parameter"));
00914                 goto exit_on_error;
00915             }
00916         }
00917         else if (!strcmp (buffer, LABEL_TAXICAB))
00918             input->norm = ERROR_NORM_TAXICAB;
00919         else
00920         {
00921             input_error (_("Unknown error norm"));
00922             goto exit_on_error;
00923         }
00924     }
00925     else
00926         input->norm = ERROR_NORM_EUCLIDIAN;
00927
00928     // Closing the JSON document
00929     g_object_unref (parser);
00930
00931 #if DEBUG_INPUT

```

```

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

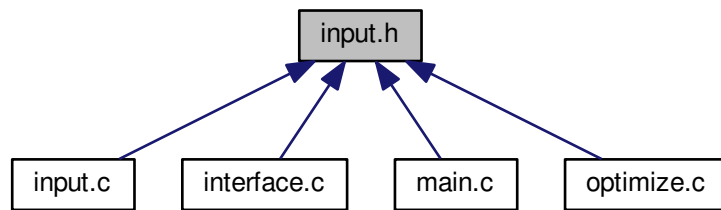
```

## 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 methods to estimate the direction search.*
- 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", _("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>	<a href="#">Input</a> data file name.
-----------------	---------------------------------------

##### Returns

1\_on\_success, 0\_on\_error.

Definition at line 952 of file [input.c](#).

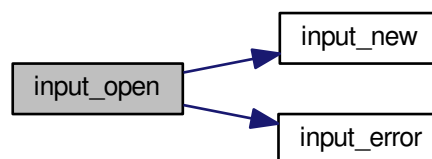
```
00953 {
00954     xmlDoc *doc;
00955     JsonParser *parser;
00956
00957     #if DEBUG_INPUT
```

```

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

```

Here is the call graph for this function:



### 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 562 of file [input.c](#).

```

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

```

```

00631     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00632     {
00633         input->algorithm = ALGORITHM_MONTE_CARLO;
00634
00635         // Obtaining simulations number
00636         input->nsimulations
00637             = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00638 );
00639         if (error_code)
00640         {
00641             input_error (_("Bad simulations number"));
00642             goto exit_on_error;
00643         }
00644     else if (!strcmp (buffer, LABEL_SWEEP))
00645         input->algorithm = ALGORITHM_SWEEP;
00646     else if (!strcmp (buffer, LABEL_GENETIC))
00647     {
00648         input->algorithm = ALGORITHM_GENETIC;
00649
00650         // Obtaining population
00651         if (json_object_get_member (object, LABEL_NPOPULATION))
00652         {
00653             input->nsimulations
00654                 = json_object_get_uint (object,
00655 LABEL_NPOPULATION, &error_code);
00656             if (error_code || input->nsimulations < 3)
00657             {
00658                 input_error (_("Invalid population number"));
00659                 goto exit_on_error;
00660             }
00661         else
00662         {
00663             input_error (_("No population number"));
00664             goto exit_on_error;
00665         }
00666
00667         // Obtaining generations
00668         if (json_object_get_member (object, LABEL_NGENERATIONS))
00669         {
00670             input->niterations
00671                 = json_object_get_uint (object,
00672 LABEL_NGENERATIONS, &error_code);
00673             if (error_code || !input->niterations)
00674             {
00675                 input_error (_("Invalid generations number"));
00676                 goto exit_on_error;
00677             }
00678         else
00679         {
00680             input_error (_("No generations number"));
00681             goto exit_on_error;
00682         }
00683
00684         // Obtaining mutation probability
00685         if (json_object_get_member (object, LABEL_MUTATION))
00686         {
00687             input->mutation_ratio
00688                 = json_object_get_float (object, LABEL_MUTATION, &error_code
00689 );
00690             if (error_code || input->mutation_ratio < 0.
00691 || input->mutation_ratio >= 1.)
00692             {
00693                 input_error (_("Invalid mutation probability"));
00694                 goto exit_on_error;
00695             }
00696         else
00697         {
00698             input_error (_("No mutation probability"));
00699             goto exit_on_error;
00700         }
00701
00702         // Obtaining reproduction probability
00703         if (json_object_get_member (object, LABEL_REPRODUCTION))
00704         {
00705             input->reproduction_ratio
00706                 = json_object_get_float (object,
00707 LABEL_REPRODUCTION, &error_code);
00708             if (error_code || input->reproduction_ratio < 0.
00709 || input->reproduction_ratio >= 1.0)
00710             {
00711                 input_error (_("Invalid reproduction probability"));
00712                 goto exit_on_error;

```

```

00713     }
00714     else
00715     {
00716         input_error (_("No reproduction probability"));
00717         goto exit_on_error;
00718     }
00719
00720     // Obtaining adaptation probability
00721     if (json_object_get_member (object, LABEL_ADAPTATION))
00722     {
00723         input->adaptation_ratio
00724         = json_object_get_float (object,
00725 LABEL_ADAPTATION, &error_code);
00726         if (error_code || input->adaptation_ratio < 0.
00727             || input->adaptation_ratio >= 1.)
00728         {
00729             input_error (_("Invalid adaptation probability"));
00730             goto exit_on_error;
00731         }
00732     }
00733     else
00734     {
00735         input_error (_("No adaptation probability"));
00736         goto exit_on_error;
00737     }
00738     // Checking survivals
00739     i = input->mutation_ratio * input->nsimulations;
00740     i += input->reproduction_ratio * input->
00741 nsimulations;
00742     i += input->adaptation_ratio * input->
00743 nsimulations;
00744     if (i > input->nsimulations - 2)
00745     {
00746         input_error
00747         (_("No enough survival entities to reproduce the population"));
00748         goto exit_on_error;
00749     }
00750     else
00751     {
00752         input_error (_("Unknown algorithm"));
00753         goto exit_on_error;
00754     }
00755     if (input->algorithm == ALGORITHM_MONTE_CARLO
00756         || input->algorithm == ALGORITHM_SWEEP)
00757     {
00758
00759         // Obtaining iterations number
00760         input->niterations
00761         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00762 );
00763         if (error_code == 1)
00764             input->niterations = 1;
00765         else if (error_code)
00766         {
00767             input_error (_("Bad iterations number"));
00768             goto exit_on_error;
00769         }
00770         // Obtaining best number
00771         input->nbest
00772         = json_object_get_uint_with_default (object,
00773 LABEL_NBEST, 1,
00774                                             &error_code);
00775         if (error_code || !input->nbest)
00776         {
00777             input_error (_("Invalid best number"));
00778             goto exit_on_error;
00779         }
00780         // Obtaining tolerance
00781         input->tolerance
00782         = json_object_get_float_with_default (object,
00783 LABEL_TOLERANCE, 0.,
00784                                             &error_code);
00785         if (error_code || input->tolerance < 0.)
00786         {
00787             input_error (_("Invalid tolerance"));
00788             goto exit_on_error;
00789         }
00790         // Getting direction search method parameters
00791         if (json_object_get_member (object, LABEL_NSTEPS))
00792         {
00793             input->nsteps

```

```

00794         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00795     if (error_code)
00796     {
00797         input_error (_("Invalid steps number"));
00798         goto exit_on_error;
00799     }
00800     buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00801     if (!strcmp (buffer, LABEL_COORDINATES))
00802         input->direction = DIRECTION_METHOD_COORDINATES;
00803     else if (!strcmp (buffer, LABEL_RANDOM))
00804     {
00805         input->direction = DIRECTION_METHOD_RANDOM;
00806         input->nestimates
00807             =
00808             json_object_get_uint (object,
00809 LABEL_NESTIMATES, &error_code);
00809         if (error_code || !input->nestimates)
00810         {
00811             input_error (_("Invalid estimates number"));
00812             goto exit_on_error;
00813         }
00814     }
00815     else
00816     {
00817         input_error
00818             (_("Unknown method to estimate the direction search"));
00819         goto exit_on_error;
00820     }
00821     input->relaxation
00822         = json_object_get_float_with_default (object,
00823 LABEL_RELAXATION,
00824                                             DEFAULT_RELAXATION,
00825                                             &error_code);
00825     if (error_code || input->relaxation < 0. || input->
00826 relaxation > 2.)
00827     {
00827         input_error (_("Invalid relaxation parameter"));
00828         goto exit_on_error;
00829     }
00830 }
00831 else
00832     input->nsteps = 0;
00833 }
00834 // Obtaining the threshold
00835 input->threshold
00836     = json_object_get_float_with_default (object,
00837 LABEL_THRESHOLD, 0.,
00838                                             &error_code);
00838 if (error_code)
00839 {
00840     input_error (_("Invalid threshold"));
00841     goto exit_on_error;
00842 }
00843
00844 // Reading the experimental data
00845 array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00846 n = json_array_get_length (array);
00847 input->experiment = (Experiment *) g_malloc (n * sizeof (
00848 Experiment));
00848 for (i = 0; i < n; ++i)
00849 {
00850     #if DEBUG_INPUT
00851         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00852                 input->nexperiments);
00853     #endif
00854     child = json_array_get_element (array, i);
00855     if (!input->nexperiments)
00856     {
00857         if (!experiment_open_json (input->experiment, child, 0))
00858             goto exit_on_error;
00859     }
00860     else
00861     {
00862         if (!experiment_open_json (input->experiment +
00863 input->nexperiments,
00864                                     child, input->experiment->
00865 ninputs))
00866             goto exit_on_error;
00867     }
00868     ++input->nexperiments;
00869     #if DEBUG_INPUT
00870         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00871                 input->nexperiments);
00872     #endif
00873 }

```

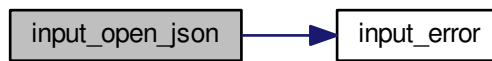


```

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

```

Here is the call graph for this function:



#### 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 (_("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173     }

```

```

00174     if (!input->variables)
00175     {
00176         input->variables =
00177             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178         if (!input->variables)
00179             input->variables =
00180                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (_("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed
00198         = xml_node_get_uint_with_default (node, (const xmlChar *)
00199     LABEL_SEED,
00200                                         DEFAULT_RANDOM_SEED, &error_code);
00201     if (error_code)
00202     {
00203         input_error (_("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 (_("Bad simulations number"));
00221             goto exit_on_error;
00222         }
00223     }
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00225         input->algorithm = ALGORITHM_SWEEP;
00226     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00227     {
00228         input->algorithm = ALGORITHM_GENETIC;
00229
00230         // Obtaining population
00231         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00232         {
00233             input->nsimulations
00234                 = xml_node_get_uint (node, (const xmlChar *)
00235     LABEL_NPOPULATION,
00236                                     &error_code);
00237             if (error_code || input->nsimulations < 3)
00238             {
00239                 input_error (_("Invalid population number"));
00240                 goto exit_on_error;
00241             }
00242         }
00243     }
00244     else
00245     {
00246         input_error (_("No population number"));
00247         goto exit_on_error;
00248     }
00249
00250     // Obtaining generations
00251     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00252     {
00253         input->niterations
00254             = xml_node_get_uint (node, (const xmlChar *)
00255     LABEL_NGENERATIONS,
00256                               &error_code);
00257         if (error_code || !input->niterations)
00258         {
00259             input_error (_("Invalid generations number"));
00260             goto exit_on_error;
00261         }
00262     }

```

```

00257     }
00258     else
00259     {
00260         input_error (_("No generations number"));
00261         goto exit_on_error;
00262     }
00263
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268         = xml_node_get_float (node, (const xmlChar *)
00269         LABEL_MUTATION,
00270         &error_code);
00271         if (error_code || input->mutation_ratio < 0.
00272             || input->mutation_ratio >= 1.)
00273         {
00274             input_error (_("Invalid mutation probability"));
00275             goto exit_on_error;
00276         }
00277     }
00278     else
00279     {
00280         input_error (_("No mutation probability"));
00281         goto exit_on_error;
00282     }
00283
00284     // Obtaining reproduction probability
00285     if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00286     {
00287         input->reproduction_ratio
00288         = xml_node_get_float (node, (const xmlChar *)
00289         LABEL_REPRODUCTION,
00290         &error_code);
00291         if (error_code || input->reproduction_ratio < 0.
00292             || input->reproduction_ratio >= 1.0)
00293         {
00294             input_error (_("Invalid reproduction probability"));
00295             goto exit_on_error;
00296         }
00297     }
00298     else
00299     {
00300         input_error (_("No reproduction probability"));
00301         goto exit_on_error;
00302     }
00303
00304     // Obtaining adaptation probability
00305     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00306     {
00307         input->adaptation_ratio
00308         = xml_node_get_float (node, (const xmlChar *)
00309         LABEL_ADAPTATION,
00310         &error_code);
00311         if (error_code || input->adaptation_ratio < 0.
00312             || input->adaptation_ratio >= 1.)
00313         {
00314             input_error (_("Invalid adaptation probability"));
00315             goto exit_on_error;
00316         }
00317     }
00318     else
00319     {
00320         input_error (_("No adaptation probability"));
00321         goto exit_on_error;
00322     }
00323
00324     // Checking survivals
00325     i = input->mutation_ratio * input->nsimulations;
00326     i += input->reproduction_ratio * input->
00327     nsimulations;
00328     i += input->adaptation_ratio * input->
00329     nsimulations;
00330     if (i > input->nsimulations - 2)
00331     {
00332         input_error
00333         (_("No enough survival entities to reproduce the population"));
00334         goto exit_on_error;
00335     }
00336     }
00337     else
00338     {
00339         input_error (_("Unknown algorithm"));
00340         goto exit_on_error;
00341     }
00342     }
00343     xmlFree (buffer);
00344     buffer = NULL;

```

```

00339
00340     if (input->algorithm == ALGORITHM_MONTE_CARLO
00341         || input->algorithm == ALGORITHM_SWEEP)
00342     {
00343
00344         // Obtaining iterations number
00345         input->niterations
00346         = xml_node_get_uint (node, (const xmlChar *)
00347 LABEL_NITERATIONS,
                                &error_code);
00348         if (error_code == 1)
00349             input->niterations = 1;
00350         else if (error_code)
00351         {
00352             input_error (_("Bad iterations number"));
00353             goto exit_on_error;
00354         }
00355
00356         // Obtaining best number
00357         input->nbest
00358         = xml_node_get_uint_with_default (node, (const xmlChar *)
00359 LABEL_NBEST,
                                1, &error_code);
00360         if (error_code || !input->nbest)
00361         {
00362             input_error (_("Invalid best number"));
00363             goto exit_on_error;
00364         }
00365         if (input->nbest > input->nsimulations)
00366         {
00367             input_error (_("Best number higher than simulations number"));
00368             goto exit_on_error;
00369         }
00370
00371         // Obtaining tolerance
00372         input->tolerance
00373         = xml_node_get_float_with_default (node,
00374                                           (const xmlChar *) LABEL_TOLERANCE,
00375                                           0., &error_code);
00376         if (error_code || input->tolerance < 0.)
00377         {
00378             input_error (_("Invalid tolerance"));
00379             goto exit_on_error;
00380         }
00381
00382         // Getting direction search method parameters
00383         if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00384         {
00385             input->nsteps =
00386             xml_node_get_uint (node, (const xmlChar *)
00387 LABEL_NSTEPS,
                                &error_code);
00388             if (error_code)
00389             {
00390                 input_error (_("Invalid steps number"));
00391                 goto exit_on_error;
00392             }
00393             buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00394             if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00395                 input->direction = DIRECTION_METHOD_COORDINATES;
00396             else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00397             {
00398                 input->direction = DIRECTION_METHOD_RANDOM;
00399                 input->nestimates
00400                 = xml_node_get_uint (node, (const xmlChar *)
00401 LABEL_NESTIMATES,
                                &error_code);
00402                 if (error_code || !input->nestimates)
00403                 {
00404                     input_error (_("Invalid estimates number"));
00405                     goto exit_on_error;
00406                 }
00407             }
00408             else
00409             {
00410                 input_error
00411                 (_("Unknown method to estimate the direction search"));
00412                 goto exit_on_error;
00413             }
00414             xmlFree (buffer);
00415             buffer = NULL;
00416             input->relaxation
00417             = xml_node_get_float_with_default (node,
00418                                               (const xmlChar *)
00419 LABEL_RELAXATION,
                                               DEFAULT_RELAXATION,
00420                                               &error_code);
00421

```

```

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

```

```

00503     buffer = NULL;
00504
00505     // Obtaining the error norm
00506     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00507     {
00508         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00509         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00510             input->norm = ERROR_NORM_EUCLIDIAN;
00511         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00512             input->norm = ERROR_NORM_MAXIMUM;
00513         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00514         {
00515             input->norm = ERROR_NORM_P;
00516             input->p
00517             =
00518             xml_node_get_float (node, (const xmlChar *)
LABEL_P, &error_code);
00519             if (!error_code)
00520             {
00521                 input_error (_("Bad P parameter"));
00522                 goto exit_on_error;
00523             }
00524         }
00525         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00526             input->norm = ERROR_NORM_TAXICAB;
00527         else
00528         {
00529             input_error (_("Unknown error norm"));
00530             goto exit_on_error;
00531         }
00532         xmlFree (buffer);
00533     }
00534     else
00535         input->norm = ERROR_NORM_EUCLIDIAN;
00536
00537     // Closing the XML document
00538     xmlFreeDoc (doc);
00539
00540     #if DEBUG_INPUT
00541     fprintf (stderr, "input_open_xml: end\n");
00542     #endif
00543     return 1;
00544
00545 exit_on_error:
00546     xmlFree (buffer);
00547     xmlFreeDoc (doc);
00548     #if DEBUG_INPUT
00549     fprintf (stderr, "input_open_xml: end\n");
00550     #endif
00551     return 0;
00552 }

```

Here is the call graph for this function:



## 5.10 input.h

```

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

```

```

00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INPUT__H
00033 #define INPUT__H 1
00034
00035 enum DirectionMethod
00036 {
00037     DIRECTION_METHOD_COORDINATES = 0,
00038     DIRECTION_METHOD_RANDOM = 1,
00039 };
00040
00041 enum ErrorNorm
00042 {
00043     ERROR_NORM_EUCLIDIAN = 0,
00044     ERROR_NORM_MAXIMUM = 1,
00045     ERROR_NORM_P = 2,
00046     ERROR_NORM_TAXICAB = 3
00047 };
00048
00049 typedef struct
00050 {
00051     Experiment *experiment;
00052     Variable *variable;
00053     char *result;
00054     char *variables;
00055     char *simulator;
00056     char *evaluator;
00057     char *directory;
00058     char *name;
00059     double tolerance;
00060     double mutation_ratio;
00061     double reproduction_ratio;
00062     double adaptation_ratio;
00063     double relaxation;
00064     double p;
00065     double threshold;
00066     unsigned long int seed;
00067     unsigned int nvariables;
00068     unsigned int nexperiments;
00069     unsigned int nsimulations;
00070     unsigned int algorithm;
00071     unsigned int nsteps;
00072     unsigned int direction;
00073     unsigned int nestimates;
00074     unsigned int niterations;
00075     unsigned int nbest;
00076     unsigned int norm;
00077     unsigned int type;
00078 } Input;
00079
00080 extern Input input[1];
00081 extern const char *result_name;
00082 extern const char *variables_name;
00083
00084 // Public functions
00085 void input_new ();
00086 void input_free ();
00087 void input_error (char *message);
00088 int input_open_xml (xmlDoc * doc);
00089 int input_open_json (JsonParser * parser);
00090 int input_open (char *filename);
00091
00092 #endif

```

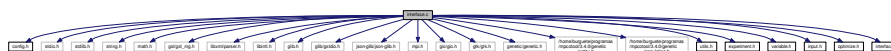


## 5.11 interface.c File Reference

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](#) (GtkApplication \*application)  
*Function to open the main window.*

## Variables

- const char \* [logo](#) []  
*Logo pixmap.*
- [Options](#) [options](#) [1]  
*Options struct to define the options dialog.*
- [Running](#) [running](#) [1]  
*Running struct to define the running dialog.*
- [Window](#) [window](#) [1]  
*Window struct to define the main interface window.*

## 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.
-----------------	------------------

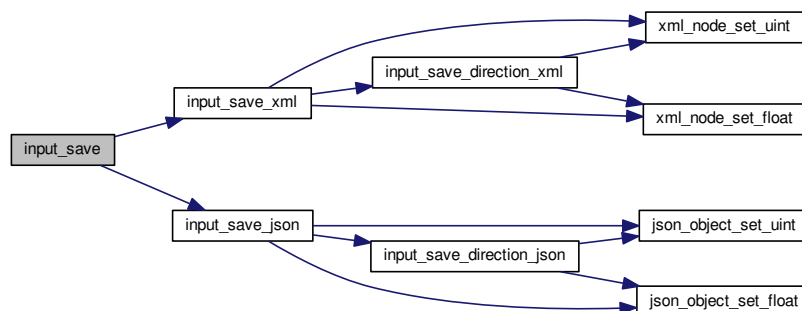
Definition at line 579 of file [interface.c](#).

```

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

```

Here is the call graph for this function:



## 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.

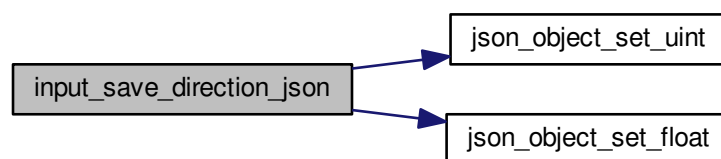
## Parameters

<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
00237 }
```

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

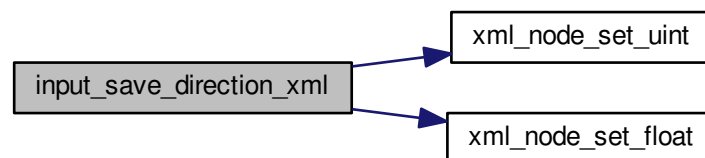
Definition at line 171 of file [interface.c](#).

```

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 416 of file [interface.c](#).

```

00417 {
00418     unsigned int i, j;
00419     char *buffer;
00420     JsonNode *node, *child;
00421     JsonObject *object, *object2;
00422     JsonArray *array;
00423     GFile *file, *file2;
00424
00425     #if DEBUG_INTERFACE
00426     fprintf (stderr, "input_save_json: start\n");
00427     #endif
00428
00429     // Setting root JSON node
00430     node = json_node_new (JSON_NODE_OBJECT);
00431     object = json_node_get_object (node);
00432     json_generator_set_root (generator, node);
00433
00434     // Adding properties to the root JSON node
00435     if (strcmp (input->result, result_name))
00436         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00437     if (strcmp (input->variables, variables_name))
00438         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00439
00440     file = g_file_new_for_path (input->directory);
00441     file2 = g_file_new_for_path (input->simulator);
00442     buffer = g_file_get_relative_path (file, file2);
00443     g_object_unref (file2);
00444     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00445     g_free (buffer);
00446     if (input->evaluator)
00447     {
00448         file2 = g_file_new_for_path (input->evaluator);
00449         buffer = g_file_get_relative_path (file, file2);
00450         g_object_unref (file2);
00451         if (strlen (buffer))
00452             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00453         g_free (buffer);
00454     }
00455     if (input->seed != DEFAULT_RANDOM_SEED)
00456         json_object_set_uint (object, LABEL_SEED,
input->seed);
00457
00458     // Setting the algorithm
00459     buffer = (char *) g_slice_alloc (64);
00460     switch (input->algorithm)
00461     {
00462     case ALGORITHM_MONTE_CARLO:
00463         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00464         snprintf (buffer, 64, "%u", input->nsimulations);
00465         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00466         snprintf (buffer, 64, "%u", input->niterations);
00467         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00468         snprintf (buffer, 64, "%.3lg", input->tolerance);
00469         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00470         snprintf (buffer, 64, "%u", input->nbest);
00471         json_object_set_string_member (object, LABEL_NBEST, buffer);
00472         input_save_direction_json (node);
00473         break;
00474     case ALGORITHM_SWEEP:
00475         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00476         snprintf (buffer, 64, "%u", input->niterations);
00477         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00478         snprintf (buffer, 64, "%.3lg", input->tolerance);
00479         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00480         snprintf (buffer, 64, "%u", input->nbest);
00481         json_object_set_string_member (object, LABEL_NBEST, buffer);
00482         input_save_direction_json (node);
00483         break;
00484     default:
00485         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00486         snprintf (buffer, 64, "%u", input->nsimulations);
00487         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00488         snprintf (buffer, 64, "%u", input->niterations);
00489         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00490         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00491         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00492         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00493         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00494         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00495         json_object_set_string_member (object, LABEL_ADAPTATION, buffer);

```

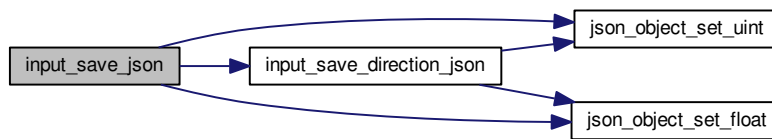
```

00497         break;
00498     }
00499     g_slice_free1 (64, buffer);
00500     if (input->threshold != 0.)
00501         json_object_set_float (object, LABEL_THRESHOLD,
00502                                input->threshold);
00503     // Setting the experimental data
00504     array = json_array_new ();
00505     for (i = 0; i < input->nexperiments; ++i)
00506     {
00507         child = json_node_new (JSON_NODE_OBJECT);
00508         object = json_node_get_object (child);
00509         json_object_set_string_member (object2, LABEL_NAME,
00510                                       input->experiment[i].name);
00511         if (input->experiment[i].weight != 1.)
00512             json_object_set_float (object2, LABEL_WEIGHT,
00513                                   input->experiment[i].weight);
00514         for (j = 0; j < input->experiment->ninputs; ++j)
00515             json_object_set_string_member (object2, template[j],
00516                                           input->experiment[i].
00517                                           template[j]);
00518         json_array_add_element (array, child);
00519     }
00520     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00521     // Setting the variables data
00522     array = json_array_new ();
00523     for (i = 0; i < input->nvariables; ++i)
00524     {
00525         child = json_node_new (JSON_NODE_OBJECT);
00526         object = json_node_get_object (child);
00527         json_object_set_string_member (object2, LABEL_NAME,
00528                                       input->variable[i].name);
00529         json_object_set_float (object2, LABEL_MINIMUM,
00530                               input->variable[i].rangemin);
00531         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532             json_object_set_float (object2,
00533                                   LABEL_ABSOLUTE_MINIMUM,
00534                                   input->variable[i].rangeminabs);
00535         json_object_set_float (object2, LABEL_MAXIMUM,
00536                               input->variable[i].rangemax);
00537         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00538             json_object_set_float (object2,
00539                                   LABEL_ABSOLUTE_MAXIMUM,
00540                                   input->variable[i].rangemaxabs);
00541         if (input->variable[i].precision !=
00542             DEFAULT_PRECISION)
00543             json_object_set_uint (object2, LABEL_PRECISION,
00544                                  input->variable[i].precision);
00545         if (input->algorithm == ALGORITHM_SWEEP)
00546             json_object_set_uint (object2, LABEL_NSWEEPS,
00547                                  input->variable[i].nsweeps);
00548         else if (input->algorithm == ALGORITHM_GENETIC)
00549             json_object_set_uint (object2, LABEL_NBITS,
00550                                  input->variable[i].nbits);
00551         if (input->nsteps)
00552             json_object_set_float (object, LABEL_STEP,
00553                                   input->variable[i].step);
00554         json_array_add_element (array, child);
00555     }
00556     json_object_set_array_member (object, LABEL_VARIABLES, array);
00557     // Saving the error norm
00558     switch (input->norm)
00559     {
00560     case ERROR_NORM_MAXIMUM:
00561         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00562         break;
00563     case ERROR_NORM_P:
00564         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00565         json_object_set_float (object, LABEL_P, input->
00566                                p);
00567         break;
00568     case ERROR_NORM_TAXICAB:
00569         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00570     }
00571     #if DEBUG_INTERFACE
00572     fprintf (stderr, "input_save_json: end\n");
00573     #endif
00574 }

```



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 243 of file [interface.c](#).

```

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

```

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

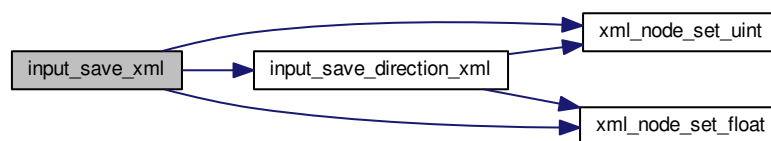
```

```

    LABEL_ABSOLUTE_MINIMUM,
00367         input->variable[i].rangeminabs);
00368     xml_node_set_float (child, (const xmlChar *)
    LABEL_MAXIMUM,
00369         input->variable[i].rangemax);
00370     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00371         xml_node_set_float (child, (const xmlChar *)
    LABEL_ABSOLUTE_MAXIMUM,
00372         input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
    DEFAULT_PRECISION)
00374         xml_node_set_uint (child, (const xmlChar *)
    LABEL_PRECISION,
00375         input->variable[i].precision);
00376     if (input->algorithm == ALGORITHM_SWEEP)
00377         xml_node_set_uint (child, (const xmlChar *)
    LABEL_NSWEEPS,
00378         input->variable[i].nsweeps);
00379     else if (input->algorithm == ALGORITHM_GENETIC)
00380         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381         input->variable[i].nbits);
00382     if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
    LABEL_STEP,
00384         input->variable[i].step);
00385 }
00386
00387 // Saving the error norm
00388 switch (input->norm)
00389 {
00390     case ERROR_NORM_MAXIMUM:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392             (const xmlChar *) LABEL_MAXIMUM);
00393         break;
00394     case ERROR_NORM_P:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396             (const xmlChar *) LABEL_P);
00397         xml_node_set_float (node, (const xmlChar *) LABEL_P,
    input->p);
00398         break;
00399     case ERROR_NORM_TAXICAB:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401             (const xmlChar *) LABEL_TAXICAB);
00402 }
00403
00404 #if DEBUG_INTERFACE
00405 fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }

```

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 732 of file [interface.c](#).

```

00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736     fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
00739                             NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);
00741     fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }

```

Here is the call graph for this function:

**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 752 of file [interface.c](#).

```

00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_direction: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_direction,
00759                             NDIRECTIONS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_direction: %u\n", i);
00761     fprintf (stderr, "window_get_direction: end\n");
00762     #endif
00763     return i;
00764 }

```

Here is the call graph for this function:



#### 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 772 of file [interface.c](#).

```
00773 {  
00774     unsigned int i;  
00775     #if DEBUG_INTERFACE  
00776     fprintf (stderr, "window_get_norm: start\n");  
00777     #endif  
00778     i = gtk_array_get_active (window->button_norm,  
00779                             NNORMS);  
00779     #if DEBUG_INTERFACE  
00780     fprintf (stderr, "window_get_norm: %u\n", i);  
00781     fprintf (stderr, "window_get_norm: end\n");  
00782     #endif  
00783     return i;  
00784 }
```

Here is the call graph for this function:



#### 5.11.2.9 window\_new()

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

Function to open the main window.

## Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2108 of file [interface.c](#).

```

02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[N DIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[N DIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135 #if DEBUG_INTERFACE
02136     fprintf (stderr, "window_new: start\n");
02137 #endif
02138
02139     // Creating the window
02140     window->window = main_window
02141         = (GtkWindow *) gtk_application_window_new (application);
02142
02143     // Finish when closing the window
02144     g_signal_connect_swapped (window->window, "delete-event",
02145                             G_CALLBACK (g_application_quit),
02146                             G_APPLICATION (application));
02147
02148     // Setting the window title
02149     gtk_window_set_title (window->window, "MPCOTool");
02150
02151     // Creating the open button
02152     window->button_open = (GtkToolButton *) gtk_tool_button_new
02153         (gtk_image_new_from_icon_name ("document-open",
02154                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157     // Creating the save button
02158     window->button_save = (GtkToolButton *) gtk_tool_button_new
02159         (gtk_image_new_from_icon_name ("document-save",
02160                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161     g_signal_connect (window->button_save, "clicked", (void (*)(
02162 window_save,
02163                             NULL);
02164
02165     // Creating the run button
02166     window->button_run = (GtkToolButton *) gtk_tool_button_new
02167         (gtk_image_new_from_icon_name ("system-run",
02168                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171     // Creating the options button
02172     window->button_options = (GtkToolButton *) gtk_tool_button_new
02173         (gtk_image_new_from_icon_name ("preferences-system",
02174                                     GTK_ICON_SIZE_LARGE_TOOLBAR),
02175         _("Options"));
02176     g_signal_connect (window->button_options, "clicked",
02177 options_new, NULL);
02178
02179     // Creating the help button
02180     window->button_help = (GtkToolButton *) gtk_tool_button_new
02181         (gtk_image_new_from_icon_name ("help-browser",
02182                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02183     g_signal_connect (window->button_help, "clicked", window_help, NULL);
02184

```

```

02183 // Creating the about button
02184 window->button_about = (GtkToolButton *) gtk_tool_button_new
02185     (gtk_image_new_from_icon_name ("help-about",
02186                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02187 g_signal_connect (window->button_about, "clicked",
02188     window_about, NULL);
02189 // Creating the exit button
02190 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02191     (gtk_image_new_from_icon_name ("application-exit",
02192                                     GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02193 g_signal_connect_swapped (window->button_exit, "clicked",
02194     G_CALLBACK (g_application_quit),
02195     G_APPLICATION (application));
02196 // Creating the buttons bar
02197 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02198 gtk_toolbar_insert
02199     (window->bar_buttons, GTK_TOOL_ITEM (window->
02200     button_open), 0);
02201 gtk_toolbar_insert
02202     (window->bar_buttons, GTK_TOOL_ITEM (window->
02203     button_save), 1);
02204 gtk_toolbar_insert
02205     (window->bar_buttons, GTK_TOOL_ITEM (window->
02206     button_run), 2);
02207 gtk_toolbar_insert
02208     (window->bar_buttons, GTK_TOOL_ITEM (window->
02209     button_options), 3);
02210 gtk_toolbar_insert
02211     (window->bar_buttons, GTK_TOOL_ITEM (window->
02212     button_help), 4);
02213 gtk_toolbar_insert
02214     (window->bar_buttons, GTK_TOOL_ITEM (window->
02215     button_about), 5);
02216 gtk_toolbar_insert
02217     (window->bar_buttons, GTK_TOOL_ITEM (window->
02218     button_exit), 6);
02219 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02220 // Creating the simulator program label and entry
02221 window->label_simulator
02222     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02223 window->button_simulator = (GtkFileChooserButton *)
02224     gtk_file_chooser_button_new (_("Simulator program"),
02225     GTK_FILE_CHOOSER_ACTION_OPEN);
02226 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02227     _("Simulator program executable file"));
02228 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02229 // Creating the evaluator program label and entry
02230 window->check_evaluator = (GtkCheckButton *)
02231     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02232 g_signal_connect (window->check_evaluator, "toggled",
02233     window_update, NULL);
02234 window->button_evaluator = (GtkFileChooserButton *)
02235     gtk_file_chooser_button_new (_("Evaluator program"),
02236     GTK_FILE_CHOOSER_ACTION_OPEN);
02237 gtk_widget_set_tooltip_text
02238     (GTK_WIDGET (window->button_evaluator),
02239     _("Optional evaluator program executable file"));
02240 // Creating the results files labels and entries
02241 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02242 window->entry_result = (GtkEntry *) gtk_entry_new ();
02243 gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->entry_result), _("Best results file"));
02245 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02246 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02247 gtk_widget_set_tooltip_text
02248     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02249 // Creating the files grid and attaching widgets
02250 window->grid_files = (GtkGrid *) gtk_grid_new ();
02251 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02252     label_simulator),
02253     0, 0, 1, 1);
02254 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02255     button_simulator),
02256     1, 0, 1, 1);
02257 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02258     check_evaluator),
02259     0, 1, 1, 1);
02260 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02261     button_evaluator),
02262     1, 1, 1, 1);
02263 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->

```

```

    label_result),
02257         0, 2, 1, 1);
02258     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02259         1, 2, 1, 1);
02260     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02261         0, 3, 1, 1);
02262     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02263         1, 3, 1, 1);
02264
02265     // Creating the algorithm properties
02266     window->label_simulations = (GtkLabel *) gtk_label_new
02267         (_("Simulations number"));
02268     window->spin_simulations
02269         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270     gtk_widget_set_tooltip_text
02271         (GTK_WIDGET (window->spin_simulations),
02272          _("Number of simulations to perform for each iteration"));
02273     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274     window->label_iterations = (GtkLabel *)
02275         gtk_label_new (_("Iterations number"));
02276     window->spin_iterations
02277         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278     gtk_widget_set_tooltip_text
02279         (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280     g_signal_connect
02281         (window->spin_iterations, "value-changed",
window_update, NULL);
02282     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283     window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284     window->spin_tolerance =
02285         (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286     gtk_widget_set_tooltip_text
02287         (GTK_WIDGET (window->spin_tolerance),
02288          _("Tolerance to set the variable interval on the next iteration"));
02289     window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290     window->spin_bests
02291         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292     gtk_widget_set_tooltip_text
02293         (GTK_WIDGET (window->spin_bests),
02294          _("Number of best simulations used to set the variable interval "
02295            "on the next iteration"));
02296     window->label_population
02297         = (GtkLabel *) gtk_label_new (_("Population number"));
02298     window->spin_population
02299         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300     gtk_widget_set_tooltip_text
02301         (GTK_WIDGET (window->spin_population),
02302          _("Number of population for the genetic algorithm"));
02303     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_population), TRUE);
02304     window->label_generations
02305         = (GtkLabel *) gtk_label_new (_("Generations number"));
02306     window->spin_generations
02307         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308     gtk_widget_set_tooltip_text
02309         (GTK_WIDGET (window->spin_generations),
02310          _("Number of generations for the genetic algorithm"));
02311     window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312     window->spin_mutation
02313         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314     gtk_widget_set_tooltip_text
02315         (GTK_WIDGET (window->spin_mutation),
02316          _("Ratio of mutation for the genetic algorithm"));
02317     window->label_reproduction
02318         = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319     window->spin_reproduction
02320         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321     gtk_widget_set_tooltip_text
02322         (GTK_WIDGET (window->spin_reproduction),
02323          _("Ratio of reproduction for the genetic algorithm"));
02324     window->label_adaptation
02325         = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326     window->spin_adaptation
02327         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328     gtk_widget_set_tooltip_text
02329         (GTK_WIDGET (window->spin_adaptation),
02330          _("Ratio of adaptation for the genetic algorithm"));
02331     window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332     window->spin_threshold = (GtkSpinButton *)
02333         gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                         precision[DEFAULT_PRECISION]);
02335     gtk_widget_set_tooltip_text
02336         (GTK_WIDGET (window->spin_threshold),
02337          _("Threshold in the objective function to finish the simulations"));
02338     window->scrolled_threshold =

```



```

02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341         GTK_WIDGET (window->spin_threshold));
02342     // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344         // GTK_ALIGN_FILL);
02345
02346     // Creating the direction search method properties
02347     window->check_direction = (GtkCheckButton *)
02348         gtk_check_button_new_with_mnemonic (_("_Direction search method"));
02349     g_signal_connect (window->check_direction, "clicked",
02350         window_update, NULL);
02351     window->grid_direction = (GtkGrid *) gtk_grid_new ();
02352     window->button_direction[0] = (GtkRadioButton *)
02353         gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02354     gtk_grid_attach (window->grid_direction,
02355         GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02356     g_signal_connect (window->button_direction[0], "clicked",
02357         window_update,
02358         NULL);
02359     for (i = 0; ++i < NDIRECTIONS;)
02360     {
02361         window->button_direction[i] = (GtkRadioButton *)
02362             gtk_radio_button_new_with_mnemonic
02363             (gtk_radio_button_get_group (window->button_direction[0]),
02364                 label_direction[i]);
02365         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02366             tip_direction[i]);
02367         gtk_grid_attach (window->grid_direction,
02368             GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02369         g_signal_connect (window->button_direction[i], "clicked",
02370             window_update, NULL);
02371     }
02372     window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02373     window->spin_steps = (GtkSpinButton *)
02374         gtk_spin_button_new_with_range (1., 1.e12, 1.);
02375     gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02376     window->label_estimates
02377         = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02378     window->spin_estimates = (GtkSpinButton *)
02379         gtk_spin_button_new_with_range (1., 1.e3, 1.);
02380     window->label_relaxation
02381         = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02382     window->spin_relaxation = (GtkSpinButton *)
02383         gtk_spin_button_new_with_range (0., 2., 0.001);
02384     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02385         window->label_steps),
02386         0, NDIRECTIONS, 1, 1);
02387     gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02388         window->spin_steps),
02389         1, NDIRECTIONS, 1, 1);
02390     gtk_grid_attach (window->grid_direction,
02391         GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02392         1, 1);
02393     gtk_grid_attach (window->grid_direction,
02394         GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02395         1);
02396     gtk_grid_attach (window->grid_direction,
02397         GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02398         1, 1);
02399     gtk_grid_attach (window->grid_direction,
02400         GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02401         1, 1);
02402
02403     // Creating the array of algorithms
02404     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02405     window->button_algorithm[0] = (GtkRadioButton *)
02406         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02407     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02408         tip_algorithm[0]);
02409     gtk_grid_attach (window->grid_algorithm,
02410         GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02411     g_signal_connect (window->button_algorithm[0], "clicked",
02412         window_set_algorithm, NULL);
02413     for (i = 0; ++i < NALGORITHMS;)
02414     {
02415         window->button_algorithm[i] = (GtkRadioButton *)
02416             gtk_radio_button_new_with_mnemonic
02417             (gtk_radio_button_get_group (window->button_algorithm[0]),
02418                 label_algorithm[i]);
02419         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02420             tip_algorithm[i]);
02421         gtk_grid_attach (window->grid_algorithm,
02422             GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02423         g_signal_connect (window->button_algorithm[i], "clicked",
02424             window_set_algorithm, NULL);
02425     }

```

```

02422 gtk_grid_attach (window->grid_algorithm,
02423                  GTK_WIDGET (window->label_simulations), 0,
02424                  NALGORITHMS, 1, 1);
02425 gtk_grid_attach (window->grid_algorithm,
02426                  GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427                  1);
02428 gtk_grid_attach (window->grid_algorithm,
02429                  GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430                  1, 1);
02431 gtk_grid_attach (window->grid_algorithm,
02432                  GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433                  1, 1);
02434 gtk_grid_attach (window->grid_algorithm,
02435                  GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436                  1, 1);
02437 gtk_grid_attach (window->grid_algorithm,
02438                  GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439                  1);
02440 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02441 window->label_bests),
02442                  0, NALGORITHMS + 3, 1, 1);
02442 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02443 window->spin_bests), 1,
02444                  NALGORITHMS + 3, 1, 1);
02444 gtk_grid_attach (window->grid_algorithm,
02445                  GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02446                  1, 1);
02447 gtk_grid_attach (window->grid_algorithm,
02448                  GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02449                  1, 1);
02450 gtk_grid_attach (window->grid_algorithm,
02451                  GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02452                  1, 1);
02453 gtk_grid_attach (window->grid_algorithm,
02454                  GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02455                  1, 1);
02456 gtk_grid_attach (window->grid_algorithm,
02457                  GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02458                  1);
02459 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02460 window->spin_mutation),
02461                  1, NALGORITHMS + 6, 1, 1);
02461 gtk_grid_attach (window->grid_algorithm,
02462                  GTK_WIDGET (window->label_reproduction), 0,
02463                  NALGORITHMS + 7, 1, 1);
02464 gtk_grid_attach (window->grid_algorithm,
02465                  GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02466                  1, 1);
02467 gtk_grid_attach (window->grid_algorithm,
02468                  GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02469                  1, 1);
02470 gtk_grid_attach (window->grid_algorithm,
02471                  GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02472                  1, 1);
02473 gtk_grid_attach (window->grid_algorithm,
02474                  GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475                  2, 1);
02476 gtk_grid_attach (window->grid_algorithm,
02477                  GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478                  2, 1);
02479 gtk_grid_attach (window->grid_algorithm,
02480                  GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481                  1, 1);
02482 gtk_grid_attach (window->grid_algorithm,
02483                  GTK_WIDGET (window->scrolled_threshold), 1,
02484                  NALGORITHMS + 11, 1, 1);
02485 window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486 gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487                  GTK_WIDGET (window->grid_algorithm));
02488
02489 // Creating the variable widgets
02490 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491 gtk_widget_set_tooltip_text
02492   (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493 window->id_variable = g_signal_connect
02494   (window->combo_variable, "changed", window_set_variable, NULL);
02495 window->button_add_variable
02496   = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497   GTK_ICON_SIZE_BUTTON);
02498 g_signal_connect
02499   (window->button_add_variable, "clicked",
02500   window_add_variable, NULL);
02500 gtk_widget_set_tooltip_text
02501   (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02502 window->button_remove_variable
02503   = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02504   GTK_ICON_SIZE_BUTTON);

```

```

02505 g_signal_connect
02506     (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
02507 gtk_widget_set_tooltip_text
02508     (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02509 window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02510 window->entry_variable = (GtkEntry *) gtk_entry_new ();
02511 gtk_widget_set_tooltip_text
02512     (GTK_WIDGET (window->entry_variable), _("Variable name"));
02513 gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02514 window->id_variable_label = g_signal_connect
02515     (window->entry_variable, "changed",
window_label_variable, NULL);
02516 window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02517 window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02518     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02519 gtk_widget_set_tooltip_text
02520     (GTK_WIDGET (window->spin_min),
    _("Minimum initial value of the variable"));
02521 window->scrolled_min
02522     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02523 gtk_container_add (GTK_CONTAINER (window->scrolled_min),
    GTK_WIDGET (window->spin_min));
02524 g_signal_connect (window->spin_min, "value-changed",
    window_rangemin_variable, NULL);
02525 window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02526 window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02527     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02528 gtk_widget_set_tooltip_text
02529     (GTK_WIDGET (window->spin_max),
    _("Maximum initial value of the variable"));
02530 window->scrolled_max
02531     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02532 gtk_container_add (GTK_CONTAINER (window->scrolled_max),
    GTK_WIDGET (window->spin_max));
02533 g_signal_connect (window->spin_max, "value-changed",
    window_rangemax_variable, NULL);
02534 window->check_minabs = (GtkCheckButton *)
02535     gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02536 g_signal_connect (window->check_minabs, "toggled",
window_update, NULL);
02537 window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02538     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02539 gtk_widget_set_tooltip_text
02540     (GTK_WIDGET (window->spin_minabs),
    _("Minimum allowed value of the variable"));
02541 window->scrolled_minabs
02542     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02543 gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
    GTK_WIDGET (window->spin_minabs));
02544 g_signal_connect (window->spin_minabs, "value-changed",
    window_rangeminabs_variable, NULL);
02545 window->check_maxabs = (GtkCheckButton *)
02546     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02547 g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02548 window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02549     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02550 gtk_widget_set_tooltip_text
02551     (GTK_WIDGET (window->spin_maxabs),
    _("Maximum allowed value of the variable"));
02552 window->scrolled_maxabs
02553     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02554 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
    GTK_WIDGET (window->spin_maxabs));
02555 g_signal_connect (window->spin_maxabs, "value-changed",
    window_rangemaxabs_variable, NULL);
02556 window->label_precision
02557     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02558 window->spin_precision = (GtkSpinButton *)
02559     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02560 gtk_widget_set_tooltip_text
02561     (GTK_WIDGET (window->spin_precision),
    _("Number of precision floating point digits\n"
    "0 is for integer numbers"));
02562 g_signal_connect (window->spin_precision, "value-changed",
    window_precision_variable, NULL);
02563 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02564 window->spin_sweeps =
02565     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02566 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
    _("Number of steps sweeping the variable"));
02567 g_signal_connect (window->spin_sweeps, "value-changed",
    window_update_variable, NULL);
02568 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02569 window->spin_bits
02570     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);

```

```

02588 gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590      _("Number of bits to encode the variable"));
02591 g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
;
02593 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596 gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598      _("Initial step size for the direction search method"));
02599 window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602     GTK_WIDGET (window->spin_step));
02603 g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606 gtk_grid_attach (window->grid_variable,
02607     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608 gtk_grid_attach (window->grid_variable,
02609     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610 gtk_grid_attach (window->grid_variable,
02611     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614 gtk_grid_attach (window->grid_variable,
02615     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616 gtk_grid_attach (window->grid_variable,
02617     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618 gtk_grid_attach (window->grid_variable,
02619     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620 gtk_grid_attach (window->grid_variable,
02621     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622 gtk_grid_attach (window->grid_variable,
02623     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624 gtk_grid_attach (window->grid_variable,
02625     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626 gtk_grid_attach (window->grid_variable,
02627     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628 gtk_grid_attach (window->grid_variable,
02629     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630 gtk_grid_attach (window->grid_variable,
02631     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632 gtk_grid_attach (window->grid_variable,
02633     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634 gtk_grid_attach (window->grid_variable,
02635     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636 gtk_grid_attach (window->grid_variable,
02637     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638 gtk_grid_attach (window->grid_variable,
02639     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640 gtk_grid_attach (window->grid_variable,
02641     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650     GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655     _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed",
window_set_experiment, NULL);
02658 window->button_add_experiment
02659     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660     GTK_ICON_SIZE_BUTTON);
02661 g_signal_connect
02662     (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02663 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664     _("Add experiment"));
02665 window->button_remove_experiment
02666     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667     GTK_ICON_SIZE_BUTTON);
02668 g_signal_connect (window->button_remove_experiment, "clicked",
02669     window_remove_experiment, NULL);
02670 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),

```

```

02671         _("Remove experiment"));
02672     window->label_experiment
02673     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674     window->button_experiment = (GtkFileChooserButton *)
02675     gtk_file_chooser_button_new (_("Experimental data file"),
02676     GTK_FILE_CHOOSER_ACTION_OPEN);
02677     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678     _("Experimental data file"));
02679     window->id_experiment_name
02680     = g_signal_connect (window->button_experiment, "selection-changed",
02681     window_name_experiment, NULL);
02682     gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683     window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684     window->spin_weight
02685     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686     gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->spin_weight),
02688     _("Weight factor to build the objective function"));
02689     g_signal_connect
02690     (window->spin_weight, "value-changed",
02691     window_weight_experiment, NULL);
02692     window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02693     gtk_grid_attach (window->grid_experiment,
02694     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02695     gtk_grid_attach (window->grid_experiment,
02696     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02697     gtk_grid_attach (window->grid_experiment,
02698     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02699 ;
02700     gtk_grid_attach (window->grid_experiment,
02701     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02702     gtk_grid_attach (window->grid_experiment,
02703     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02704     gtk_grid_attach (window->grid_experiment,
02705     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02706     gtk_grid_attach (window->grid_experiment,
02707     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02708     for (i = 0; i < MAX_NINPUTS; ++i)
02709     {
02710         snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02711         window->check_template[i] = (GtkCheckButton *)
02712         gtk_check_button_new_with_label (buffer3);
02713         window->id_template[i]
02714         = g_signal_connect (window->check_template[i], "toggled",
02715         window_inputs_experiment, NULL);
02716         gtk_grid_attach (window->grid_experiment,
02717         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02718         1);
02719         window->button_template[i] =
02720         (GtkFileChooserButton *)
02721         gtk_file_chooser_button_new (_("Input template"),
02722         GTK_FILE_CHOOSER_ACTION_OPEN);
02723         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02724         _("Experimental input template file"));
02725         window->id_input[i] =
02726         g_signal_connect_swapped (window->button_template[i],
02727         "selection-changed",
02728         (void (*)(void *)) window_template_experiment,
02729         (void *) (size_t) i);
02730         gtk_grid_attach (window->grid_experiment,
02731         GTK_WIDGET (window->button_template[i]),
02732         1, 3 + i, 3, 1);
02733     }
02734     window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02735     gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02736     GTK_WIDGET (window->grid_experiment));
02737 // Creating the error norm widgets
02738     window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02739     window->grid_norm = (GtkGrid *) gtk_grid_new ();
02740     gtk_container_add (GTK_CONTAINER (window->frame_norm),
02741     GTK_WIDGET (window->grid_norm));
02742     window->button_norm[0] = (GtkRadioButton *)
02743     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02744     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02745     tip_norm[0]);
02746     gtk_grid_attach (window->grid_norm,
02747     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02748     g_signal_connect (window->button_norm[0], "clicked",
02749     window_update, NULL);
02750     for (i = 0; ++i < NNORMS;)
02751     {
02752         window->button_norm[i] = (GtkRadioButton *)
02753         gtk_radio_button_new_with_mnemonic
02754         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02755         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02756         tip_norm[i]);

```

```

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

```

### 5.11.2.10 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 1904 of file [interface.c](#).

```

01905 {
01906     unsigned int i;
01907     char *buffer;
01908     #if DEBUG_INTERFACE
01909     fprintf (stderr, "window_read: start\n");
01910     #endif
01911
01912     // Reading new input file
01913     input_free ();
01914     if (!input_open (filename))
01915     {
01916         #if DEBUG_INTERFACE
01917         fprintf (stderr, "window_read: end\n");
01918         #endif
01919         return 0;
01920     }
01921
01922     // Setting GTK+ widgets data
01923     gtk_entry_set_text (window->entry_result, input->result);
01924     gtk_entry_set_text (window->entry_variables, input->
variables);
01925     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01926     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01927     g_free (buffer);
01928     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01929     if (input->evaluator)
01930     {
01931         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01932         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01933         g_free (buffer);
01934     }
01935     gtk_toggle_button_set_active
01936     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01937     switch (input->algorithm)
01938     {
01939     case ALGORITHM_MONTE_CARLO:
01940         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01941     case ALGORITHM_SWEEP:
01942         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01943         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01944         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01945         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01946         if (input->nsteps)
01947         {
01948             gtk_toggle_button_set_active
01949             (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01950             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01951             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01952             switch (input->direction)
01953             {
01954             case DIRECTION_METHOD_RANDOM:
01955                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01956             }
01957         }
01958     }
01959 }

```

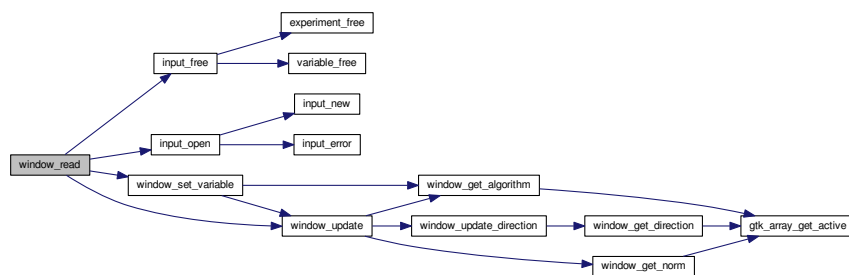


```

01967     }
01968     break;
01969     default:
01970         gtk_spin_button_set_value (window->spin_population,
01971                                   (gdouble) input->nsimulations);
01972         gtk_spin_button_set_value (window->spin_generations,
01973                                   (gdouble) input->niterations);
01974         gtk_spin_button_set_value (window->spin_mutation,
01975                                   input->mutation_ratio);
01976         gtk_spin_button_set_value (window->spin_reproduction,
01977                                   input->reproduction_ratio);
01978         gtk_spin_button_set_value (window->spin_adaptation,
01979                                   input->adaptation_ratio);
01980     }
01981     gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983     gtk_spin_button_set_value (window->spin_p, input->p);
01984     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986     g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
01987     gtk_combo_box_text_remove_all (window->combo_experiment);
01988     for (i = 0; i < input->nexperiments; ++i)
01989         gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i].name);
01990     g_signal_handler_unblock
01991     (window->button_experiment, window->
id_experiment_name);
01992     g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01993     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01994     g_signal_handler_block (window->combo_variable, window->
id_variable);
01995     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01996     gtk_combo_box_text_remove_all (window->combo_variable);
01997     for (i = 0; i < input->nvariables; ++i)
01998         gtk_combo_box_text_append_text (window->combo_variable,
input->variable[i].name);
01999     g_signal_handler_unblock (window->entry_variable,
window->id_variable_label);
02000     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02001     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02002     window_set_variable ();
02003     window_update ();
02004     #if DEBUG_INTERFACE
02005     fprintf (stderr, "window_read: end\n");
02006     #endif
02007     return 1;
02008 }

```

Here is the call graph for this function:



#### 5.11.2.11 window\_save()

```
int window_save ( )
```

Function to save the input file.



## Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file [interface.c](#).

```

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

```

```

00904         input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906     }
00907
00908     // Setting the algorithm
00909     switch (window_get_algorithm ())
00910     {
00911     case ALGORITHM_MONTE_CARLO:
00912         input->algorithm = ALGORITHM_MONTE_CARLO;
00913         input->nsimulations
00914             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915         input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917         input->tolerance =
00918             gtk_spin_button_get_value (window->spin_tolerance);
00919         input->nbest =
00920             gtk_spin_button_get_value_as_int (window->spin_bests);
00921         window_save_direction ();
00922         break;
00923     case ALGORITHM_SWEEP:
00924         input->algorithm = ALGORITHM_SWEEP;
00925         input->niterations
00926             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927         input->tolerance =
00928             gtk_spin_button_get_value (window->spin_tolerance);
00929         input->nbest =
00930             gtk_spin_button_get_value_as_int (window->spin_bests);
00931         window_save_direction ();
00932         break;
00933     default:
00934         input->algorithm = ALGORITHM_GENETIC;
00935         input->nsimulations
00936             = gtk_spin_button_get_value_as_int (window->spin_population);
00937         input->niterations
00938             = gtk_spin_button_get_value_as_int (window->spin_generations);
00939         input->mutation_ratio
00940             = gtk_spin_button_get_value (window->spin_mutation);
00941         input->reproduction_ratio
00942             = gtk_spin_button_get_value (window->spin_reproduction);
00943         input->adaptation_ratio
00944             = gtk_spin_button_get_value (window->spin_adaptation);
00945         break;
00946     }
00947     input->norm = window_get_norm ();
00948     input->p = gtk_spin_button_get_value (window->spin_p);
00949     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00950
00951     // Saving the XML file
00952     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953     input_save (buffer);
00954
00955     // Closing and freeing memory
00956     g_free (buffer);
00957     gtk_widget_destroy (GTK_WIDGET (dlg));
00958     #if DEBUG_INTERFACE
00959     fprintf (stderr, "window_save: end\n");
00960     #endif
00961     return 1;
00962 }
00963
00964 // Closing and freeing memory
00965 gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967 fprintf (stderr, "window_save: end\n");
00968 #endif
00969 return 0;
00970 }

```

#### 5.11.2.12 window\_template\_experiment()

```

void window_template_experiment (
    void * data )

```

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

#### Parameters

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

Definition at line 1542 of file [interface.c](#).

```

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

## 5.12 interface.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <gsl/gsl_rng.h>
00039 #include <libxml/parser.h>
00040 #include <libintl.h>
00041 #include <glib.h>
00042 #include <glib/gstdio.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
```

```

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 #00000000FFFF",
00122     "X      c #FFFF00000000",
00123     "                                     ",
00124     "                                     ",
00125     "                                     ",
00126     "      .      .      .      .      ",
00127     "      .      .      .      .      ",
00128     "      .      .      .      .      ",
00129     "      .      .      .      .      ",
00130     "      .      .      XXX      .      ",
00131     "      .      .      XXXXX      .      ",
00132     "      .      .      XXXXX      .      ",
00133     "      .      .      XXXXX      .      ",
00134     "      XXX      .      XXX      XXX      ",
00135     "      XXXXX      .      .      XXXXX      ",
00136     "      XXXXX      .      .      XXXXX      ",
00137     "      XXXXX      .      .      XXXXX      ",
00138     "      XXX      .      .      XXX      ",
00139     "      .      .      .      .      ",
00140     "      .      XXX      .      .      ",
00141     "      .      XXXXX      .      .      ",
00142     "      .      XXXXX      .      .      ",
00143     "      .      XXXXX      .      .      ",
00144     "      .      XXX      .      .      ",
00145     "      .      .      .      .      ",

```

```

00146 " . . . . . " ,
00147 " . . . . . " ,
00148 " . . . . . " ,
00149 " . . . . . " ,
00150 " . . . . . " ,
00151 " . . . . . " ,
00152 " . . . . . " ,
00153 " . . . . . " ,
00154 " . . . . . " };
00155 */
00156
00157 Options options[1];
00159 Running running[1];
00161 Window window[1];
00163
00170 void
00171 input_save_direction_xml (xmlNode * node)
00172 {
00173     #if DEBUG_INTERFACE
00174         fprintf (stderr, "input_save_direction_xml: start\n");
00175     #endif
00176     if (input->nsteps)
00177     {
00178         xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
00179 input->nsteps);
00179         if (input->relaxation != DEFAULT_RELAXATION)
00180             xml_node_set_float (node, (const xmlChar *)
00181 LABEL_RELAXATION,
00182 input->relaxation);
00183         switch (input->direction)
00184         {
00185             case DIRECTION_METHOD_COORDINATES:
00186                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00187 (const xmlChar *) LABEL_COORDINATES);
00188                 break;
00189             default:
00190                 xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
00191 (const xmlChar *) LABEL_RANDOM);
00192                 xml_node_set_uint (node, (const xmlChar *)
00193 LABEL_NESTIMATES,
00194 input->nestimates);
00195         }
00196     }
00197     #if DEBUG_INTERFACE
00198         fprintf (stderr, "input_save_direction_xml: end\n");
00199     #endif
00200 }
00201
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);
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
00237 }
00238
00242 void
00243 input_save_xml (xmlDoc * doc)
00244 {
00245     unsigned int i, j;
00246     char *buffer;
00247     xmlNode *node, *child;

```

```

00248     GFile *file, *file2;
00249
00250 #if DEBUG_INTERFACE
00251     fprintf(stderr, "input_save_xml: start\n");
00252 #endif
00253
00254     // Setting root XML node
00255     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00256     xmlDocSetRootElement (doc, node);
00257
00258     // Adding properties to the root XML node
00259     if (xmlStrcmp
00260         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00261         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00262             (xmlChar *) input->result);
00263     if (xmlStrcmp
00264         ((const xmlChar *) input->variables, (const xmlChar *)
00265             variables_name))
00266         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00267             (xmlChar *) input->variables);
00268     file = g_file_new_for_path (input->directory);
00269     file2 = g_file_new_for_path (input->simulator);
00270     buffer = g_file_get_relative_path (file, file2);
00271     g_object_unref (file2);
00272     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273     g_free (buffer);
00274     if (input->evaluator)
00275     {
00276         file2 = g_file_new_for_path (input->evaluator);
00277         buffer = g_file_get_relative_path (file, file2);
00278         g_object_unref (file2);
00279         if (xmlStrlen ((xmlChar *) buffer))
00280             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00281                 (xmlChar *) buffer);
00282         g_free (buffer);
00283     }
00284     if (input->seed != DEFAULT_RANDOM_SEED)
00285         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
00286             input->seed);
00287
00288     // Setting the algorithm
00289     buffer = (char *) g_slice_alloc (64);
00290     switch (input->algorithm)
00291     {
00292     case ALGORITHM_MONTE_CARLO:
00293         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00294             (const xmlChar *) LABEL_MONTE_CARLO);
00295         snprintf (buffer, 64, "%u", input->nsimulations);
00296         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00297             (xmlChar *) buffer);
00298         snprintf (buffer, 64, "%u", input->niterations);
00299         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00300             (xmlChar *) buffer);
00301         snprintf (buffer, 64, "%.3lg", input->tolerance);
00302         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00303             (xmlChar *) buffer);
00304         snprintf (buffer, 64, "%u", input->nbest);
00305         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00306         input_save_direction_xml (node);
00307         break;
00308     case ALGORITHM_SWEEP:
00309         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00310             (const xmlChar *) LABEL_SWEEP);
00311         snprintf (buffer, 64, "%u", input->nsimulations);
00312         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00313             (xmlChar *) buffer);
00314         snprintf (buffer, 64, "%.3lg", input->tolerance);
00315         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE,
00316             (xmlChar *) buffer);
00317         snprintf (buffer, 64, "%u", input->nbest);
00318         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00319         input_save_direction_xml (node);
00320         break;
00321     default:
00322         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00323             (const xmlChar *) LABEL_GENETIC);
00324         snprintf (buffer, 64, "%u", input->nsimulations);
00325         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00326             (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%u", input->niterations);
00328         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00329             (xmlChar *) buffer);
00330         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00331         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00332         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00333         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00334             (xmlChar *) buffer);

```

```

00333     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00334     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION,
00335                 (xmlChar *) buffer);
00336     break;
00337 }
00338 g_slice_free1 (64, buffer);
00339 if (input->threshold != 0.)
00340     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,
00341                         input->threshold);
00342
00343 // Setting the experimental data
00344 for (i = 0; i < input->nexperiments; ++i)
00345 {
00346     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00347     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00348                 (xmlChar *) input->experiment[i].name);
00349     if (input->experiment[i].weight != 1.)
00350         xml_node_set_float (child, (const xmlChar *)
LABEL_WEIGHT,
00351                             input->experiment[i].weight);
00352     for (j = 0; j < input->experiment->ninputs; ++j)
00353         xmlSetProp (child, (const xmlChar *) template[j],
00354                     (xmlChar *) input->experiment[i].template[j]);
00355 }
00356
00357 // Setting the variables data
00358 for (i = 0; i < input->nvariables; ++i)
00359 {
00360     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00361     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00362                 (xmlChar *) input->variable[i].name);
00363     xml_node_set_float (child, (const xmlChar *)
LABEL_MINIMUM,
00364                         input->variable[i].rangemin);
00365     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00366         xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MINIMUM,
00367                             input->variable[i].rangeminabs);
00368     xml_node_set_float (child, (const xmlChar *)
LABEL_MAXIMUM,
00369                         input->variable[i].rangemax);
00370     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00371         xml_node_set_float (child, (const xmlChar *)
LABEL_ABSOLUTE_MAXIMUM,
00372                             input->variable[i].rangemaxabs);
00373     if (input->variable[i].precision !=
DEFAULT_PRECISION)
00374         xml_node_set_uint (child, (const xmlChar *)
LABEL_PRECISION,
00375                             input->variable[i].precision);
00376     if (input->algorithm == ALGORITHM_SWEEP)
00377         xml_node_set_uint (child, (const xmlChar *)
LABEL_NSWEEPS,
00378                             input->variable[i].nsweeps);
00379     else if (input->algorithm == ALGORITHM_GENETIC)
00380         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00381                             input->variable[i].nbits);
00382     if (input->nsteps)
00383         xml_node_set_float (child, (const xmlChar *)
LABEL_STEP,
00384                             input->variable[i].step);
00385 }
00386
00387 // Saving the error norm
00388 switch (input->norm)
00389 {
00390     case ERROR_NORM_MAXIMUM:
00391         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00392                     (const xmlChar *) LABEL_MAXIMUM);
00393         break;
00394     case ERROR_NORM_P:
00395         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00396                     (const xmlChar *) LABEL_P);
00397         xml_node_set_float (node, (const xmlChar *) LABEL_P,
input->p);
00398         break;
00399     case ERROR_NORM_TAXICAB:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                     (const xmlChar *) LABEL_TAXICAB);
00402 }
00403
00404 #if DEBUG_INTERFACE
00405     fprintf (stderr, "input_save: end\n");
00406 #endif
00407 }
00408

```

```

00415 void
00416 input_save_json (JsonGenerator * generator)
00417 {
00418     unsigned int i, j;
00419     char *buffer;
00420     JsonNode *node, *child;
00421     JsonObject *object, *object2;
00422     JsonArray *array;
00423     GFile *file, *file2;
00424
00425     #if DEBUG_INTERFACE
00426         fprintf (stderr, "input_save_json: start\n");
00427     #endif
00428
00429     // Setting root JSON node
00430     node = json_node_new (JSON_NODE_OBJECT);
00431     object = json_node_get_object (node);
00432     json_generator_set_root (generator, node);
00433
00434     // Adding properties to the root JSON node
00435     if (strcmp (input->result, result_name))
00436         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00437     if (strcmp (input->variables, variables_name))
00438         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00439     file = g_file_new_for_path (input->directory);
00440     file2 = g_file_new_for_path (input->simulator);
00441     buffer = g_file_get_relative_path (file, file2);
00442     g_object_unref (file2);
00443     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00444     g_free (buffer);
00445     if (input->evaluator)
00446     {
00447         file2 = g_file_new_for_path (input->evaluator);
00448         buffer = g_file_get_relative_path (file, file2);
00449         g_object_unref (file2);
00450         if (strlen (buffer))
00451             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00452         g_free (buffer);
00453     }
00454     if (input->seed != DEFAULT_RANDOM_SEED)
00455         json_object_set_uint (object, LABEL_SEED,
input->seed);
00456
00457     // Setting the algorithm
00458     buffer = (char *) g_slice_alloc (64);
00459     switch (input->algorithm)
00460     {
00461         case ALGORITHM_MONTE_CARLO:
00462             json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00463             snprintf (buffer, 64, "%u", input->nsimulations);
00464             json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00465             snprintf (buffer, 64, "%u", input->niterations);
00466             json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00467             snprintf (buffer, 64, "%.3lg", input->tolerance);
00468             json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00469             snprintf (buffer, 64, "%u", input->nbest);
00470             json_object_set_string_member (object, LABEL_NBEST, buffer);
00471             input_save_direction_json (node);
00472             break;
00473         case ALGORITHM_SWEEP:
00474             json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00475             snprintf (buffer, 64, "%u", input->niterations);
00476             json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00477             snprintf (buffer, 64, "%.3lg", input->tolerance);
00478             json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00479             snprintf (buffer, 64, "%u", input->nbest);
00480             json_object_set_string_member (object, LABEL_NBEST, buffer);
00481             input_save_direction_json (node);
00482             break;
00483         default:
00484             json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00485             snprintf (buffer, 64, "%u", input->nsimulations);
00486             json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00487             snprintf (buffer, 64, "%u", input->niterations);
00488             json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00489             snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00490             json_object_set_string_member (object, LABEL_MUTATION, buffer);
00491             snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00492             json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00493             snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00494             json_object_set_string_member (object, LABEL_ADAPTATION, buffer);
00495             break;
00496     }
00497

```



```

00498     }
00499     g_slice_free1 (64, buffer);
00500     if (input->threshold != 0.)
00501         json_object_set_float (object, LABEL_THRESHOLD,
input->threshold);
00502
00503     // Setting the experimental data
00504     array = json_array_new ();
00505     for (i = 0; i < input->nexperiments; ++i)
00506     {
00507         child = json_node_new (JSON_NODE_OBJECT);
00508         object = json_node_get_object (child);
00509         json_object_set_string_member (object2, LABEL_NAME,
input->experiment[i].name);
00510
00511         if (input->experiment[i].weight != 1.)
00512             json_object_set_float (object2, LABEL_WEIGHT,
input->experiment[i].weight);
00513
00514         for (j = 0; j < input->experiment->ninputs; ++j)
00515             json_object_set_string_member (object2, template[j],
input->experiment[i].
template[j]);
00516
00517         json_array_add_element (array, child);
00518     }
00519     json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00520
00521     // Setting the variables data
00522     array = json_array_new ();
00523     for (i = 0; i < input->nvariables; ++i)
00524     {
00525         child = json_node_new (JSON_NODE_OBJECT);
00526         object = json_node_get_object (child);
00527         json_object_set_string_member (object2, LABEL_NAME,
input->variable[i].name);
00528
00529         json_object_set_float (object2, LABEL_MINIMUM,
input->variable[i].rangemin);
00530
00531         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00532             json_object_set_float (object2,
LABEL_ABSOLUTE_MINIMUM,
input->variable[i].rangeminabs);
00533
00534         json_object_set_float (object2, LABEL_MAXIMUM,
input->variable[i].rangemax);
00535
00536         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537             json_object_set_float (object2,
LABEL_ABSOLUTE_MAXIMUM,
input->variable[i].rangemaxabs);
00538
00539         if (input->variable[i].precision !=
DEFAULT_PRECISION)
00540             json_object_set_uint (object2, LABEL_PRECISION,
input->variable[i].precision);
00541
00542         if (input->algorithm == ALGORITHM_SWEEP)
00543             json_object_set_uint (object2, LABEL_NSWEEPS,
input->variable[i].nsweeps);
00544
00545         else if (input->algorithm == ALGORITHM_GENETIC)
00546             json_object_set_uint (object2, LABEL_NBITS,
input->variable[i].nbits);
00547
00548         if (input->nsteps)
00549             json_object_set_float (object, LABEL_STEP,
input->variable[i].step);
00550
00551         json_array_add_element (array, child);
00552     }
00553     json_object_set_array_member (object, LABEL_VARIABLES, array);
00554
00555     // Saving the error norm
00556     switch (input->norm)
00557     {
00558     case ERROR_NORM_MAXIMUM:
00559         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00560         break;
00561     case ERROR_NORM_P:
00562         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00563         json_object_set_float (object, LABEL_P, input->
p);
00564         break;
00565     case ERROR_NORM_TAXICAB:
00566         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00567         break;
00568     }
00569
00570     #if DEBUG_INTERFACE
00571     fprintf (stderr, "input_save_json: end\n");
00572     #endif
00573 }
00574
00575 void
00576 input_save (char *filename)
00577 {
00578     xmlDoc *doc;
00579     JsonGenerator *generator;

```

```

00583
00584 #if DEBUG_INTERFACE
00585     fprintf (stderr, "input_save: start\n");
00586 #endif
00587
00588 // Getting the input file directory
00589 input->name = g_path_get_basename (filename);
00590 input->directory = g_path_get_dirname (filename);
00591
00592 if (input->type == INPUT_TYPE_XML)
00593 {
00594     // Opening the input file
00595     doc = xmlNewDoc ((const xmlChar *) "1.0");
00596     input_save_xml (doc);
00597
00598     // Saving the XML file
00599     xmlSaveFormatFile (filename, doc, 1);
00600
00601     // Freeing memory
00602     xmlFreeDoc (doc);
00603 }
00604 else
00605 {
00606     // Opening the input file
00607     generator = json_generator_new ();
00608     json_generator_set_pretty (generator, TRUE);
00609     input_save_json (generator);
00610
00611     // Saving the JSON file
00612     json_generator_to_file (generator, filename, NULL);
00613
00614     // Freeing memory
00615     g_object_unref (generator);
00616 }
00617
00618 #if DEBUG_INTERFACE
00619     fprintf (stderr, "input_save: end\n");
00620 #endif
00621 }
00622
00627 void
00628 options_new ()
00629 {
00630 #if DEBUG_INTERFACE
00631     fprintf (stderr, "options_new: start\n");
00632 #endif
00633     options->label_seed = (GtkLabel *)
00634         gtk_label_new (_("Pseudo-random numbers generator seed"));
00635     options->spin_seed = (GtkSpinButton *)
00636         gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00637     gtk_widget_set_tooltip_text
00638         (GTK_WIDGET (options->spin_seed),
00639          _("Seed to init the pseudo-random numbers generator"));
00639     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
seed);
00641     options->label_threads = (GtkLabel *)
00642         gtk_label_new (_("Threads number for the stochastic algorithm"));
00643     options->spin_threads
00644         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00645     gtk_widget_set_tooltip_text
00646         (GTK_WIDGET (options->spin_threads),
00647          _("Number of threads to perform the calibration/optimization for "
00648            "the stochastic algorithm"));
00649     gtk_spin_button_set_value (options->spin_threads, (gdouble)
nthreads);
00650     options->label_direction = (GtkLabel *)
00651         gtk_label_new (_("Threads number for the direction search method"));
00652     options->spin_direction =
00653         (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00654     gtk_widget_set_tooltip_text (GTK_WIDGET (options->spin_direction),
00655         _("Number of threads to perform the calibration/optimization for "
00656           "the direction search method"));
00657     gtk_spin_button_set_value (options->spin_direction,
00658         (gdouble) nthreads_direction);
00659     options->grid = (GtkGrid *) gtk_grid_new ();
00660     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1,
00661         1);
00662     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1,
00663         1);
00664     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads), 0, 1,
00665         1, 1);
00666     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads), 1, 1, 1,
00667         1);
00668     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction), 0, 2,
00669         1, 1);
00670     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction), 1, 2,
00671

```

```

00672         1, 1);
00673     gtk_widget_show_all (GTK_WIDGET (options->grid));
00674     options->dialog = (GtkDialog *)
00675         gtk_dialog_new_with_buttons (_("Options"),
00676                                     window->window,
00677                                     GTK_DIALOG_MODAL,
00678                                     _("_OK"), GTK_RESPONSE_OK,
00679                                     _("_Cancel"), GTK_RESPONSE_CANCEL, NULL);
00680     gtk_container_add
00681         (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00682          GTK_WIDGET (options->grid));
00683     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00684     {
00685         input->seed
00686             = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00687         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00688         nthreads_direction
00689             = gtk_spin_button_get_value_as_int (options->spin_direction);
00690     }
00691     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00692     #if DEBUG_INTERFACE
00693     fprintf (stderr, "options_new: end\n");
00694     #endif
00695 }
00696
00701 void
00702 running_new ()
00703 {
00704     #if DEBUG_INTERFACE
00705     fprintf (stderr, "running_new: start\n");
00706     #endif
00707     running->label = (GtkLabel *) gtk_label_new (_("Calculating ..."));
00708     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00709     running->grid = (GtkGrid *) gtk_grid_new ();
00710     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00711     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00712     running->dialog = (GtkDialog *)
00713         gtk_dialog_new_with_buttons (_("Calculating"),
00714                                     window->window, GTK_DIALOG_MODAL, NULL,
00715                                     NULL);
00716     gtk_container_add (GTK_CONTAINER
00717         (gtk_dialog_get_content_area (running->dialog)),
00718         GTK_WIDGET (running->grid));
00719     gtk_spinner_start (running->spinner);
00720     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00721     #if DEBUG_INTERFACE
00722     fprintf (stderr, "running_new: end\n");
00723     #endif
00724 }
00725
00731 unsigned int
00732 window_get_algorithm ()
00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736     fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
00739                             NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740     fprintf (stderr, "window_get_algorithm: %u\n", i);
00741     fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }
00745
00751 unsigned int
00752 window_get_direction ()
00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_direction: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_direction,
00759                             NDIRECTIONS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_direction: %u\n", i);
00761     fprintf (stderr, "window_get_direction: end\n");
00762     #endif
00763     return i;
00764 }
00765
00771 unsigned int
00772 window_get_norm ()
00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE

```

```

00776     fprintf (stderr, "window_get_norm: start\n");
00777 #endif
00778     i = gtk_array_get_active (window->button_norm,
00779                             NNORMS);
00779 #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782 #endif
00783     return i;
00784 }
00785
00790 void
00791 window_save_direction ()
00792 {
00793     #if DEBUG_INTERFACE
00794     fprintf (stderr, "window_save_direction: start\n");
00795     #endif
00796     if (gtk_toggle_button_get_active
00797         (GTK_TOGGLE_BUTTON (window->check_direction)))
00798     {
00799         input->nsteps = gtk_spin_button_get_value_as_int (window->
00800 spin_steps);
00801         input->relaxation = gtk_spin_button_get_value (window->
00802 spin_relaxation);
00803         switch (window_get_direction ())
00804         {
00805             case DIRECTION_METHOD_COORDINATES:
00806                 input->direction = DIRECTION_METHOD_COORDINATES;
00807                 break;
00808             default:
00809                 input->direction = DIRECTION_METHOD_RANDOM;
00810                 input->nestimates
00811                     = gtk_spin_button_get_value_as_int (window->spin_estimates);
00812         }
00813     }
00814     else
00815         input->nsteps = 0;
00816 #if DEBUG_INTERFACE
00817     fprintf (stderr, "window_save_direction: end\n");
00818 #endif
00819 }
00820
00824 int
00825 window_save ()
00826 {
00827     GtkFileChooserDialog *dlg;
00828     GtkFileFilter *filter1, *filter2;
00829     char *buffer;
00830
00831     #if DEBUG_INTERFACE
00832     fprintf (stderr, "window_save: start\n");
00833     #endif
00834
00835     // Opening the saving dialog
00836     dlg = (GtkFileChooserDialog *)
00837         gtk_file_chooser_dialog_new (_("Save file"),
00838                                     window->window,
00839                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00840                                     _("_Cancel"),
00841                                     GTK_RESPONSE_CANCEL,
00842                                     _("_OK"), GTK_RESPONSE_OK, NULL);
00843     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg),
00844 TRUE);
00845     buffer = g_build_filename (input->directory, input->name, NULL);
00846     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00847     g_free (buffer);
00848
00849     // Adding XML filter
00850     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00851     gtk_file_filter_set_name (filter1, "XML");
00852     gtk_file_filter_add_pattern (filter1, "*.xml");
00853     gtk_file_filter_add_pattern (filter1, "*.XML");
00854     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00855
00856     // Adding JSON filter
00857     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00858     gtk_file_filter_set_name (filter2, "JSON");
00859     gtk_file_filter_add_pattern (filter2, "*.json");
00860     gtk_file_filter_add_pattern (filter2, "*.JSON");
00861     gtk_file_filter_add_pattern (filter2, "*.js");
00862     gtk_file_filter_add_pattern (filter2, "*.JS");
00863     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865     if (input->type == INPUT_TYPE_XML)
00866         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867     else
00868         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);

```

```

00869
00870 // If OK response then saving
00871 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872 {
00873     // Setting input file type
00874     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875     buffer = (char *) gtk_file_filter_get_name (filter1);
00876     if (!strcmp (buffer, "XML"))
00877         input->type = INPUT_TYPE_XML;
00878     else
00879         input->type = INPUT_TYPE_JSON;
00880
00881     // Adding properties to the root XML node
00882     input->simulator = gtk_file_chooser_get_filename
00883         (GTK_FILE_CHOOSER (window->button_simulator));
00884     if (gtk_toggle_button_get_active
00885         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886         input->evaluator = gtk_file_chooser_get_filename
00887             (GTK_FILE_CHOOSER (window->button_evaluator));
00888     else
00889         input->evaluator = NULL;
00890     if (input->type == INPUT_TYPE_XML)
00891     {
00892         input->result
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                                     gtk_entry_get_text (window->entry_result));
00895         input->variables
00896             = (char *) xmlStrdup ((const xmlChar *)
00897                                     gtk_entry_get_text
00898                                         (window->entry_variables));
00899     }
00900     else
00901     {
00902         input->result =
00903             g_strdup (gtk_entry_get_text (window->entry_result));
00904         input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906     }
00907
00908     // Setting the algorithm
00909     switch (window_get_algorithm ())
00910     {
00911         case ALGORITHM_MONTE_CARLO:
00912             input->algorithm = ALGORITHM_MONTE_CARLO;
00913             input->nsimulations
00914                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915             input->niterations
00916                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917             input->tolerance =
00918                 gtk_spin_button_get_value (window->spin_tolerance);
00919             input->nbest =
00920                 gtk_spin_button_get_value_as_int (window->spin_bests);
00921             window_save_direction ();
00922             break;
00923         case ALGORITHM_SWEEP:
00924             input->algorithm = ALGORITHM_SWEEP;
00925             input->niterations
00926                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927             input->tolerance =
00928                 gtk_spin_button_get_value (window->spin_tolerance);
00929             input->nbest =
00930                 gtk_spin_button_get_value_as_int (window->spin_bests);
00931             window_save_direction ();
00932             break;
00933         default:
00934             input->algorithm = ALGORITHM_GENETIC;
00935             input->nsimulations
00936                 = gtk_spin_button_get_value_as_int (window->spin_population);
00937             input->niterations
00938                 = gtk_spin_button_get_value_as_int (window->spin_generations);
00939             input->mutation_ratio
00940                 = gtk_spin_button_get_value (window->spin_mutation);
00941             input->reproduction_ratio
00942                 = gtk_spin_button_get_value (window->spin_reproduction);
00943             input->adaptation_ratio
00944                 = gtk_spin_button_get_value (window->spin_adaptation);
00945             break;
00946     }
00947     input->norm = window_get_norm ();
00948     input->p = gtk_spin_button_get_value (window->spin_p);
00949     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00950
00951     // Saving the XML file
00952     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953     input_save (buffer);
00954

```

```

00955     // Closing and freeing memory
00956     g_free (buffer);
00957     gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959     fprintf (stderr, "window_save: end\n");
00960 #endif
00961     return 1;
00962 }
00963
00964 // Closing and freeing memory
00965 gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967 fprintf (stderr, "window_save: end\n");
00968 #endif
00969 return 0;
00970 }
00971
00972 void
00973 window_run ()
00974 {
00975     unsigned int i;
00976     char *msg, *msg2, buffer[64], buffer2[64];
00977 #if DEBUG_INTERFACE
00978     fprintf (stderr, "window_run: start\n");
00979 #endif
00980 if (!window_save ())
00981 {
00982 #if DEBUG_INTERFACE
00983     fprintf (stderr, "window_run: end\n");
00984 #endif
00985     return;
00986 }
00987 running_new ();
00988 while (gtk_events_pending ())
00989     gtk_main_iteration ();
00990 optimize_open ();
00991 #if DEBUG_INTERFACE
00992     fprintf (stderr, "window_run: closing running dialog\n");
00993 #endif
00994 gtk_spinner_stop (running->spinner);
00995 gtk_widget_destroy (GTK_WIDGET (running->dialog));
00996 #if DEBUG_INTERFACE
00997     fprintf (stderr, "window_run: displaying results\n");
00998 #endif
00999 snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
01000 msg2 = g_strdup (buffer);
01001 for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
01002 {
01003     snprintf (buffer, 64, "%s = %s\n",
01004             input->variable[i].name,
01005             format[input->variable[i].precision]);
01006     snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01007     msg = g_strconcat (msg2, buffer2, NULL);
01008     g_free (msg2);
01009 }
01010 snprintf (buffer, 64, "%s = %.6lg s", _("Calculation time"),
01011         optimize->calculation_time);
01012 msg = g_strconcat (msg2, buffer, NULL);
01013 g_free (msg2);
01014 show_message (_("Best result"), msg, INFO_TYPE);
01015 g_free (msg);
01016 #if DEBUG_INTERFACE
01017     fprintf (stderr, "window_run: freeing memory\n");
01018 #endif
01019 optimize_free ();
01020 #if DEBUG_INTERFACE
01021     fprintf (stderr, "window_run: end\n");
01022 #endif
01023 }
01024
01025 void
01026 window_help ()
01027 {
01028     char *buffer, *buffer2;
01029 #if DEBUG_INTERFACE
01030     fprintf (stderr, "window_help: start\n");
01031 #endif
01032 buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01033         _("user-manual.pdf"), NULL);
01034 buffer = g_filename_to_uri (buffer2, NULL, NULL);
01035 g_free (buffer2);
01036 gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01037 #if DEBUG_INTERFACE
01038     fprintf (stderr, "window_help: uri=%s\n", buffer);
01039 #endif
01040 g_free (buffer);
01041 #if DEBUG_INTERFACE

```

```

01050     fprintf (stderr, "window_help: end\n");
01051 #endif
01052 }
01053
01054 void
01055 window_about ()
01056 {
01057     static const gchar *authors[] = {
01058         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01059         "Borja Latorre Garcés <borja.latorre@csic.es>",
01060         NULL
01061     };
01062 #if DEBUG_INTERFACE
01063     fprintf (stderr, "window_about: start\n");
01064 #endif
01065     gtk_show_about_dialog
01066     (window->window,
01067      "program_name", "MPCOTool",
01068      "comments",
01069      _("The Multi-Purposes Calibration and Optimization Tool.\n"
01070       "A software to perform calibrations or optimizations of empirical"
01071       " parameters"),
01072      "authors", authors,
01073      "translator-credits",
01074      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01075      "(english, french and spanish)\n"
01076      "Uğur Çayoğlu (german)",
01077      "version", "3.4.0",
01078      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01079      "logo", window->logo,
01080      "website", "https://github.com/jburguete/mpcotool",
01081      "license-type", GTK_LICENSE_BSD, NULL);
01082 #if DEBUG_INTERFACE
01083     fprintf (stderr, "window_about: end\n");
01084 #endif
01085 }
01086
01087 void
01088 window_update_direction ()
01089 {
01090 #if DEBUG_INTERFACE
01091     fprintf (stderr, "window_update_direction: start\n");
01092 #endif
01093     gtk_widget_show (GTK_WIDGET (window->check_direction));
01094     if (gtk_toggle_button_get_active
01095         (GTK_TOGGLE_BUTTON (window->check_direction)))
01096     {
01097         gtk_widget_show (GTK_WIDGET (window->grid_direction));
01098         gtk_widget_show (GTK_WIDGET (window->label_step));
01099         gtk_widget_show (GTK_WIDGET (window->spin_step));
01100     }
01101     switch (window_get_direction ())
01102     {
01103     case DIRECTION_METHOD_COORDINATES:
01104         gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01105         gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01106         break;
01107     default:
01108         gtk_widget_show (GTK_WIDGET (window->label_estimates));
01109         gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01110     }
01111 #if DEBUG_INTERFACE
01112     fprintf (stderr, "window_update_direction: end\n");
01113 #endif
01114 }
01115
01116 void
01117 window_update ()
01118 {
01119     unsigned int i;
01120 #if DEBUG_INTERFACE
01121     fprintf (stderr, "window_update: start\n");
01122 #endif
01123     gtk_widget_set_sensitive
01124     (GTK_WIDGET (window->button_evaluator),
01125      gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01126      (window->check_evaluator)));
01127     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01128     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01129     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01130     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01131     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01132     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01133     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01134     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01135     gtk_widget_hide (GTK_WIDGET (window->label_population));
01136     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01137 }

```

```

01150 gtk_widget_hide (GTK_WIDGET (window->label_generations));
01151 gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01152 gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01153 gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01154 gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01155 gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01156 gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01157 gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01158 gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01159 gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01160 gtk_widget_hide (GTK_WIDGET (window->label_bits));
01161 gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01162 gtk_widget_hide (GTK_WIDGET (window->check_direction));
01163 gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01164 gtk_widget_hide (GTK_WIDGET (window->label_step));
01165 gtk_widget_hide (GTK_WIDGET (window->spin_step));
01166 gtk_widget_hide (GTK_WIDGET (window->label_p));
01167 gtk_widget_hide (GTK_WIDGET (window->spin_p));
01168 i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01169 switch (window_get_algorithm ())
01170 {
01171     case ALGORITHM_MONTE_CARLO:
01172         gtk_widget_show (GTK_WIDGET (window->label_simulations));
01173         gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01174         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01175         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01176         if (i > 1)
01177         {
01178             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01179             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01180             gtk_widget_show (GTK_WIDGET (window->label_bests));
01181             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01182         }
01183         window_update_direction ();
01184         break;
01185     case ALGORITHM_SWEEP:
01186         gtk_widget_show (GTK_WIDGET (window->label_iterations));
01187         gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01188         if (i > 1)
01189         {
01190             gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01191             gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01192             gtk_widget_show (GTK_WIDGET (window->label_bests));
01193             gtk_widget_show (GTK_WIDGET (window->spin_bests));
01194         }
01195         gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01196         gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01197         gtk_widget_show (GTK_WIDGET (window->check_direction));
01198         window_update_direction ();
01199         break;
01200     default:
01201         gtk_widget_show (GTK_WIDGET (window->label_population));
01202         gtk_widget_show (GTK_WIDGET (window->spin_population));
01203         gtk_widget_show (GTK_WIDGET (window->label_generations));
01204         gtk_widget_show (GTK_WIDGET (window->spin_generations));
01205         gtk_widget_show (GTK_WIDGET (window->label_mutation));
01206         gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01207         gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01208         gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01209         gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01210         gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01211         gtk_widget_show (GTK_WIDGET (window->label_bits));
01212         gtk_widget_show (GTK_WIDGET (window->spin_bits));
01213     }
01214     gtk_widget_set_sensitive
01215     (GTK_WIDGET (window->button_remove_experiment),
01216      input->nexperiments > 1);
01217     gtk_widget_set_sensitive
01218     (GTK_WIDGET (window->button_remove_variable), input->
01219      nvariables > 1);
01220     for (i = 0; i < input->experiment->ninputs; ++i)
01221     {
01222         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01223         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01224         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01225         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01226         g_signal_handler_block
01227         (window->check_template[i], window->id_template[i]);
01228         g_signal_handler_block (window->button_template[i],
01229                                window->id_input[i]);
01230         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01231                                     (window->check_template[i]), 1);
01232         g_signal_handler_unblock (window->button_template[i],
01233                                  window->id_input[i]);
01234         g_signal_handler_unblock (window->check_template[i],
01235                                  window->id_template[i]);
01236     }

```



```

01235     if (i > 0)
01236     {
01237         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]),
01238             1);
01239         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i - 1]),
01240             gtk_toggle_button_get_active
01241             GTK_TOGGLE_BUTTON (window->check_template
01242                 [i - 1]));
01243     }
01244     if (i < MAX_NINPUTS)
01245     {
01246         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01247         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01248         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01249         gtk_widget_set_sensitive
01250             (GTK_WIDGET (window->button_template[i]),
01251             gtk_toggle_button_get_active
01252             GTK_TOGGLE_BUTTON (window->check_template[i]));
01253         g_signal_handler_block
01254             (window->check_template[i], window->id_template[i]);
01255         g_signal_handler_block (window->button_template[i],
01256             window->id_input[i]);
01257         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01258             (window->check_template[i]), 0);
01259         g_signal_handler_unblock (window->button_template[i],
01260             window->id_input[i]);
01261         g_signal_handler_unblock (window->check_template[i],
01262             window->id_template[i]);
01263     }
01264     while (++i < MAX_NINPUTS)
01265     {
01266         gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01267         gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01268     }
01269     gtk_widget_set_sensitive
01270         (GTK_WIDGET (window->spin_minabs),
01271         gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01272     gtk_widget_set_sensitive
01273         (GTK_WIDGET (window->spin_maxabs),
01274         gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01275     if (window_get_norm () == ERROR_NORM_P)
01276     {
01277         gtk_widget_show (GTK_WIDGET (window->label_p));
01278         gtk_widget_show (GTK_WIDGET (window->spin_p));
01279     }
01280 #if DEBUG_INTERFACE
01281     fprintf (stderr, "window_update: end\n");
01282 #endif
01283 }
01284
01285 void
01286 window_set_algorithm ()
01287 {
01288     int i;
01289 #if DEBUG_INTERFACE
01290     fprintf (stderr, "window_set_algorithm: start\n");
01291 #endif
01292     i = window_get_algorithm ();
01293     switch (i)
01294     {
01295     case ALGORITHM_SWEEP:
01296         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01297         if (i < 0)
01298             i = 0;
01299         gtk_spin_button_set_value (window->spin_sweeps,
01300             (gdouble) input->variable[i].
01301             nsweeps);
01302         break;
01303     case ALGORITHM_GENETIC:
01304         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01305         if (i < 0)
01306             i = 0;
01307         gtk_spin_button_set_value (window->spin_bits,
01308             (gdouble) input->variable[i].nbits);
01309     }
01310     window_update ();
01311 #if DEBUG_INTERFACE
01312     fprintf (stderr, "window_set_algorithm: end\n");
01313 #endif
01314 }
01315
01316 void
01317 window_set_experiment ()
01318 {
01319     unsigned int i, j;
01320     char *buffer1, *buffer2;
01321 #if DEBUG_INTERFACE

```

```

01329     fprintf (stderr, "window_set_experiment: start\n");
01330 #endif
01331     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01332     gtk_spin_button_set_value (window->spin_weight,
01333                               input->experiment[i].weight);
01334     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01335     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01336     g_free (buffer1);
01337     g_signal_handler_block
01338     (window->button_experiment, window->id_experiment_name);
01339     gtk_file_chooser_set_filename
01340     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01341     g_signal_handler_unblock
01342     (window->button_experiment, window->id_experiment_name);
01343     g_free (buffer2);
01344     for (j = 0; j < input->experiment->ninputs; ++j)
01345     {
01346         g_signal_handler_block (window->button_template[j],
01347                                 window->id_input[j]);
01348         buffer2 =
01349         g_build_filename (input->directory, input->experiment[i].
01350         template[j],
01351                           NULL);
01352         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01353                                       (window->button_template[j]), buffer2);
01354         g_free (buffer2);
01355         g_signal_handler_unblock
01356         (window->button_template[j], window->id_input[j]);
01357     }
01358 #if DEBUG_INTERFACE
01359     fprintf (stderr, "window_set_experiment: end\n");
01360 #endif
01361 }
01362 void
01363 window_remove_experiment ()
01364 {
01365     unsigned int i, j;
01366 #if DEBUG_INTERFACE
01367     fprintf (stderr, "window_remove_experiment: start\n");
01368 #endif
01369     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01370     g_signal_handler_block (window->combo_experiment, window->
01371                             id_experiment);
01372     gtk_combo_box_text_remove (window->combo_experiment, i);
01373     g_signal_handler_unblock (window->combo_experiment, window->
01374                             id_experiment);
01375     experiment_free (input->experiment + i, input->
01376                     type);
01377     --input->nexperiments;
01378     for (j = i; j < input->nexperiments; ++j)
01379         memcpy (input->experiment + j, input->experiment + j + 1,
01380               sizeof (Experiment));
01381     j = input->nexperiments - 1;
01382     if (i > j)
01383         i = j;
01384     for (j = 0; j < input->experiment->ninputs; ++j)
01385         g_signal_handler_block (window->button_template[j], window->
01386                                 id_input[j]);
01387     g_signal_handler_block
01388     (window->button_experiment, window->id_experiment_name);
01389     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01390     g_signal_handler_unblock
01391     (window->button_experiment, window->id_experiment_name);
01392     for (j = 0; j < input->experiment->ninputs; ++j)
01393         g_signal_handler_unblock (window->button_template[j],
01394                                 window->id_input[j]);
01395     window_update ();
01396 #if DEBUG_INTERFACE
01397     fprintf (stderr, "window_remove_experiment: end\n");
01398 #endif
01399 }
01400 void
01401 window_add_experiment ()
01402 {
01403     unsigned int i, j;
01404 #if DEBUG_INTERFACE
01405     fprintf (stderr, "window_add_experiment: start\n");
01406 #endif
01407     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01408     g_signal_handler_block (window->combo_experiment, window->
01409                             id_experiment);
01410     gtk_combo_box_text_insert_text
01411     (window->combo_experiment, i, input->experiment[i].
01412     name);
01413     g_signal_handler_unblock (window->combo_experiment, window->

```

```

    id_experiment);
01417     input->experiment = (Experiment *) g_realloc
01418     (input->experiment, (input->nexperiments + 1) * sizeof (
Experiment));
01419     for (j = input->nexperiments - 1; j > i; --j)
01420         memcpy (input->experiment + j + 1, input->experiment + j,
01421                 sizeof (Experiment));
01422     input->experiment[j + 1].weight = input->experiment[j].
weight;
01423     input->experiment[j + 1].ninputs = input->
experiment[j].ninputs;
01424     if (input->type == INPUT_TYPE_XML)
01425     {
01426         input->experiment[j + 1].name
01427         = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
name);
01428         for (j = 0; j < input->experiment->ninputs; ++j)
01429             input->experiment[i + 1].template[j]
01430             = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
template[j]);
01431     }
01432     else
01433     {
01434         input->experiment[j + 1].name = g_strdup (input->
experiment[j].name);
01435         for (j = 0; j < input->experiment->ninputs; ++j)
01436             input->experiment[i + 1].template[j]
01437             = g_strdup (input->experiment[i].template[j]);
01438     }
01439     ++input->nexperiments;
01440     for (j = 0; j < input->experiment->ninputs; ++j)
01441         g_signal_handler_block (window->button_template[j], window->
id_input[j]);
01442     g_signal_handler_block
01443     (window->button_experiment, window->id_experiment_name);
01444     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01445     g_signal_handler_unblock
01446     (window->button_experiment, window->id_experiment_name);
01447     for (j = 0; j < input->experiment->ninputs; ++j)
01448         g_signal_handler_unblock (window->button_template[j],
01449                                   window->id_input[j]);
01450     window_update ();
01451 #if DEBUG_INTERFACE
01452     fprintf (stderr, "window_add_experiment: end\n");
01453 #endif
01454 }
01455
01460 void
01461 window_name_experiment ()
01462 {
01463     unsigned int i;
01464     char *buffer;
01465     GFile *file1, *file2;
01466 #if DEBUG_INTERFACE
01467     fprintf (stderr, "window_name_experiment: start\n");
01468 #endif
01469     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01470     file1
01471     =
01472     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01473     file2 = g_file_new_for_path (input->directory);
01474     buffer = g_file_get_relative_path (file2, file1);
01475     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01476     gtk_combo_box_text_remove (window->combo_experiment, i);
01477     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01478     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01479     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01480     g_free (buffer);
01481     g_object_unref (file2);
01482     g_object_unref (file1);
01483 #if DEBUG_INTERFACE
01484     fprintf (stderr, "window_name_experiment: end\n");
01485 #endif
01486 }
01487
01492 void
01493 window_weight_experiment ()
01494 {
01495     unsigned int i;
01496 #if DEBUG_INTERFACE
01497     fprintf (stderr, "window_weight_experiment: start\n");
01498 #endif
01499     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01500     input->experiment[i].weight =
01501     gtk_spin_button_get_value (window->spin_weight);

```

```

01502 #if DEBUG_INTERFACE
01503     fprintf (stderr, "window_weight_experiment: end\n");
01504 #endif
01505 }
01506
01512 void
01513 window_inputs_experiment ()
01514 {
01515     unsigned int j;
01516 #if DEBUG_INTERFACE
01517     fprintf (stderr, "window_inputs_experiment: start\n");
01518 #endif
01519     j = input->experiment->ninputs - 1;
01520     if (j
01521         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01522                                             (window->check_template[j])))
01523         --input->experiment->ninputs;
01524     if (input->experiment->ninputs < MAX_NINPUTS
01525         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01526                                             (window->check_template[j])))
01527         ++input->experiment->ninputs;
01528     window_update ();
01529 #if DEBUG_INTERFACE
01530     fprintf (stderr, "window_inputs_experiment: end\n");
01531 #endif
01532 }
01533
01541 void
01542 window_template_experiment (void *data)
01543 {
01544     unsigned int i, j;
01545     char *buffer;
01546     GFile *file1, *file2;
01547 #if DEBUG_INTERFACE
01548     fprintf (stderr, "window_template_experiment: start\n");
01549 #endif
01550     i = (size_t) data;
01551     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01552     file1
01553     =
01554     gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01555     file2 = g_file_new_for_path (input->directory);
01556     buffer = g_file_get_relative_path (file2, file1);
01557     if (input->type == INPUT_TYPE_XML)
01558         input->experiment[j].template[i] =
01559         (char *) xmlStrdup ((xmlChar *) buffer);
01560     else
01561         input->experiment[j].template[i] = g_strdup (buffer);
01562     g_free (buffer);
01563     g_object_unref (file2);
01564     g_object_unref (file1);
01565 #if DEBUG_INTERFACE
01566     fprintf (stderr, "window_template_experiment: end\n");
01567 #endif
01568 }
01569
01574 void
01575 window_set_variable ()
01576 {
01577     unsigned int i;
01578 #if DEBUG_INTERFACE
01579     fprintf (stderr, "window_set_variable: start\n");
01580 #endif
01581     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01582     id_signal_handler_block (window->entry_variable, window->
01583                             id_variable_label);
01584     gtk_entry_set_text (window->entry_variable, input->variable[i].
01585                         name);
01586     g_signal_handler_unblock (window->entry_variable,
01587                               window->id_variable_label);
01588     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01589                               rangemin);
01590     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01591                               rangemax);
01592     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01593     {
01594         gtk_spin_button_set_value (window->spin_minabs,
01595                                     input->variable[i].rangeminabs);
01596         gtk_toggle_button_set_active
01597             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01598     }
01599     else
01600     {
01601         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01602         gtk_toggle_button_set_active
01603             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01604     }

```

```

01601     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01602     {
01603         gtk_spin_button_set_value (window->spin_maxabs,
01604             input->variable[i].rangemaxabs);
01605         gtk_toggle_button_set_active
01606             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01607     }
01608     else
01609     {
01610         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01611         gtk_toggle_button_set_active
01612             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01613     }
01614     gtk_spin_button_set_value (window->spin_precision,
01615         input->variable[i].precision);
01616     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
nsteps);
01617     if (input->nsteps)
01618         gtk_spin_button_set_value (window->spin_step, input->variable[i].
step);
01619     #if DEBUG_INTERFACE
01620         fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01621             input->variable[i].precision);
01622     #endif
01623     switch (window_get_algorithm ())
01624     {
01625         case ALGORITHM_SWEEP:
01626             gtk_spin_button_set_value (window->spin_sweeps,
01627                 (gdouble) input->variable[i].
nsweeps);
01628     #if DEBUG_INTERFACE
01629         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01630             input->variable[i].nsweeps);
01631     #endif
01632         break;
01633         case ALGORITHM_GENETIC:
01634             gtk_spin_button_set_value (window->spin_bits,
01635                 (gdouble) input->variable[i].nbits);
01636     #if DEBUG_INTERFACE
01637         fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01638             input->variable[i].nbits);
01639     #endif
01640         break;
01641     }
01642     window_update ();
01643     #if DEBUG_INTERFACE
01644         fprintf (stderr, "window_set_variable: end\n");
01645     #endif
01646 }
01647
01652 void
01653 window_remove_variable ()
01654 {
01655     unsigned int i, j;
01656     #if DEBUG_INTERFACE
01657         fprintf (stderr, "window_remove_variable: start\n");
01658     #endif
01659     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01660     g_signal_handler_block (window->combo_variable, window->
id_variable);
01661     gtk_combo_box_text_remove (window->combo_variable, i);
01662     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01663     xmlFree (input->variable[i].name);
01664     --input->nvariables;
01665     for (j = i; j < input->nvariables; ++j)
01666         memcpy (input->variable + j, input->variable + j + 1, sizeof (
Variable));
01667     j = input->nvariables - 1;
01668     if (i > j)
01669         i = j;
01670     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01671     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01672     g_signal_handler_unblock (window->entry_variable,
01673         window->id_variable_label);
01674     window_update ();
01675     #if DEBUG_INTERFACE
01676         fprintf (stderr, "window_remove_variable: end\n");
01677     #endif
01678 }
01679
01684 void
01685 window_add_variable ()
01686 {
01687     unsigned int i, j;
01688     #if DEBUG_INTERFACE

```

```

01689     fprintf (stderr, "window_add_variable: start\n");
01690 #endif
01691     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01692     g_signal_handler_block (window->combo_variable, window->
01693         id_variable);
01693     gtk_combo_box_text_insert_text (window->combo_variable, i,
01694         input->variable[i].name);
01695     g_signal_handler_unblock (window->combo_variable, window->
01696         id_variable);
01696     input->variable = (Variable *) g_realloc
01697         (input->variable, (input->nvariables + 1) * sizeof (
01698         Variable));
01698     for (j = input->nvariables - 1; j > i; --j)
01699         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01700         Variable));
01700     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01701         Variable));
01701     if (input->type == INPUT_TYPE_XML)
01702         input->variable[j + 1].name
01703             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01704     else
01705         input->variable[j + 1].name = g_strdup (input->
01706         variable[j].name);
01706     ++input->nvariables;
01707     g_signal_handler_block (window->entry_variable, window->
01708         id_variable_label);
01708     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01709     g_signal_handler_unblock (window->entry_variable,
01710         window->id_variable_label);
01711     window_update ();
01712 #if DEBUG_INTERFACE
01713     fprintf (stderr, "window_add_variable: end\n");
01714 #endif
01715 }
01716
01721 void
01722 window_label_variable ()
01723 {
01724     unsigned int i;
01725     const char *buffer;
01726 #if DEBUG_INTERFACE
01727     fprintf (stderr, "window_label_variable: start\n");
01728 #endif
01729     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01730     buffer = gtk_entry_get_text (window->entry_variable);
01731     g_signal_handler_block (window->combo_variable, window->
01732         id_variable);
01732     gtk_combo_box_text_remove (window->combo_variable, i);
01733     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01734     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01735     g_signal_handler_unblock (window->combo_variable, window->
01736         id_variable);
01736 #if DEBUG_INTERFACE
01737     fprintf (stderr, "window_label_variable: end\n");
01738 #endif
01739 }
01740
01745 void
01746 window_precision_variable ()
01747 {
01748     unsigned int i;
01749 #if DEBUG_INTERFACE
01750     fprintf (stderr, "window_precision_variable: start\n");
01751 #endif
01752     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01753     input->variable[i].precision
01754         =
01755         (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01756     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
01757         precision);
01757     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
01758         precision);
01758     gtk_spin_button_set_digits (window->spin_minabs,
01759         input->variable[i].precision);
01760     gtk_spin_button_set_digits (window->spin_maxabs,
01761         input->variable[i].precision);
01762 #if DEBUG_INTERFACE
01763     fprintf (stderr, "window_precision_variable: end\n");
01764 #endif
01765 }
01766
01771 void
01772 window_rangemin_variable ()
01773 {
01774     unsigned int i;
01775 #if DEBUG_INTERFACE
01776     fprintf (stderr, "window_rangemin_variable: start\n");

```

```

01777 #endif
01778     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01779     input->variable[i].rangemin = gtk_spin_button_get_value (window->
    spin_min);
01780 #if DEBUG_INTERFACE
01781     fprintf (stderr, "window_rangemin_variable: end\n");
01782 #endif
01783 }
01784
01785 void
01790 window_rangemax_variable ()
01791 {
01792     unsigned int i;
01793     #if DEBUG_INTERFACE
01794     fprintf (stderr, "window_rangemax_variable: start\n");
01795     #endif
01796     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01797     input->variable[i].rangemax = gtk_spin_button_get_value (window->
    spin_max);
01798     #if DEBUG_INTERFACE
01799     fprintf (stderr, "window_rangemax_variable: end\n");
01800     #endif
01801 }
01802
01803 void
01808 window_rangeminabs_variable ()
01809 {
01810     unsigned int i;
01811     #if DEBUG_INTERFACE
01812     fprintf (stderr, "window_rangeminabs_variable: start\n");
01813     #endif
01814     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01815     input->variable[i].rangeminabs
    = gtk_spin_button_get_value (window->spin_minabs);
01816     #if DEBUG_INTERFACE
01817     fprintf (stderr, "window_rangeminabs_variable: end\n");
01818     #endif
01819 }
01820
01821 void
01826 window_rangemaxabs_variable ()
01827 {
01828     unsigned int i;
01829     #if DEBUG_INTERFACE
01830     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01831     #endif
01832     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01833     input->variable[i].rangemaxabs
    = gtk_spin_button_get_value (window->spin_maxabs);
01834     #if DEBUG_INTERFACE
01835     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01836     #endif
01837 }
01838
01839 void
01846 window_step_variable ()
01847 {
01848     unsigned int i;
01849     #if DEBUG_INTERFACE
01850     fprintf (stderr, "window_step_variable: start\n");
01851     #endif
01852     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01853     input->variable[i].step = gtk_spin_button_get_value (window->
    spin_step);
01854     #if DEBUG_INTERFACE
01855     fprintf (stderr, "window_step_variable: end\n");
01856     #endif
01857 }
01858
01859 void
01864 window_update_variable ()
01865 {
01866     int i;
01867     #if DEBUG_INTERFACE
01868     fprintf (stderr, "window_update_variable: start\n");
01869     #endif
01870     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01871     if (i < 0)
01872         i = 0;
01873     switch (window_get_algorithm ())
01874     {
01875         case ALGORITHM_SWEEP:
01876             input->variable[i].nsweeps
    = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01877             #if DEBUG_INTERFACE
01878             fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
    input->variable[i].nsweeps);
01880

```

```

01881 #endif
01882     break;
01883     case ALGORITHM_GENETIC:
01884         input->variable[i].nbits
01885         = gtk_spin_button_get_value_as_int (window->spin_bits);
01886 #if DEBUG_INTERFACE
01887     fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01888             input->variable[i].nbits);
01889 #endif
01890 }
01891 #if DEBUG_INTERFACE
01892     fprintf (stderr, "window_update_variable: end\n");
01893 #endif
01894 }
01895
01903 int
01904 window_read (char *filename)
01905 {
01906     unsigned int i;
01907     char *buffer;
01908 #if DEBUG_INTERFACE
01909     fprintf (stderr, "window_read: start\n");
01910 #endif
01911
01912     // Reading new input file
01913     input_free ();
01914     if (!input_open (filename))
01915     {
01916 #if DEBUG_INTERFACE
01917         fprintf (stderr, "window_read: end\n");
01918 #endif
01919         return 0;
01920     }
01921
01922     // Setting GTK+ widgets data
01923     gtk_entry_set_text (window->entry_result, input->result);
01924     gtk_entry_set_text (window->entry_variables, input->
variables);
01925     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01926     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01927     g_free (buffer);
01928     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01929     if (input->evaluator)
01930     {
01931         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01932         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_evaluator), buffer);
01933         g_free (buffer);
01934     }
01935     gtk_toggle_button_set_active
01936     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01937     switch (input->algorithm)
01938     {
01939     case ALGORITHM_MONTE_CARLO:
01940         gtk_spin_button_set_value (window->spin_simulations,
(gdouble) input->nsimulations);
01941     case ALGORITHM_SWEEP:
01942         gtk_spin_button_set_value (window->spin_iterations,
(gdouble) input->niterations);
01943         gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
nbest);
01944         gtk_spin_button_set_value (window->spin_tolerance, input->
tolerance);
01945         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
(window->check_direction),
input->nsteps);
01946         if (input->nsteps)
01947         {
01948             gtk_toggle_button_set_active
01949             (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01950             gtk_spin_button_set_value (window->spin_steps,
(gdouble) input->nsteps);
01951             gtk_spin_button_set_value (window->spin_relaxation,
(gdouble) input->relaxation);
01952             switch (input->direction)
01953             {
01954             case DIRECTION_METHOD_RANDOM:
01955                 gtk_spin_button_set_value (window->spin_estimates,
(gdouble) input->nestimates);
01956             }
01957         }
01958     }

```



```

01968         break;
01969     default:
01970         gtk_spin_button_set_value (window->spin_population,
01971                                   (gdouble) input->nsimulations);
01972         gtk_spin_button_set_value (window->spin_generations,
01973                                   (gdouble) input->niterations);
01974         gtk_spin_button_set_value (window->spin_mutation,
01975                                   input->mutation_ratio);
01976         gtk_spin_button_set_value (window->spin_reproduction,
01977                                   input->reproduction_ratio);
01978         gtk_spin_button_set_value (window->spin_adaptation,
01979                                   input->adaptation_ratio);
01980     }
01981     gtk_toggle_button_set_active
01982     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01983     gtk_spin_button_set_value (window->spin_p, input->p);
01984     gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01985     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01986     g_signal_handler_block (window->button_experiment,
01987                             window->id_experiment_name);
01988     gtk_combo_box_text_remove_all (window->combo_experiment);
01989     for (i = 0; i < input->nexperiments; ++i)
01990         gtk_combo_box_text_append_text (window->combo_experiment,
01991                                         input->experiment[i].name);
01992     g_signal_handler_unblock
01993     (window->button_experiment, window->id_experiment_name);
01994     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01995     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01996     g_signal_handler_block (window->combo_variable, window->
id_variable);
01997     g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01998     gtk_combo_box_text_remove_all (window->combo_variable);
01999     for (i = 0; i < input->nvariables; ++i)
02000         gtk_combo_box_text_append_text (window->combo_variable,
02001                                         input->variable[i].name);
02002     g_signal_handler_unblock (window->entry_variable,
02003                             window->id_variable_label);
02004     g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02005     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02006     window_set_variable ();
02007     window_update ();
02008
02009     #if DEBUG_INTERFACE
02010     fprintf (stderr, "window_read: end\n");
02011     #endif
02012     return 1;
02013 }
02014
02015 void
02020 window_open ()
02021 {
02022     GtkFileChooserDialog *dlg;
02023     GtkFileFilter *filter;
02024     char *buffer, *directory, *name;
02025
02026     #if DEBUG_INTERFACE
02027     fprintf (stderr, "window_open: start\n");
02028     #endif
02029
02030     // Saving a backup of the current input file
02031     directory = g_strdup (input->directory);
02032     name = g_strdup (input->name);
02033
02034     // Opening dialog
02035     dlg = (GtkFileChooserDialog *)
02036         gtk_file_chooser_dialog_new (_("Open input file"),
02037                                     window->window,
02038                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02039                                     _("_Cancel"), GTK_RESPONSE_CANCEL,
02040                                     _("_OK"), GTK_RESPONSE_OK, NULL);
02041
02042     // Adding XML filter
02043     filter = (GtkFileFilter *) gtk_file_filter_new ();
02044     gtk_file_filter_set_name (filter, "XML");
02045     gtk_file_filter_add_pattern (filter, "*.xml");
02046     gtk_file_filter_add_pattern (filter, "*.XML");
02047     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02048
02049     // Adding JSON filter
02050     filter = (GtkFileFilter *) gtk_file_filter_new ();
02051     gtk_file_filter_set_name (filter, "JSON");
02052     gtk_file_filter_add_pattern (filter, "*.json");

```

```

02053 gtk_file_filter_add_pattern (filter, "*.JSON");
02054 gtk_file_filter_add_pattern (filter, "*.js");
02055 gtk_file_filter_add_pattern (filter, "*.JS");
02056 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02057
02058 // If OK saving
02059 while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02060 {
02061
02062     // Traying to open the input file
02063     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02064     if (!window_read (buffer))
02065     {
02066 #if DEBUG_INTERFACE
02067         fprintf (stderr, "window_open: error reading input file\n");
02068 #endif
02069         g_free (buffer);
02070
02071         // Reading backup file on error
02072         buffer = g_build_filename (directory, name, NULL);
02073         if (!input_open (buffer))
02074         {
02075
02076             // Closing on backup file reading error
02077 #if DEBUG_INTERFACE
02078             fprintf (stderr, "window_read: error reading backup file\n");
02079 #endif
02080             g_free (buffer);
02081             break;
02082         }
02083         g_free (buffer);
02084     }
02085     else
02086     {
02087         g_free (buffer);
02088         break;
02089     }
02090 }
02091
02092 // Freeing and closing
02093 g_free (name);
02094 g_free (directory);
02095 gtk_widget_destroy (GTK_WIDGET (dlg));
02096 #if DEBUG_INTERFACE
02097     fprintf (stderr, "window_open: end\n");
02098 #endif
02099 }
02100
02107 void
02108 window_new (GtkApplication * application)
02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[NDIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[NDIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135 #if DEBUG_INTERFACE
02136     fprintf (stderr, "window_new: start\n");
02137 #endif
02138
02139 // Creating the window
02140 window->window = main_window
02141     = (GtkWindow *) gtk_application_window_new (application);
02142
02143 // Finish when closing the window
02144 g_signal_connect_swapped (window->window, "delete-event",
02145     G_CALLBACK (g_application_quit),

```

```

02146             G_APPLICATION (application));
02147
02148 // Setting the window title
02149 gtk_window_set_title (window->window, "MPCOTool");
02150
02151 // Creating the open button
02152 window->button_open = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("document-open",
02154         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157 // Creating the save button
02158 window->button_save = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("document-save",
02160         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161 g_signal_connect (window->button_save, "clicked", (void (*)(
02162     window_save,
02163         NULL);
02164
02165 // Creating the run button
02166 window->button_run = (GtkToolButton *) gtk_tool_button_new
02167     (gtk_image_new_from_icon_name ("system-run",
02168         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171 // Creating the options button
02172 window->button_options = (GtkToolButton *) gtk_tool_button_new
02173     (gtk_image_new_from_icon_name ("preferences-system",
02174         GTK_ICON_SIZE_LARGE_TOOLBAR),
02175     _("Options"));
02176 g_signal_connect (window->button_options, "clicked", options_new, NULL);
02177
02178 // Creating the help button
02179 window->button_help = (GtkToolButton *) gtk_tool_button_new
02180     (gtk_image_new_from_icon_name ("help-browser",
02181         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02182 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02183
02184 // Creating the about button
02185 window->button_about = (GtkToolButton *) gtk_tool_button_new
02186     (gtk_image_new_from_icon_name ("help-about",
02187         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02188 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02189
02190 // Creating the exit button
02191 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02192     (gtk_image_new_from_icon_name ("application-exit",
02193         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02194 g_signal_connect_swapped (window->button_exit, "clicked",
02195     G_CALLBACK (g_application_quit),
02196     G_APPLICATION (application));
02197
02198 // Creating the buttons bar
02199 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02200 gtk_toolbar_insert
02201     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02202 gtk_toolbar_insert
02203     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02204 gtk_toolbar_insert
02205     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02206 gtk_toolbar_insert
02207     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02208 gtk_toolbar_insert
02209     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02210 gtk_toolbar_insert
02211     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02212 gtk_toolbar_insert
02213     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02214 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02215
02216 // Creating the simulator program label and entry
02217 window->label_simulator
02218     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02219 window->button_simulator = (GtkFileChooserButton *)
02220     gtk_file_chooser_button_new (_("Simulator program"),
02221     GTK_FILE_CHOOSER_ACTION_OPEN);
02222 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02223     _("Simulator program executable file"));
02224 gtk_widget_set_hexpand (GTK_WIDGET (window->button_simulator), TRUE);
02225
02226 // Creating the evaluator program label and entry
02227 window->check_evaluator = (GtkCheckButton *)
02228     gtk_check_button_new_with_mnemonic (_("Evaluator program"));
02229 g_signal_connect (window->check_evaluator, "toggled",
02230     window_update, NULL);
02231 window->button_evaluator = (GtkFileChooserButton *)
02232     gtk_file_chooser_button_new (_("Evaluator program"),

```

```

02231                                     GTK_FILE_CHOOSER_ACTION_OPEN);
02232 gtk_widget_set_tooltip_text
02233     (GTK_WIDGET (window->button_evaluator),
02234      _("Optional evaluator program executable file"));
02235
02236 // Creating the results files labels and entries
02237 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02238 window->entry_result = (GtkEntry *) gtk_entry_new ();
02239 gtk_widget_set_tooltip_text
02240     (GTK_WIDGET (window->entry_result), _("Best results file"));
02241 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02242 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02243 gtk_widget_set_tooltip_text
02244     (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02245
02246 // Creating the files grid and attaching widgets
02247 window->grid_files = (GtkGrid *) gtk_grid_new ();
02248 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02249                 0, 0, 1, 1);
02250 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02251                 1, 0, 1, 1);
02252 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02253                 0, 1, 1, 1);
02254 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02255                 1, 1, 1, 1);
02256 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02257                 0, 2, 1, 1);
02258 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02259                 1, 2, 1, 1);
02260 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02261                 0, 3, 1, 1);
02262 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02263                 1, 3, 1, 1);
02264
02265 // Creating the algorithm properties
02266 window->label_simulations = (GtkLabel *) gtk_label_new
02267     (_("Simulations number"));
02268 window->spin_simulations
02269     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270 gtk_widget_set_tooltip_text
02271     (GTK_WIDGET (window->spin_simulations),
02272      _("Number of simulations to perform for each iteration"));
02273 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274 window->label_iterations = (GtkLabel *)
02275     gtk_label_new (_("Iterations number"));
02276 window->spin_iterations
02277     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278 gtk_widget_set_tooltip_text
02279     (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280 g_signal_connect
02281     (window->spin_iterations, "value-changed", window_update, NULL);
02282 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284 window->spin_tolerance =
02285     (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286 gtk_widget_set_tooltip_text
02287     (GTK_WIDGET (window->spin_tolerance),
02288      _("Tolerance to set the variable interval on the next iteration"));
02289 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290 window->spin_bests
02291     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292 gtk_widget_set_tooltip_text
02293     (GTK_WIDGET (window->spin_bests),
02294      _("Number of best simulations used to set the variable interval "
02295        "on the next iteration"));
02296 window->label_population
02297     = (GtkLabel *) gtk_label_new (_("Population number"));
02298 window->spin_population
02299     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02300 gtk_widget_set_tooltip_text
02301     (GTK_WIDGET (window->spin_population),
02302      _("Number of population for the genetic algorithm"));
02303 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02304 window->label_generations
02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306 window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308 gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),

```

```

02310     _("Number of generations for the genetic algorithm"));
02311 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312 window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314 gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317 window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319 window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321 gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324 window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326 window->spin_adaptation
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328 gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332 window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335 gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338 window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                   GTK_WIDGET (window->spin_threshold));
02342 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                          GTK_ALIGN_FILL);
02345
02346 // Creating the direction search method properties
02347 window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("Direction search method"));
02349 g_signal_connect (window->check_direction, "clicked",
02350                  window_update, NULL);
02351 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02352 window->button_direction[0] = (GtkRadioButton *)
02353     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02354 gtk_grid_attach (window->grid_direction,
02355                 GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02356 g_signal_connect (window->button_direction[0], "clicked",
02357                  window_update,
02358                  NULL);
02359 for (i = 0; ++i < NDIRECTIONS;)
02360 {
02361     window->button_direction[i] = (GtkRadioButton *)
02362         gtk_radio_button_new_with_mnemonic
02363             (gtk_radio_button_get_group (window->button_direction[0]),
02364              label_direction[i]);
02365     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02366                                 tip_direction[i]);
02367     gtk_grid_attach (window->grid_direction,
02368                     GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02369     g_signal_connect (window->button_direction[i], "clicked",
02370                      window_update, NULL);
02371 }
02372 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02373 window->spin_steps = (GtkSpinButton *)
02374     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02375 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02376 window->label_estimates
02377     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02378 window->spin_estimates = (GtkSpinButton *)
02379     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02380 window->label_relaxation
02381     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02382 window->spin_relaxation = (GtkSpinButton *)
02383     gtk_spin_button_new_with_range (0., 2., 0.001);
02384 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02385 label_steps),
02386                 0, NDIRECTIONS, 1, 1);
02387 gtk_grid_attach (window->grid_direction, GTK_WIDGET (window->
02388 spin_steps),
02389                 1, NDIRECTIONS, 1, 1);
02390 gtk_grid_attach (window->grid_direction,
02391                 GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,
02392                 1, 1);
02393 gtk_grid_attach (window->grid_direction,
02394                 GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02395                 1);
02396 gtk_grid_attach (window->grid_direction,

```

```

02393         GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394         1, 1);
02395     gtk_grid_attach (window->grid_direction,
02396         GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397         1, 1);
02398
02399     // Creating the array of algorithms
02400     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401     window->button_algorithm[0] = (GtkRadioButton *)
02402         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404         tip_algorithm[0]);
02405     gtk_grid_attach (window->grid_algorithm,
02406         GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407     g_signal_connect (window->button_algorithm[0], "clicked",
02408         window_set_algorithm, NULL);
02409     for (i = 0; ++i < NALGORITHMS;)
02410     {
02411         window->button_algorithm[i] = (GtkRadioButton *)
02412             gtk_radio_button_new_with_mnemonic
02413             (gtk_radio_button_get_group (window->button_algorithm[0]),
02414             label_algorithm[i]);
02415         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416             tip_algorithm[i]);
02417         gtk_grid_attach (window->grid_algorithm,
02418             GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419         g_signal_connect (window->button_algorithm[i], "clicked",
02420             window_set_algorithm, NULL);
02421     }
02422     gtk_grid_attach (window->grid_algorithm,
02423         GTK_WIDGET (window->label_simulations), 0,
02424         NALGORITHMS, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm,
02426         GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427         1);
02428     gtk_grid_attach (window->grid_algorithm,
02429         GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430         1, 1);
02431     gtk_grid_attach (window->grid_algorithm,
02432         GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433         1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435         GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436         1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438         GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439         1);
02440     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02441         label_bests),
02442         0, NALGORITHMS + 3, 1, 1);
02443     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02444         spin_bests), 1,
02445         NALGORITHMS + 3, 1, 1);
02446     gtk_grid_attach (window->grid_algorithm,
02447         GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02448         1, 1);
02449     gtk_grid_attach (window->grid_algorithm,
02450         GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02451         1, 1);
02452     gtk_grid_attach (window->grid_algorithm,
02453         GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02454         1, 1);
02455     gtk_grid_attach (window->grid_algorithm,
02456         GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02457         1, 1);
02458     gtk_grid_attach (window->grid_algorithm,
02459         GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02460         1);
02461     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
02462         spin_mutation),
02463         1, NALGORITHMS + 6, 1, 1);
02464     gtk_grid_attach (window->grid_algorithm,
02465         GTK_WIDGET (window->label_reproduction), 0,
02466         NALGORITHMS + 7, 1, 1);
02467     gtk_grid_attach (window->grid_algorithm,
02468         GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02469         1, 1);
02470     gtk_grid_attach (window->grid_algorithm,
02471         GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02472         1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474         GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,
02475         1, 1);
02476     gtk_grid_attach (window->grid_algorithm,
02477         GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02478         2, 1);
02479     gtk_grid_attach (window->grid_algorithm,

```

```

02477         GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478         2, 1);
02479     gtk_grid_attach (window->grid_algorithm,
02480         GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481         1, 1);
02482     gtk_grid_attach (window->grid_algorithm,
02483         GTK_WIDGET (window->scrolled_threshold), 1,
02484         NALGORITHMS + 11, 1, 1);
02485     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487         GTK_WIDGET (window->grid_algorithm));
02488
02489     // Creating the variable widgets
02490     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491     gtk_widget_set_tooltip_text
02492         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493     window->id_variable = g_signal_connect
02494         (window->combo_variable, "changed", window_set_variable, NULL);
02495     window->button_add_variable
02496         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497             GTK_ICON_SIZE_BUTTON);
02498     g_signal_connect
02499         (window->button_add_variable, "clicked",
02500         window_add_variable, NULL);
02501     gtk_widget_set_tooltip_text
02502         (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02503     window->button_remove_variable
02504         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02505             GTK_ICON_SIZE_BUTTON);
02506     g_signal_connect
02507         (window->button_remove_variable, "clicked",
02508         window_remove_variable, NULL);
02509     gtk_widget_set_tooltip_text
02510         (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02511     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02512     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02513     gtk_widget_set_tooltip_text
02514         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02515     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02516     window->id_variable_label = g_signal_connect
02517         (window->entry_variable, "changed", window_label_variable, NULL);
02518     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02519     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02520         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02521     gtk_widget_set_tooltip_text
02522         (GTK_WIDGET (window->spin_min),
02523         _("Minimum initial value of the variable"));
02524     window->scrolled_min
02525         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02526     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02527         GTK_WIDGET (window->spin_min));
02528     g_signal_connect (window->spin_min, "value-changed",
02529         window_rangemin_variable, NULL);
02530     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02531     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02532         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02533     gtk_widget_set_tooltip_text
02534         (GTK_WIDGET (window->spin_max),
02535         _("Maximum initial value of the variable"));
02536     window->scrolled_max
02537         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02538     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02539         GTK_WIDGET (window->spin_max));
02540     g_signal_connect (window->spin_max, "value-changed",
02541         window_rangemax_variable, NULL);
02542     window->check_minabs = (GtkCheckButton *)
02543         gtk_check_button_new_with_mnemonic (_("_Absolute minimum"));
02544     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02545     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02546         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02547     gtk_widget_set_tooltip_text
02548         (GTK_WIDGET (window->spin_minabs),
02549         _("Minimum allowed value of the variable"));
02550     window->scrolled_minabs
02551         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02552     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02553         GTK_WIDGET (window->spin_minabs));
02554     g_signal_connect (window->spin_minabs, "value-changed",
02555         window_rangeminabs_variable, NULL);
02556     window->check_maxabs = (GtkCheckButton *)
02557         gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02558     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02559     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02560         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02561     gtk_widget_set_tooltip_text
02562         (GTK_WIDGET (window->spin_maxabs),
02563         _("Maximum allowed value of the variable"));

```



```

02562 window->scrolled_maxabs
02563     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564 gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565     GTK_WIDGET (window->spin_maxabs));
02566 g_signal_connect (window->spin_maxabs, "value-changed",
02567     window_rangemaxabs_variable, NULL);
02568 window->label_precision
02569     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570 window->spin_precision = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572 gtk_widget_set_tooltip_text
02573     (GTK_WIDGET (window->spin_precision),
02574     _("Number of precision floating point digits\n"
02575     "0 is for integer numbers"));
02576 g_signal_connect (window->spin_precision, "value-changed",
02577     window_precision_variable, NULL);
02578 window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579 window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582     _("Number of steps sweeping the variable"));
02583 g_signal_connect (window->spin_sweeps, "value-changed",
02584     window_update_variable, NULL);
02585 window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586 window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588 gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590     _("Number of bits to encode the variable"));
02591 g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL);
02593 window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594 window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596 gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598     _("Initial step size for the direction search method"));
02599 window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602     GTK_WIDGET (window->spin_step));
02603 g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606 gtk_grid_attach (window->grid_variable,
02607     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608 gtk_grid_attach (window->grid_variable,
02609     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610 gtk_grid_attach (window->grid_variable,
02611     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612 gtk_grid_attach (window->grid_variable,
02613     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614 gtk_grid_attach (window->grid_variable,
02615     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616 gtk_grid_attach (window->grid_variable,
02617     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618 gtk_grid_attach (window->grid_variable,
02619     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620 gtk_grid_attach (window->grid_variable,
02621     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622 gtk_grid_attach (window->grid_variable,
02623     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624 gtk_grid_attach (window->grid_variable,
02625     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626 gtk_grid_attach (window->grid_variable,
02627     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628 gtk_grid_attach (window->grid_variable,
02629     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630 gtk_grid_attach (window->grid_variable,
02631     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632 gtk_grid_attach (window->grid_variable,
02633     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634 gtk_grid_attach (window->grid_variable,
02635     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636 gtk_grid_attach (window->grid_variable,
02637     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638 gtk_grid_attach (window->grid_variable,
02639     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02640 gtk_grid_attach (window->grid_variable,
02641     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));

```



```

02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                     GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                             _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657     (window->combo_experiment, "changed", window_set_experiment, NULL)
;
02658 window->button_add_experiment
02659     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660                                                    GTK_ICON_SIZE_BUTTON);
02661 g_signal_connect
02662     (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02663 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664                             _("Add experiment"));
02665 window->button_remove_experiment
02666     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667                                                    GTK_ICON_SIZE_BUTTON);
02668 g_signal_connect (window->button_remove_experiment, "clicked",
02669                  window_remove_experiment, NULL);
02670 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02671                             _("Remove experiment"));
02672 window->label_experiment
02673     = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02674 window->button_experiment = (GtkFileChooserButton *)
02675     gtk_file_chooser_button_new (_("Experimental data file"),
02676                                 GTK_FILE_CHOOSER_ACTION_OPEN);
02677 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02678                             _("Experimental data file"));
02679 window->id_experiment_name
02680     = g_signal_connect (window->button_experiment, "selection-changed",
02681                        window_name_experiment, NULL);
02682 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);
02683 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02684 window->spin_weight
02685     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02686 gtk_widget_set_tooltip_text
02687     (GTK_WIDGET (window->spin_weight),
02688      _("Weight factor to build the objective function"));
02689 g_signal_connect
02690     (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
02691 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02692 gtk_grid_attach (window->grid_experiment,
02693                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02694 gtk_grid_attach (window->grid_experiment,
02695                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02696 gtk_grid_attach (window->grid_experiment,
02697                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02698 gtk_grid_attach (window->grid_experiment,
02699                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700 gtk_grid_attach (window->grid_experiment,
02701                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702 gtk_grid_attach (window->grid_experiment,
02703                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704 gtk_grid_attach (window->grid_experiment,
02705                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706 for (i = 0; i < MAX_NINPUTS; ++i)
02707 {
02708     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709     window->check_template[i] = (GtkCheckButton *)
02710     gtk_check_button_new_with_label (buffer3);
02711     window->id_template[i]
02712     = g_signal_connect (window->check_template[i], "toggled",
02713                        window_inputs_experiment, NULL);
02714     gtk_grid_attach (window->grid_experiment,
02715                     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716                     1);
02717     window->button_template[i] =
02718     (GtkFileChooserButton *)
02719     gtk_file_chooser_button_new (_("Input template"),
02720                                 GTK_FILE_CHOOSER_ACTION_OPEN);
02721     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),
02722                                 _("Experimental input template file"));
02723     window->id_input[i] =
02724     g_signal_connect_swapped (window->button_template[i],
02725                              "selection-changed",
02726                              (void (*)(void *)) window_template_experiment,
02727                              (void *) (size_t) i);
02728     gtk_grid_attach (window->grid_experiment,
02729                     GTK_WIDGET (window->button_template[i]),
02730                     1, 3 + i, 3, 1);
02731 }
02732 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));

```

```

02733 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734                     GTK_WIDGET (window->grid_experiment));
02735
02736 // Creating the error norm widgets
02737 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740                     GTK_WIDGET (window->grid_norm));
02741 window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02744                             tip_norm[0]);
02745 gtk_grid_attach (window->grid_norm,
02746                 GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02748 for (i = 0; ++i < NNORMS;)
02749 {
02750     window->button_norm[i] = (GtkRadioButton *)
02751         gtk_radio_button_new_with_mnemonic
02752             (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02753     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02754                                 tip_norm[i]);
02755     gtk_grid_attach (window->grid_norm,
02756                     GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02757     g_signal_connect (window->button_norm[i], "clicked",
02758 window_update,
02759                     NULL);
02760 }
02761 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02762 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1,
02763                 1);
02764 window->spin_p =
02765     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02766                                                       G_MAXDOUBLE, 0.01);
02767 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02768                             _("P parameter for the P error norm"));
02769 window->scrolled_p =
02770     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02771 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02772                     GTK_WIDGET (window->spin_p));
02773 gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02774 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02775 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02776                 1, 2, 1, 2);
02777
02778 // Creating the grid and attaching the widgets to the grid
02779 window->grid = (GtkGrid *) gtk_grid_new ();
02780 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02781                 1);
02782 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02783 gtk_grid_attach (window->grid,
02784                 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02785 gtk_grid_attach (window->grid,
02786                 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02787 gtk_grid_attach (window->grid,
02788                 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02789 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02790 gtk_container_add (GTK_CONTAINER (window->window),
02791                     GTK_WIDGET (window->grid));
02792
02793 // Setting the window logo
02794 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02795 gtk_window_set_icon (window->window, window->logo);
02796
02797 // Showing the window
02798 gtk_widget_show_all (GTK_WIDGET (window->window));
02799
02800 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02801 #if GTK_MINOR_VERSION >= 16
02802 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02803 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02804 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02805 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02806 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02807 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02808 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02809                             40);
02810 #endif
02811
02812 // Reading initial example
02813 input_new ();
02814 buffer2 = g_get_current_dir ();
02815 buffer =
02816     g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02817 g_free (buffer2);
02818 window_read (buffer);
02819 g_free (buffer);

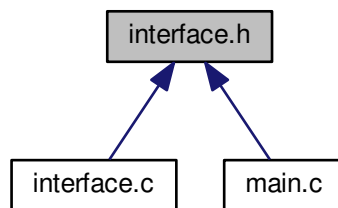
```

```
02819
02820 #if DEBUG_INTERFACE
02821     fprintf (stderr, "window_new: start\n");
02822 #endif
02823 }
```

## 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

- unsigned int [gtk\\_array\\_get\\_active](#) (GtkRadioButton \*array[], unsigned int n)  
*Function to get the active GtkRadioButton.*
- void [input\\_save](#) (char \*filename)  
*Function to save the input file.*
- void [options\\_new](#) ()  
*Function to open the options dialog.*
- void [running\\_new](#) ()  
*Function to open the running dialog.*

- unsigned int [window\\_get\\_algorithm](#) ()  
*Function to get the stochastic algorithm number.*
- unsigned int [window\\_get\\_direction](#) ()  
*Function to get the direction search method number.*
- unsigned int [window\\_get\\_norm](#) ()  
*Function to get the norm method number.*
- void [window\\_save\\_direction](#) ()  
*Function to save the direction search method data in the input file.*
- int [window\\_save](#) ()  
*Function to save the input file.*
- void [window\\_run](#) ()  
*Function to run a optimization.*
- void [window\\_help](#) ()  
*Function to show a help dialog.*
- void [window\\_update\\_direction](#) ()  
*Function to update direction search method widgets view in the main window.*
- void [window\\_update](#) ()  
*Function to update the main window view.*
- void [window\\_set\\_algorithm](#) ()  
*Function to avoid memory errors changing the algorithm.*
- void [window\\_set\\_experiment](#) ()  
*Function to set the experiment data in the main window.*
- void [window\\_remove\\_experiment](#) ()  
*Function to remove an experiment in the main window.*
- void [window\\_add\\_experiment](#) ()  
*Function to add an experiment in the main window.*
- void [window\\_name\\_experiment](#) ()  
*Function to set the experiment name in the main window.*
- void [window\\_weight\\_experiment](#) ()  
*Function to update the experiment weight in the main window.*
- void [window\\_inputs\\_experiment](#) ()  
*Function to update the experiment input templates number in the main window.*
- void [window\\_template\\_experiment](#) (void \*data)  
*Function to update the experiment i-th input template in the main window.*
- void [window\\_set\\_variable](#) ()  
*Function to set the variable data in the main window.*
- void [window\\_remove\\_variable](#) ()  
*Function to remove a variable in the main window.*
- void [window\\_add\\_variable](#) ()  
*Function to add a variable in the main window.*
- void [window\\_label\\_variable](#) ()  
*Function to set the variable label in the main window.*
- void [window\\_precision\\_variable](#) ()  
*Function to update the variable precision in the main window.*
- void [window\\_rangemin\\_variable](#) ()  
*Function to update the variable rangemin in the main window.*
- void [window\\_rangemax\\_variable](#) ()  
*Function to update the variable rangemax in the main window.*
- void [window\\_rangeminabs\\_variable](#) ()  
*Function to update the variable rangeminabs in the main window.*
- void [window\\_rangemaxabs\\_variable](#) ()

- *Function to update the variable rangemaxabs in the main window.*  
void [window\\_update\\_variable](#) ()
- *Function to update the variable data in the main window.*  
int [window\\_read](#) (char \*filename)
- *Function to read the input data of a file.*  
void [window\\_open](#) ()
- *Function to open the input data.*  
void [window\\_new](#) (GtkApplication \*application)
- *Function to open the main window.*

## Variables

- const char \* [logo](#) []  
*Logo pixmap.*
- [Options](#) [options](#) [1]  
*Options struct to define the options dialog.*
- [Running](#) [running](#) [1]  
*Running struct to define the running dialog.*
- [Window](#) [window](#) [1]  
*Window struct to define the main interface window.*

### 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 (
    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 GtkRadioButton.

Definition at line 567 of file [utils.c](#).

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

**5.13.2.2 input\_save()**

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

Function to save the input file.

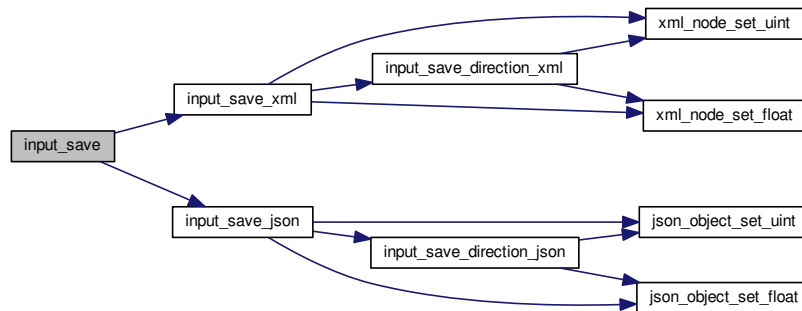
**Parameters**

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

Definition at line 579 of file [interface.c](#).

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

Here is the call graph for this function:



### 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 732 of file [interface.c](#).

```

00733 {
00734     unsigned int i;
00735     #if DEBUG_INTERFACE
00736         fprintf (stderr, "window_get_algorithm: start\n");
00737     #endif
00738     i = gtk_array_get_active (window->button_algorithm,
00739                             NALGORITHMS);
00739     #if DEBUG_INTERFACE
00740         fprintf (stderr, "window_get_algorithm: %u\n", i);
00741         fprintf (stderr, "window_get_algorithm: end\n");
00742     #endif
00743     return i;
00744 }

```

Here is the call graph for this function:



#### 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 752 of file [interface.c](#).

```
00753 {
00754     unsigned int i;
00755     #if DEBUG_INTERFACE
00756     fprintf (stderr, "window_get_direction: start\n");
00757     #endif
00758     i = gtk_array_get_active (window->button_direction,
00759                             NDIRECTIONS);
00759     #if DEBUG_INTERFACE
00760     fprintf (stderr, "window_get_direction: %u\n", i);
00761     fprintf (stderr, "window_get_direction: end\n");
00762     #endif
00763     return i;
00764 }
```

Here is the call graph for this function:



#### 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 772 of file [interface.c](#).

```
00773 {
00774     unsigned int i;
00775     #if DEBUG_INTERFACE
00776     fprintf (stderr, "window_get_norm: start\n");
00777     #endif
00778     i = gtk_array_get_active (window->button_norm,
00779                             NNORMS);
00779     #if DEBUG_INTERFACE
00780     fprintf (stderr, "window_get_norm: %u\n", i);
00781     fprintf (stderr, "window_get_norm: end\n");
00782     #endif
00783     return i;
00784 }
```



Here is the call graph for this function:



### 5.13.2.6 window\_new()

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

Function to open the main window.

#### Parameters

<i>application</i>	GtkApplication struct.
--------------------	------------------------

Definition at line 2108 of file [interface.c](#).

```

02109 {
02110     unsigned int i;
02111     char *buffer, *buffer2, buffer3[64];
02112     char *label_algorithm[NALGORITHMS] = {
02113         "_Monte-Carlo", _("_Sweep"), _("_Genetic")
02114     };
02115     char *tip_algorithm[NALGORITHMS] = {
02116         _("Monte-Carlo brute force algorithm"),
02117         _("Sweep brute force algorithm"),
02118         _("Genetic algorithm")
02119     };
02120     char *label_direction[NDIRECTIONS] = {
02121         _("_Coordinates descent"), _("_Random")
02122     };
02123     char *tip_direction[NDIRECTIONS] = {
02124         _("Coordinates direction estimate method"),
02125         _("Random direction estimate method")
02126     };
02127     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02128     char *tip_norm[NNORMS] = {
02129         _("Euclidean error norm (L2)"),
02130         _("Maximum error norm (L)"),
02131         _("P error norm (Lp)"),
02132         _("Taxicab error norm (L1)")
02133     };
02134
02135     #if DEBUG_INTERFACE
02136         fprintf (stderr, "window_new: start\n");
02137     #endif
02138
02139     // Creating the window
02140     window->window = main_window
02141         = (GtkWindow *) gtk_application_window_new (application);
02142
02143     // Finish when closing the window
02144     g_signal_connect_swapped (window->window, "delete-event",
02145                               G_CALLBACK (g_application_quit),
02146                               G_APPLICATION (application));
02147
02148     // Setting the window title
02149     gtk_window_set_title (window->window, "MPCOTool");
02150

```

```

02151 // Creating the open button
02152 window->button_open = (GtkToolButton *) gtk_tool_button_new
02153     (gtk_image_new_from_icon_name ("document-open",
02154         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Open"));
02155 g_signal_connect (window->button_open, "clicked", window_open, NULL);
02156
02157 // Creating the save button
02158 window->button_save = (GtkToolButton *) gtk_tool_button_new
02159     (gtk_image_new_from_icon_name ("document-save",
02160         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Save"));
02161 g_signal_connect (window->button_save, "clicked", (void (*)(
02162     window_save,
02163         NULL));
02164
02165 // Creating the run button
02166 window->button_run = (GtkToolButton *) gtk_tool_button_new
02167     (gtk_image_new_from_icon_name ("system-run",
02168         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Run"));
02169 g_signal_connect (window->button_run, "clicked", window_run, NULL);
02170
02171 // Creating the options button
02172 window->button_options = (GtkToolButton *) gtk_tool_button_new
02173     (gtk_image_new_from_icon_name ("preferences-system",
02174         GTK_ICON_SIZE_LARGE_TOOLBAR),
02175     _("Options"));
02176 g_signal_connect (window->button_options, "clicked",
02177     options_new, NULL);
02178
02179 // Creating the help button
02180 window->button_help = (GtkToolButton *) gtk_tool_button_new
02181     (gtk_image_new_from_icon_name ("help-browser",
02182         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Help"));
02183 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02184
02185 // Creating the about button
02186 window->button_about = (GtkToolButton *) gtk_tool_button_new
02187     (gtk_image_new_from_icon_name ("help-about",
02188         GTK_ICON_SIZE_LARGE_TOOLBAR), _("About"));
02189 g_signal_connect (window->button_about, "clicked",
02190     window_about, NULL);
02191
02192 // Creating the exit button
02193 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02194     (gtk_image_new_from_icon_name ("application-exit",
02195         GTK_ICON_SIZE_LARGE_TOOLBAR), _("Exit"));
02196 g_signal_connect_swapped (window->button_exit, "clicked",
02197     G_CALLBACK (g_application_quit),
02198     G_APPLICATION (application));
02199
02200 // Creating the buttons bar
02201 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02202 gtk_toolbar_insert
02203     (window->bar_buttons, GTK_TOOL_ITEM (window->
02204     button_open), 0);
02205 gtk_toolbar_insert
02206     (window->bar_buttons, GTK_TOOL_ITEM (window->
02207     button_save), 1);
02208 gtk_toolbar_insert
02209     (window->bar_buttons, GTK_TOOL_ITEM (window->
02210     button_run), 2);
02211 gtk_toolbar_insert
02212     (window->bar_buttons, GTK_TOOL_ITEM (window->
02213     button_options), 3);
02214 gtk_toolbar_insert
02215     (window->bar_buttons, GTK_TOOL_ITEM (window->
02216     button_help), 4);
02217 gtk_toolbar_insert
02218     (window->bar_buttons, GTK_TOOL_ITEM (window->
02219     button_about), 5);
02220 gtk_toolbar_insert
02221     (window->bar_buttons, GTK_TOOL_ITEM (window->
02222     button_exit), 6);
02223 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02224
02225 // Creating the simulator program label and entry
02226 window->label_simulator
02227     = (GtkLabel *) gtk_label_new (_("Simulator program"));
02228 window->button_simulator = (GtkFileChooserButton *)
02229     gtk_file_chooser_button_new (_("Simulator program"),
02230     GTK_FILE_CHOOSER_ACTION_OPEN);
02231 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02232     _("Simulator program executable file"));
02233 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02234
02235 // Creating the evaluator program label and entry
02236 window->check_evaluator = (GtkCheckButton *)
02237     gtk_check_button_new_with_mnemonic (_("_Evaluator program"));

```

```

02228 g_signal_connect (window->check_evaluator, "toggled",
window_update, NULL);
02229 window->button_evaluator = (GtkFileChooserButton *)
02230 gtk_file_chooser_button_new (_("Evaluator program"),
02231 GTK_FILE_CHOOSER_ACTION_OPEN);
02232 gtk_widget_set_tooltip_text
02233 (GTK_WIDGET (window->button_evaluator),
02234 _("Optional evaluator program executable file"));
02235
02236 // Creating the results files labels and entries
02237 window->label_result = (GtkLabel *) gtk_label_new (_("Result file"));
02238 window->entry_result = (GtkEntry *) gtk_entry_new ();
02239 gtk_widget_set_tooltip_text
02240 (GTK_WIDGET (window->entry_result), _("Best results file"));
02241 window->label_variables = (GtkLabel *) gtk_label_new (_("Variables file"));
02242 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02243 gtk_widget_set_tooltip_text
02244 (GTK_WIDGET (window->entry_variables), _("All simulated results file"));
02245
02246 // Creating the files grid and attaching widgets
02247 window->grid_files = (GtkGrid *) gtk_grid_new ();
02248 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_simulator),
02249 0, 0, 1, 1);
02250 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_simulator),
02251 1, 0, 1, 1);
02252 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
check_evaluator),
02253 0, 1, 1, 1);
02254 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
button_evaluator),
02255 1, 1, 1, 1);
02256 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_result),
02257 0, 2, 1, 1);
02258 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_result),
02259 1, 2, 1, 1);
02260 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02261 0, 3, 1, 1);
02262 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02263 1, 3, 1, 1);
02264
02265 // Creating the algorithm properties
02266 window->label_simulations = (GtkLabel *) gtk_label_new
02267 (_("Simulations number"));
02268 window->spin_simulations
02269 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02270 gtk_widget_set_tooltip_text
02271 (GTK_WIDGET (window->spin_simulations),
02272 _("Number of simulations to perform for each iteration"));
02273 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02274 window->label_iterations = (GtkLabel *)
02275 gtk_label_new (_("Iterations number"));
02276 window->spin_iterations
02277 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02278 gtk_widget_set_tooltip_text
02279 (GTK_WIDGET (window->spin_iterations), _("Number of iterations"));
02280 g_signal_connect
02281 (window->spin_iterations, "value-changed",
window_update, NULL);
02282 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02283 window->label_tolerance = (GtkLabel *) gtk_label_new (_("Tolerance"));
02284 window->spin_tolerance =
02285 (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02286 gtk_widget_set_tooltip_text
02287 (GTK_WIDGET (window->spin_tolerance),
02288 _("Tolerance to set the variable interval on the next iteration"));
02289 window->label_bests = (GtkLabel *) gtk_label_new (_("Bests number"));
02290 window->spin_bests
02291 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02292 gtk_widget_set_tooltip_text
02293 (GTK_WIDGET (window->spin_bests),
02294 _("Number of best simulations used to set the variable interval "
"on the next iteration"));
02295 window->label_population
02296 = (GtkLabel *) gtk_label_new (_("Population number"));
02297 window->spin_population
02298 = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02299 gtk_widget_set_tooltip_text
02300 (GTK_WIDGET (window->spin_population),
02301 _("Number of population for the genetic algorithm"));
02302 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_population), TRUE);
02303 window->label_generations
02304

```

```

02305     = (GtkLabel *) gtk_label_new (_("Generations number"));
02306 window->spin_generations
02307     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02308 gtk_widget_set_tooltip_text
02309     (GTK_WIDGET (window->spin_generations),
02310      _("Number of generations for the genetic algorithm"));
02311 window->label_mutation = (GtkLabel *) gtk_label_new (_("Mutation ratio"));
02312 window->spin_mutation
02313     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02314 gtk_widget_set_tooltip_text
02315     (GTK_WIDGET (window->spin_mutation),
02316      _("Ratio of mutation for the genetic algorithm"));
02317 window->label_reproduction
02318     = (GtkLabel *) gtk_label_new (_("Reproduction ratio"));
02319 window->spin_reproduction
02320     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02321 gtk_widget_set_tooltip_text
02322     (GTK_WIDGET (window->spin_reproduction),
02323      _("Ratio of reproduction for the genetic algorithm"));
02324 window->label_adaptation
02325     = (GtkLabel *) gtk_label_new (_("Adaptation ratio"));
02326 window->spin_adaptation
02327     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02328 gtk_widget_set_tooltip_text
02329     (GTK_WIDGET (window->spin_adaptation),
02330      _("Ratio of adaptation for the genetic algorithm"));
02331 window->label_threshold = (GtkLabel *) gtk_label_new (_("Threshold"));
02332 window->spin_threshold = (GtkSpinButton *)
02333     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE,
02334                                     precision[DEFAULT_PRECISION]);
02335 gtk_widget_set_tooltip_text
02336     (GTK_WIDGET (window->spin_threshold),
02337      _("Threshold in the objective function to finish the simulations"));
02338 window->scrolled_threshold =
02339     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02340 gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02341                  GTK_WIDGET (window->spin_threshold));
02342 // gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02343 // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),
02344 //                          GTK_ALIGN_FILL);
02345
02346 // Creating the direction search method properties
02347 window->check_direction = (GtkCheckButton *)
02348     gtk_check_button_new_with_mnemonic (_("Direction search method"));
02349 g_signal_connect (window->check_direction, "clicked",
02350                 window_update, NULL);
02351 window->grid_direction = (GtkGrid *) gtk_grid_new ();
02352 window->button_direction[0] = (GtkRadioButton *)
02353     gtk_radio_button_new_with_mnemonic (NULL, label_direction[0]);
02354 gtk_grid_attach (window->grid_direction,
02355                 GTK_WIDGET (window->button_direction[0]), 0, 0, 1, 1);
02356 g_signal_connect (window->button_direction[0], "clicked",
02357                 window_update,
02358                 NULL);
02359 for (i = 0; ++i < NDIRECTIONS;)
02360 {
02361     window->button_direction[i] = (GtkRadioButton *)
02362         gtk_radio_button_new_with_mnemonic
02363             (gtk_radio_button_get_group (window->button_direction[0]),
02364              label_direction[i]);
02365     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_direction[i]),
02366                                 tip_direction[i]);
02367     gtk_grid_attach (window->grid_direction,
02368                     GTK_WIDGET (window->button_direction[i]), 0, i, 1, 1);
02369     g_signal_connect (window->button_direction[i], "clicked",
02370                     window_update, NULL);
02371 }
02372 window->label_steps = (GtkLabel *) gtk_label_new (_("Steps number"));
02373 window->spin_steps = (GtkSpinButton *)
02374     gtk_spin_button_new_with_range (1., 1.e12, 1.);
02375 gtk_widget_set_hexpand (GTK_WIDGET (window->spin_steps), TRUE);
02376 window->label_estimates
02377     = (GtkLabel *) gtk_label_new (_("Direction estimates number"));
02378 window->spin_estimates = (GtkSpinButton *)
02379     gtk_spin_button_new_with_range (1., 1.e3, 1.);
02380 window->label_relaxation
02381     = (GtkLabel *) gtk_label_new (_("Relaxation parameter"));
02382 window->spin_relaxation = (GtkSpinButton *)
02383     gtk_spin_button_new_with_range (0., 2., 0.001);
02384 gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02385     window->label_steps),
02386     0, NDIRECTIONS, 1, 1);
02387 gtk_grid_attach (window->grid_direction, GTK_WIDGET (
02388     window->spin_steps),
02389     1, NDIRECTIONS, 1, 1);
02390 gtk_grid_attach (window->grid_direction,
02391                 GTK_WIDGET (window->label_estimates), 0, NDIRECTIONS + 1,

```

```

02388         1, 1);
02389     gtk_grid_attach (window->grid_direction,
02390         GTK_WIDGET (window->spin_estimates), 1, NDIRECTIONS + 1, 1,
02391         1);
02392     gtk_grid_attach (window->grid_direction,
02393         GTK_WIDGET (window->label_relaxation), 0, NDIRECTIONS + 2,
02394         1, 1);
02395     gtk_grid_attach (window->grid_direction,
02396         GTK_WIDGET (window->spin_relaxation), 1, NDIRECTIONS + 2,
02397         1, 1);
02398
02399     // Creating the array of algorithms
02400     window->grid_algorithm = (GtkGrid *) gtk_grid_new ();
02401     window->button_algorithm[0] = (GtkRadioButton *)
02402         gtk_radio_button_new_with_mnemonic (NULL, label_algorithm[0]);
02403     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[0]),
02404         tip_algorithm[0]);
02405     gtk_grid_attach (window->grid_algorithm,
02406         GTK_WIDGET (window->button_algorithm[0]), 0, 0, 1, 1);
02407     g_signal_connect (window->button_algorithm[0], "clicked",
02408         window_set_algorithm, NULL);
02409     for (i = 0; ++i < NALGORITHMS;)
02410     {
02411         window->button_algorithm[i] = (GtkRadioButton *)
02412             gtk_radio_button_new_with_mnemonic
02413             (gtk_radio_button_get_group (window->button_algorithm[0]),
02414             label_algorithm[i]);
02415         gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_algorithm[i]),
02416             tip_algorithm[i]);
02417         gtk_grid_attach (window->grid_algorithm,
02418             GTK_WIDGET (window->button_algorithm[i]), 0, i, 1, 1);
02419         g_signal_connect (window->button_algorithm[i], "clicked",
02420             window_set_algorithm, NULL);
02421     }
02422     gtk_grid_attach (window->grid_algorithm,
02423         GTK_WIDGET (window->label_simulations), 0,
02424         NALGORITHMS, 1, 1);
02425     gtk_grid_attach (window->grid_algorithm,
02426         GTK_WIDGET (window->spin_simulations), 1, NALGORITHMS, 1,
02427         1);
02428     gtk_grid_attach (window->grid_algorithm,
02429         GTK_WIDGET (window->label_iterations), 0, NALGORITHMS + 1,
02430         1, 1);
02431     gtk_grid_attach (window->grid_algorithm,
02432         GTK_WIDGET (window->spin_iterations), 1, NALGORITHMS + 1,
02433         1, 1);
02434     gtk_grid_attach (window->grid_algorithm,
02435         GTK_WIDGET (window->label_tolerance), 0, NALGORITHMS + 2,
02436         1, 1);
02437     gtk_grid_attach (window->grid_algorithm,
02438         GTK_WIDGET (window->spin_tolerance), 1, NALGORITHMS + 2, 1,
02439         1);
02440     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02441         window->label_bests),
02442         0, NALGORITHMS + 3, 1, 1);
02443     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02444         window->spin_bests), 1,
02445         NALGORITHMS + 3, 1, 1);
02446     gtk_grid_attach (window->grid_algorithm,
02447         GTK_WIDGET (window->label_population), 0, NALGORITHMS + 4,
02448         1, 1);
02449     gtk_grid_attach (window->grid_algorithm,
02450         GTK_WIDGET (window->spin_population), 1, NALGORITHMS + 4,
02451         1, 1);
02452     gtk_grid_attach (window->grid_algorithm,
02453         GTK_WIDGET (window->label_generations), 0, NALGORITHMS + 5,
02454         1, 1);
02455     gtk_grid_attach (window->grid_algorithm,
02456         GTK_WIDGET (window->spin_generations), 1, NALGORITHMS + 5,
02457         1, 1);
02458     gtk_grid_attach (window->grid_algorithm,
02459         GTK_WIDGET (window->label_mutation), 0, NALGORITHMS + 6, 1,
02460         1);
02461     gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (
02462         window->spin_mutation),
02463         1, NALGORITHMS + 6, 1, 1);
02464     gtk_grid_attach (window->grid_algorithm,
02465         GTK_WIDGET (window->label_reproduction), 0,
02466         NALGORITHMS + 7, 1, 1);
02467     gtk_grid_attach (window->grid_algorithm,
02468         GTK_WIDGET (window->spin_reproduction), 1, NALGORITHMS + 7,
02469         1, 1);
02470     gtk_grid_attach (window->grid_algorithm,
02471         GTK_WIDGET (window->label_adaptation), 0, NALGORITHMS + 8,
02472         1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474         GTK_WIDGET (window->spin_adaptation), 1, NALGORITHMS + 8,

```

```

02472         1, 1);
02473     gtk_grid_attach (window->grid_algorithm,
02474         GTK_WIDGET (window->check_direction), 0, NALGORITHMS + 9,
02475         2, 1);
02476     gtk_grid_attach (window->grid_algorithm,
02477         GTK_WIDGET (window->grid_direction), 0, NALGORITHMS + 10,
02478         2, 1);
02479     gtk_grid_attach (window->grid_algorithm,
02480         GTK_WIDGET (window->label_threshold), 0, NALGORITHMS + 11,
02481         1, 1);
02482     gtk_grid_attach (window->grid_algorithm,
02483         GTK_WIDGET (window->scrolled_threshold), 1,
02484         NALGORITHMS + 11, 1, 1);
02485     window->frame_algorithm = (GtkFrame *) gtk_frame_new (_("Algorithm"));
02486     gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02487         GTK_WIDGET (window->grid_algorithm));
02488
02489     // Creating the variable widgets
02490     window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02491     gtk_widget_set_tooltip_text
02492         (GTK_WIDGET (window->combo_variable), _("Variables selector"));
02493     window->id_variable = g_signal_connect
02494         (window->combo_variable, "changed", window_set_variable, NULL);
02495     window->button_add_variable
02496         = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02497             GTK_ICON_SIZE_BUTTON);
02498     g_signal_connect
02499         (window->button_add_variable, "clicked",
02500         window_add_variable, NULL);
02501     gtk_widget_set_tooltip_text
02502         (GTK_WIDGET (window->button_add_variable), _("Add variable"));
02503     window->button_remove_variable
02504         = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02505             GTK_ICON_SIZE_BUTTON);
02506     g_signal_connect
02507         (window->button_remove_variable, "clicked",
02508         window_remove_variable, NULL);
02509     gtk_widget_set_tooltip_text
02510         (GTK_WIDGET (window->button_remove_variable), _("Remove variable"));
02511     window->label_variable = (GtkLabel *) gtk_label_new (_("Name"));
02512     window->entry_variable = (GtkEntry *) gtk_entry_new ();
02513     gtk_widget_set_tooltip_text
02514         (GTK_WIDGET (window->entry_variable), _("Variable name"));
02515     gtk_widget_set_hexexpand (GTK_WIDGET (window->entry_variable), TRUE);
02516     window->id_variable_label = g_signal_connect
02517         (window->entry_variable, "changed",
02518         window_label_variable, NULL);
02519     window->label_min = (GtkLabel *) gtk_label_new (_("Minimum"));
02520     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02521         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02522     gtk_widget_set_tooltip_text
02523         (GTK_WIDGET (window->spin_min),
02524         _("Minimum initial value of the variable"));
02525     window->scrolled_min
02526         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02527     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02528         GTK_WIDGET (window->spin_min));
02529     g_signal_connect (window->spin_min, "value-changed",
02530         window_rangemin_variable, NULL);
02531     window->label_max = (GtkLabel *) gtk_label_new (_("Maximum"));
02532     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02533         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02534     gtk_widget_set_tooltip_text
02535         (GTK_WIDGET (window->spin_max),
02536         _("Maximum initial value of the variable"));
02537     window->scrolled_max
02538         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02539     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02540         GTK_WIDGET (window->spin_max));
02541     g_signal_connect (window->spin_max, "value-changed",
02542         window_rangemax_variable, NULL);
02543     window->check_minabs = (GtkCheckButton *)
02544         gtk_check_button_new_with_mnemonic (_("Absolute minimum"));
02545     g_signal_connect (window->check_minabs, "toggled",
02546         window_update, NULL);
02547     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02548         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02549     gtk_widget_set_tooltip_text
02550         (GTK_WIDGET (window->spin_minabs),
02551         _("Minimum allowed value of the variable"));
02552     window->scrolled_minabs
02553         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02554     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02555         GTK_WIDGET (window->spin_minabs));
02556     g_signal_connect (window->spin_minabs, "value-changed",
02557         window_rangeminabs_variable, NULL);
02558     window->check_maxabs = (GtkCheckButton *)

```

```
02555     gtk_check_button_new_with_mnemonic (_("_Absolute maximum"));
02556     g_signal_connect (window->check_maxabs, "toggled",
window_update, NULL);
02557     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02558     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02559     gtk_widget_set_tooltip_text
02560     (GTK_WIDGET (window->spin_maxabs),
02561     _("Maximum allowed value of the variable"));
02562     window->scrolled_maxabs
02563     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02564     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02565     GTK_WIDGET (window->spin_maxabs));
02566     g_signal_connect (window->spin_maxabs, "value-changed",
02567     window_rangemaxabs_variable, NULL);
02568     window->label_precision
02569     = (GtkLabel *) gtk_label_new (_("Precision digits"));
02570     window->spin_precision = (GtkSpinButton *)
02571     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02572     gtk_widget_set_tooltip_text
02573     (GTK_WIDGET (window->spin_precision),
02574     _("Number of precision floating point digits\n"
02575     "0 is for integer numbers"));
02576     g_signal_connect (window->spin_precision, "value-changed",
02577     window_precision_variable, NULL);
02578     window->label_sweeps = (GtkLabel *) gtk_label_new (_("Sweeps number"));
02579     window->spin_sweeps =
02580     (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02581     gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_sweeps),
02582     _("Number of steps sweeping the variable"));
02583     g_signal_connect (window->spin_sweeps, "value-changed",
02584     window_update_variable, NULL);
02585     window->label_bits = (GtkLabel *) gtk_label_new (_("Bits number"));
02586     window->spin_bits
02587     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02588     gtk_widget_set_tooltip_text
02589     (GTK_WIDGET (window->spin_bits),
02590     _("Number of bits to encode the variable"));
02591     g_signal_connect
02592     (window->spin_bits, "value-changed", window_update_variable, NULL)
;
02593     window->label_step = (GtkLabel *) gtk_label_new (_("Step size"));
02594     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02595     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02596     gtk_widget_set_tooltip_text
02597     (GTK_WIDGET (window->spin_step),
02598     _("Initial step size for the direction search method"));
02599     window->scrolled_step
02600     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02601     gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02602     GTK_WIDGET (window->spin_step));
02603     g_signal_connect
02604     (window->spin_step, "value-changed", window_step_variable, NULL);
02605     window->grid_variable = (GtkGrid *) gtk_grid_new ();
02606     gtk_grid_attach (window->grid_variable,
02607     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02608     gtk_grid_attach (window->grid_variable,
02609     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02610     gtk_grid_attach (window->grid_variable,
02611     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02612     gtk_grid_attach (window->grid_variable,
02613     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02614     gtk_grid_attach (window->grid_variable,
02615     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02616     gtk_grid_attach (window->grid_variable,
02617     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02618     gtk_grid_attach (window->grid_variable,
02619     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02620     gtk_grid_attach (window->grid_variable,
02621     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02622     gtk_grid_attach (window->grid_variable,
02623     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02624     gtk_grid_attach (window->grid_variable,
02625     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02626     gtk_grid_attach (window->grid_variable,
02627     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02628     gtk_grid_attach (window->grid_variable,
02629     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02630     gtk_grid_attach (window->grid_variable,
02631     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02632     gtk_grid_attach (window->grid_variable,
02633     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02634     gtk_grid_attach (window->grid_variable,
02635     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02636     gtk_grid_attach (window->grid_variable,
02637     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02638     gtk_grid_attach (window->grid_variable,
02639     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
```



```

02640 gtk_grid_attach (window->grid_variable,
02641                 GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02642 gtk_grid_attach (window->grid_variable,
02643                 GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02644 gtk_grid_attach (window->grid_variable,
02645                 GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02646 gtk_grid_attach (window->grid_variable,
02647                 GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02648 window->frame_variable = (GtkFrame *) gtk_frame_new (_("Variable"));
02649 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02650                  GTK_WIDGET (window->grid_variable));
02651
02652 // Creating the experiment widgets
02653 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02654 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02655                             _("Experiment selector"));
02656 window->id_experiment = g_signal_connect
02657 (window->combo_experiment, "changed",
window_set_experiment, NULL);
02658 window->button_add_experiment
02659 = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02660                                               GTK_ICON_SIZE_BUTTON);
02661 g_signal_connect
02662 (window->button_add_experiment, "clicked",
window_add_experiment, NULL);
02663 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02664                             _("Add experiment"));
02665 window->button_remove_experiment
02666 = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02667                                               GTK_ICON_SIZE_BUTTON);
02668 g_signal_connect (window->button_remove_experiment, "clicked",
02669                  window_remove_experiment, NULL);
02670 gtk_widget_set_tooltip_text (GTK_WIDGET (window->
button_remove_experiment),
                             _("Remove experiment"));
02671 window->label_experiment
02672 = (GtkLabel *) gtk_label_new (_("Experimental data file"));
02673 window->button_experiment = (GtkFileChooserButton *)
02674 gtk_file_chooser_button_new (_("Experimental data file"),
02675                             GTK_FILE_CHOOSER_ACTION_OPEN);
02676 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02677                             _("Experimental data file"));
02678 window->id_experiment_name
02679 = g_signal_connect (window->button_experiment, "selection-changed",
02680                    window_name_experiment, NULL);
02681 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_experiment), TRUE);
02682 window->label_weight = (GtkLabel *) gtk_label_new (_("Weight"));
02683 window->spin_weight
02684 = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02685 gtk_widget_set_tooltip_text
02686 (GTK_WIDGET (window->spin_weight),
02687  _("Weight factor to build the objective function"));
02688 g_signal_connect
02689 (window->spin_weight, "value-changed",
window_weight_experiment, NULL);
02690 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02691 gtk_grid_attach (window->grid_experiment,
02692                 GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02693 gtk_grid_attach (window->grid_experiment,
02694                 GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02695 gtk_grid_attach (window->grid_experiment,
02696                 GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02697 ;
02698 gtk_grid_attach (window->grid_experiment,
02699                 GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02700 gtk_grid_attach (window->grid_experiment,
02701                 GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02702 gtk_grid_attach (window->grid_experiment,
02703                 GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02704 gtk_grid_attach (window->grid_experiment,
02705                 GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02706 for (i = 0; i < MAX_NINPUTS; ++i)
02707 {
02708     snprintf (buffer3, 64, "%s %u", _("Input template"), i + 1);
02709     window->check_template[i] = (GtkCheckButton *)
02710     gtk_check_button_new_with_label (buffer3);
02711     window->id_template[i]
02712     = g_signal_connect (window->check_template[i], "toggled",
02713                        window_inputs_experiment, NULL);
02714     gtk_grid_attach (window->grid_experiment,
02715                     GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1,
02716                     1);
02717     window->button_template[i] =
02718     (GtkFileChooserButton *)
02719     gtk_file_chooser_button_new (_("Input template"),
02720                                 GTK_FILE_CHOOSER_ACTION_OPEN);
02721     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_template[i]),

```



```

02722         _("Experimental input template file"));
02723     window->id_input[i] =
02724         g_signal_connect_swapped (window->button_template[i],
02725             "selection-changed",
02726             (void (*)(void *)) window_template_experiment,
02727             (void *) (size_t) i);
02728     gtk_grid_attach (window->grid_experiment,
02729         GTK_WIDGET (window->button_template[i]),
02730         1, 3 + i, 3, 1);
02731 }
02732 window->frame_experiment = (GtkFrame *) gtk_frame_new (_("Experiment"));
02733 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02734     GTK_WIDGET (window->grid_experiment));
02735
02736 // Creating the error norm widgets
02737 window->frame_norm = (GtkFrame *) gtk_frame_new (_("Error norm"));
02738 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02739 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02740     GTK_WIDGET (window->grid_norm));
02741 window->button_norm[0] = (GtkRadioButton *)
02742     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02743 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02744     tip_norm[0]);
02745 gtk_grid_attach (window->grid_norm,
02746     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02747 g_signal_connect (window->button_norm[0], "clicked",
02748     window_update, NULL);
02749 for (i = 0; ++i < NNORMS;)
02750 {
02751     window->button_norm[i] = (GtkRadioButton *)
02752         gtk_radio_button_new_with_mnemonic
02753         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02754     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02755         tip_norm[i]);
02756     gtk_grid_attach (window->grid_norm,
02757         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02758     g_signal_connect (window->button_norm[i], "clicked",
02759         window_update,
02760         NULL);
02761 }
02762 window->label_p = (GtkLabel *) gtk_label_new (_("P parameter"));
02763 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02764     label_p), 1, 1, 1,
02765     1);
02766 window->spin_p =
02767     (GtkSpinButton *) gtk_spin_button_new_with_range (-G_MAXDOUBLE,
02768         G_MAXDOUBLE, 0.01);
02769 gtk_widget_set_tooltip_text (GTK_WIDGET (window->spin_p),
02770     _("P parameter for the P error norm"));
02771 window->scrolled_p =
02772     (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02773 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02774     GTK_WIDGET (window->spin_p));
02775 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02776 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02777 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->
02778     scrolled_p),
02779     1, 2, 1, 2);
02780
02781 // Creating the grid and attaching the widgets to the grid
02782 window->grid = (GtkGrid *) gtk_grid_new ();
02783 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3,
02784     1);
02785 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02786 gtk_grid_attach (window->grid,
02787     GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02788 gtk_grid_attach (window->grid,
02789     GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02790 gtk_grid_attach (window->grid,
02791     GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02792 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02793 gtk_container_add (GTK_CONTAINER (window->window),
02794     GTK_WIDGET (window->grid));
02795
02796 // Setting the window logo
02797 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02798 gtk_window_set_icon (window->window, window->logo);
02799
02800 // Showing the window
02801 gtk_widget_show_all (GTK_WIDGET (window->window));
02802
02803 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02804 #if GTK_MINOR_VERSION >= 16
02805     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02806     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02807     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02808     gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02809 #endif

```

```

02805  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02806  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02807  gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1,
02808                               40);
02809  #endif
02810
02811  // Reading initial example
02812  input_new ();
02813  buffer2 = g_get_current_dir ();
02814  buffer =
02815      g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02816  g_free (buffer2);
02817  window_read (buffer);
02818  g_free (buffer);
02819
02820  #if DEBUG_INTERFACE
02821  fprintf (stderr, "window_new: start\n");
02822  #endif
02823 }

```

### 5.13.2.7 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 1904 of file [interface.c](#).

```

01905 {
01906     unsigned int i;
01907     char *buffer;
01908     #if DEBUG_INTERFACE
01909     fprintf (stderr, "window_read: start\n");
01910     #endif
01911
01912     // Reading new input file
01913     input_free ();
01914     if (!input_open (filename))
01915     {
01916     #if DEBUG_INTERFACE
01917         fprintf (stderr, "window_read: end\n");
01918     #endif
01919         return 0;
01920     }
01921
01922     // Setting GTK+ widgets data
01923     gtk_entry_set_text (window->entry_result, input->result);
01924     gtk_entry_set_text (window->entry_variables, input->
variables);
01925     buffer = g_build_filename (input->directory, input->
simulator, NULL);
01926     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
(window->button_simulator), buffer);
01927     g_free (buffer);
01928     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
(size_t) input->evaluator);
01929     if (input->evaluator)
01930     {
01931         buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01932         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER

```

```

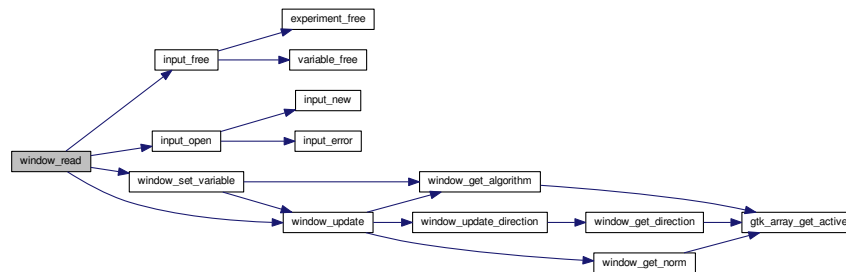
01935                                     (window->button_evaluator), buffer);
01936     g_free (buffer);
01937 }
01938 gtk_toggle_button_set_active
01939 (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01940 switch (input->algorithm)
01941 {
01942     case ALGORITHM_MONTE_CARLO:
01943         gtk_spin_button_set_value (window->spin_simulations,
01944                                     (gdouble) input->nsimulations);
01945     case ALGORITHM_SWEEP:
01946         gtk_spin_button_set_value (window->spin_iterations,
01947                                     (gdouble) input->niterations);
01948         gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01949         gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01950         gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON
01951                                     (window->check_direction),
input->nsteps);
01952         if (input->nsteps)
01953         {
01954             gtk_toggle_button_set_active
01955                 (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01956             gtk_spin_button_set_value (window->spin_steps,
01957                                         (gdouble) input->nsteps);
01958             gtk_spin_button_set_value (window->spin_relaxation,
01959                                         (gdouble) input->relaxation);
01960             switch (input->direction)
01961             {
01962                 case DIRECTION_METHOD_RANDOM:
01963                     gtk_spin_button_set_value (window->spin_estimates,
01964                                                 (gdouble) input->nestimates);
01965             }
01966             break;
01967         default:
01968             gtk_spin_button_set_value (window->spin_population,
01969                                         (gdouble) input->nsimulations);
01970             gtk_spin_button_set_value (window->spin_generations,
01971                                         (gdouble) input->niterations);
01972             gtk_spin_button_set_value (window->spin_mutation,
01973                                         input->mutation_ratio);
01974             gtk_spin_button_set_value (window->spin_reproduction,
01975                                         input->reproduction_ratio);
01976             gtk_spin_button_set_value (window->spin_adaptation,
01977                                         input->adaptation_ratio);
01978         }
01979         gtk_toggle_button_set_active
01980             (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01981         gtk_spin_button_set_value (window->spin_p, input->p);
01982         gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01983         g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01984         g_signal_handler_block (window->button_experiment,
01985                                 window->id_experiment_name);
01986         gtk_combo_box_text_remove_all (window->combo_experiment);
01987         for (i = 0; i < input->nexperiments; ++i)
01988             gtk_combo_box_text_append_text (window->combo_experiment,
01989                                             input->experiment[i].name);
01990         g_signal_handler_unblock
01991             (window->button_experiment, window->
id_experiment_name);
01992         g_signal_handler_unblock (window->combo_experiment,
01993                                 window->id_experiment);
01994         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01995         g_signal_handler_block (window->combo_variable, window->
id_variable);
01996         g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01997         gtk_combo_box_text_remove_all (window->combo_variable);
01998         for (i = 0; i < input->nvariables; ++i)
01999             gtk_combo_box_text_append_text (window->combo_variable,
02000                                             input->variable[i].name);
02001         g_signal_handler_unblock (window->entry_variable,
02002                                 window->id_variable_label);
02003         g_signal_handler_unblock (window->combo_variable, window->
id_variable);
02004         gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
02005         window_set_variable ();
02006         window_update ();
02007     }
02008 #if DEBUG_INTERFACE
02009     fprintf (stderr, "window_read: end\n");
02010 
```

```

02011 #endif
02012     return 1;
02013 }

```

Here is the call graph for this function:



#### 5.13.2.8 window\_save()

```
int window_save ( )
```

Function to save the input file.

#### Returns

1 on OK, 0 on Cancel.

Definition at line 825 of file [interface.c](#).

```

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

```

```

00861 gtk_file_filter_add_pattern (filter2, "*.js");
00862 gtk_file_filter_add_pattern (filter2, "*.JS");
00863 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865 if (input->type == INPUT_TYPE_XML)
00866     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00867 else
00868     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00869
00870 // If OK response then saving
00871 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00872 {
00873     // Setting input file type
00874     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00875     buffer = (char *) gtk_file_filter_get_name (filter1);
00876     if (!strcmp (buffer, "XML"))
00877         input->type = INPUT_TYPE_XML;
00878     else
00879         input->type = INPUT_TYPE_JSON;
00880
00881     // Adding properties to the root XML node
00882     input->simulator = gtk_file_chooser_get_filename
00883         (GTK_FILE_CHOOSER (window->button_simulator));
00884     if (gtk_toggle_button_get_active
00885         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00886         input->evaluator = gtk_file_chooser_get_filename
00887             (GTK_FILE_CHOOSER (window->button_evaluator));
00888     else
00889         input->evaluator = NULL;
00890     if (input->type == INPUT_TYPE_XML)
00891     {
00892         input->result
00893             = (char *) xmlStrdup ((const xmlChar *)
00894                 gtk_entry_get_text (window->entry_result));
00895         input->variables
00896             = (char *) xmlStrdup ((const xmlChar *)
00897                 gtk_entry_get_text
00898                     (window->entry_variables));
00899     }
00900     else
00901     {
00902         input->result =
00903             g_strdup (gtk_entry_get_text (window->entry_result));
00904         input->variables =
00905             g_strdup (gtk_entry_get_text (window->entry_variables));
00906     }
00907
00908     // Setting the algorithm
00909     switch (window_get_algorithm ())
00910     {
00911     case ALGORITHM_MONTE_CARLO:
00912         input->algorithm = ALGORITHM_MONTE_CARLO;
00913         input->nsimulations
00914             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00915         input->niterations
00916             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00917         input->tolerance =
00918             gtk_spin_button_get_value (window->spin_tolerance);
00919         input->nbest =
00920             gtk_spin_button_get_value_as_int (window->spin_bests);
00921         window_save_direction ();
00922         break;
00923     case ALGORITHM_SWEEP:
00924         input->algorithm = ALGORITHM_SWEEP;
00925         input->niterations
00926             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00927         input->tolerance =
00928             gtk_spin_button_get_value (window->spin_tolerance);
00929         input->nbest =
00930             gtk_spin_button_get_value_as_int (window->spin_bests);
00931         window_save_direction ();
00932         break;
00933     default:
00934         input->algorithm = ALGORITHM_GENETIC;
00935         input->nsimulations
00936             = gtk_spin_button_get_value_as_int (window->spin_population);
00937         input->niterations
00938             = gtk_spin_button_get_value_as_int (window->spin_generations);
00939         input->mutation_ratio
00940             = gtk_spin_button_get_value (window->spin_mutation);
00941         input->reproduction_ratio
00942             = gtk_spin_button_get_value (window->spin_reproduction);
00943         input->adaptation_ratio
00944             = gtk_spin_button_get_value (window->spin_adaptation);
00945         break;
00946     }
00947     input->norm = window_get_norm ();

```

```

00948     input->p = gtk_spin_button_get_value (window->spin_p);
00949     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00950
00951     // Saving the XML file
00952     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00953     input_save (buffer);
00954
00955     // Closing and freeing memory
00956     g_free (buffer);
00957     gtk_widget_destroy (GTK_WIDGET (dlg));
00958 #if DEBUG_INTERFACE
00959     fprintf (stderr, "window_save: end\n");
00960 #endif
00961     return 1;
00962 }
00963
00964 // Closing and freeing memory
00965 gtk_widget_destroy (GTK_WIDGET (dlg));
00966 #if DEBUG_INTERFACE
00967     fprintf (stderr, "window_save: end\n");
00968 #endif
00969     return 0;
00970 }

```

### 5.13.2.9 window\_template\_experiment()

```

void window_template_experiment (
    void * data )

```

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

#### Parameters

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

Definition at line 1542 of file [interface.c](#).

```

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

```

## 5.14 interface.h

```
00001 /*
```

```

00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef INTERFACE__H
00033 #define INTERFACE__H 1
00034
00035 #define MAX_LENGTH (DEFAULT_PRECISION + 8)
00036
00037 typedef struct
00038 {
00039     GtkWidget *dialog;
00040     GtkWidget *grid;
00041     GtkWidget *label_seed;
00042     GtkWidget *spin_seed;
00043     GtkWidget *label_threads;
00044     GtkWidget *spin_threads;
00045     GtkWidget *label_direction;
00046     GtkWidget *spin_direction;
00047 } Options;
00048
00049 typedef struct
00050 {
00051     GtkWidget *dialog;
00052     GtkWidget *label;
00053     GtkWidget *spinner;
00054     GtkWidget *grid;
00055 } Running;
00056
00057 typedef struct
00058 {
00059     GtkWidget *window;
00060     GtkWidget *grid;
00061     GtkWidget *bar_buttons;
00062     GtkWidget *button_open;
00063     GtkWidget *button_save;
00064     GtkWidget *button_run;
00065     GtkWidget *button_options;
00066     GtkWidget *button_help;
00067     GtkWidget *button_about;
00068     GtkWidget *button_exit;
00069     GtkWidget *grid_files;
00070     GtkWidget *label_simulator;
00071     GtkWidget *FileChooserButton *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *FileChooserButton *button_evaluator;
00074     GtkWidget *label_result;
00075     GtkWidget *entry_result;
00076     GtkWidget *label_variables;
00077     GtkWidget *entry_variables;
00078     GtkWidget *frame_norm;
00079     GtkWidget *grid_norm;
00080     GtkWidget *radioButton *button_norm[NNORMS];
00081     GtkWidget *label_p;
00082     GtkWidget *spin_p;
00083     GtkWidget *scrolled_p;
00084     GtkWidget *frame_algorithm;
00085     GtkWidget *grid_algorithm;
00086     GtkWidget *radioButton *button_algorithm[NALGORITHMS];
00087     GtkWidget *label_simulations;
00088     GtkWidget *spin_simulations;

```

```

00117   GtkWidget *label_iterations;
00118   GtkSpinButton *spin_iterations;
00120   GtkWidget *label_tolerance;
00121   GtkSpinButton *spin_tolerance;
00122   GtkWidget *label_bests;
00123   GtkSpinButton *spin_bests;
00124   GtkWidget *label_population;
00125   GtkSpinButton *spin_population;
00127   GtkWidget *label_generations;
00128   GtkSpinButton *spin_generations;
00130   GtkWidget *label_mutation;
00131   GtkSpinButton *spin_mutation;
00132   GtkWidget *label_reproduction;
00133   GtkSpinButton *spin_reproduction;
00135   GtkWidget *label_adaptation;
00136   GtkSpinButton *spin_adaptation;
00138   GtkCheckButton *check_direction;
00140   GtkWidget *grid_direction;
00142   GtkRadioButton *button_direction[NDIRECTIONS];
00144   GtkWidget *label_steps;
00145   GtkSpinButton *spin_steps;
00146   GtkWidget *label_estimates;
00147   GtkSpinButton *spin_estimates;
00149   GtkWidget *label_relaxation;
00151   GtkSpinButton *spin_relaxation;
00153   GtkWidget *label_threshold;
00154   GtkSpinButton *spin_threshold;
00155   GtkScrolledWindow *scrolled_threshold;
00157   GtkFrame *frame_variable;
00158   GtkWidget *grid_variable;
00159   GtkComboBoxText *combo_variable;
00161   GtkButton *button_add_variable;
00162   GtkButton *button_remove_variable;
00163   GtkWidget *label_variable;
00164   GtkEntry *entry_variable;
00165   GtkWidget *label_min;
00166   GtkSpinButton *spin_min;
00167   GtkScrolledWindow *scrolled_min;
00168   GtkWidget *label_max;
00169   GtkSpinButton *spin_max;
00170   GtkScrolledWindow *scrolled_max;
00171   GtkCheckButton *check_minabs;
00172   GtkSpinButton *spin_minabs;
00173   GtkScrolledWindow *scrolled_minabs;
00174   GtkCheckButton *check_maxabs;
00175   GtkSpinButton *spin_maxabs;
00176   GtkScrolledWindow *scrolled_maxabs;
00177   GtkWidget *label_precision;
00178   GtkSpinButton *spin_precision;
00179   GtkWidget *label_sweeps;
00180   GtkSpinButton *spin_sweeps;
00181   GtkWidget *label_bits;
00182   GtkSpinButton *spin_bits;
00183   GtkWidget *label_step;
00184   GtkSpinButton *spin_step;
00185   GtkScrolledWindow *scrolled_step;
00186   GtkFrame *frame_experiment;
00187   GtkWidget *grid_experiment;
00188   GtkComboBoxText *combo_experiment;
00189   GtkButton *button_add_experiment;
00190   GtkButton *button_remove_experiment;
00191   GtkWidget *label_experiment;
00192   GtkFileChooserButton *button_experiment;
00194   GtkWidget *label_weight;
00195   GtkSpinButton *spin_weight;
00196   GtkCheckButton *check_template[MAX_NINPUTS];
00198   GtkFileChooserButton *button_template[MAX_NINPUTS];
00200   GdkPixbuf *logo;
00201   Experiment *experiment;
00202   Variable *variable;
00203   char *application_directory;
00204   gulong id_experiment;
00205   gulong id_experiment_name;
00206   gulong id_variable;
00207   gulong id_variable_label;
00208   gulong id_template[MAX_NINPUTS];
00210   gulong id_input[MAX_NINPUTS];
00212   unsigned int nexperiments;
00213   unsigned int nvariables;
00214 } Window;
00215
00216 // Global variables
00217 extern const char *logo[];
00218 extern Options options[1];
00219 extern Running running[1];
00220 extern Window window[1];
00221

```



```

00222 // Inline functions
00223 #if GTK_MINOR_VERSION < 10
00224 static inline GtkWidget *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227     GtkWidget *button;
00228     GtkWidget *image;
00229     button = (GtkWidget *) gtk_button_new ();
00230     image = (GtkWidget *) gtk_image_new_from_icon_name (name, size);
00231     gtk_button_set_image (button, GTK_WIDGET (image));
00232     return button;
00233 }
00234 #endif
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);
00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_direction ();
00243 unsigned int window_get_norm ();
00244 void window_save_direction ();
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_direction ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new (GtkApplication * application);
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"

```



## Parameters

<i>argn</i>	Arguments number.
<i>argc</i>	Arguments pointer.

## Returns

0 on success, >0 on error.

Definition at line 86 of file [main.c](#).

```

00088 {
00089     #if HAVE_GTK
00090         GtkApplication *application;
00091         char *buffer;
00092     #endif
00093
00094     // Starting pseudo-random numbers generator
00095     #if DEBUG_MAIN
00096         fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00097     #endif
00098     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00099
00100     // Allowing spaces in the XML data file
00101     #if DEBUG_MAIN
00102         fprintf (stderr, "main: allowing spaces in the XML data file\n");
00103     #endif
00104     xmlKeepBlanksDefault (0);
00105
00106     // Starting MPI
00107     #if HAVE_MPI
00108     #if DEBUG_MAIN
00109         fprintf (stderr, "main: starting MPI\n");
00110     #endif
00111     MPI_Init (&argn, &argc);
00112     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00113     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00114     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00115     #else
00116     ntasks = 1;
00117     #endif
00118
00119     // Resetting result and variables file names
00120     #if DEBUG_MAIN
00121         fprintf (stderr, "main: resetting result and variables file names\n");
00122     #endif
00123     input->result = input->variables = NULL;
00124
00125     #if HAVE_GTK
00126
00127     // Getting threads number and pseudo-random numbers generator seed
00128     nthreads_direction = nthreads = cores_number ();
00129     optimize->seed = DEFAULT_RANDOM_SEED;
00130
00131     // Setting local language and international floating point numbers notation
00132     setlocale (LC_ALL, "");
00133     setlocale (LC_NUMERIC, "C");
00134     window->application_directory = g_get_current_dir ();
00135     buffer = g_build_filename (window->application_directory,
00136                               LOCALE_DIR, NULL);
00137     bindtextdomain (PROGRAM_INTERFACE, buffer);
00138     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00139     textdomain (PROGRAM_INTERFACE);
00140
00141     // Initing GTK+
00142     #if !EXTERNAL_LIBRARY
00143         show_pending = process_pending;
00144     #endif
00145     gtk_disable_setlocale ();
00146     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00147                                       G_APPLICATION_FLAGS_NONE);
00148     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00149
00150     // Opening the main window
00151     g_application_run (G_APPLICATION (application), 0, NULL);
00152
00153     // Freeing memory
00154     input_free ();

```

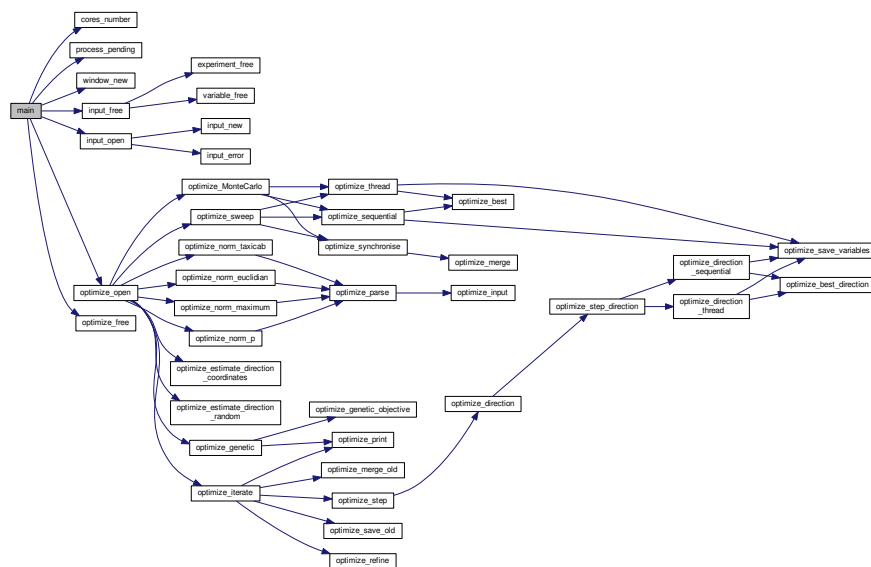
```

00154     g_free (buffer);
00155     gtk_widget_destroy (GTK_WIDGET (window->window));
00156     g_object_unref (application);
00157     g_free (window->application_directory);
00158
00159 #else
00160
00161     // Checking syntax
00162     if (argn < 2)
00163     {
00164         printf ("The syntax is:\n"
00165             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166             "[variables_file]\n");
00167         return 1;
00168     }
00169
00170     // Getting threads number and pseudo-random numbers generator seed
00171 #if DEBUG_MAIN
00172     fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00173         "generator seed\n");
00174 #endif
00175     nthreads_direction = nthreads = cores_number ();
00176     optimize->seed = DEFAULT_RANDOM_SEED;
00177     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00178     {
00179         nthreads_direction = nthreads = atoi (argc[2]);
00180         if (!nthreads)
00181         {
00182             printf ("Bad threads number\n");
00183             return 2;
00184         }
00185         argc += 2;
00186         argn -= 2;
00187         if (argn > 2 && !strcmp (argc[1], "-seed"))
00188         {
00189             optimize->seed = atoi (argc[2]);
00190             argc += 2;
00191             argn -= 2;
00192         }
00193     }
00194     else if (argn > 2 && !strcmp (argc[1], "-seed"))
00195     {
00196         optimize->seed = atoi (argc[2]);
00197         argc += 2;
00198         argn -= 2;
00199         if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00200         {
00201             nthreads_direction = nthreads = atoi (argc[2]);
00202             if (!nthreads)
00203             {
00204                 printf ("Bad threads number\n");
00205                 return 2;
00206             }
00207             argc += 2;
00208             argn -= 2;
00209         }
00210     }
00211     printf ("nthreads=%u\n", nthreads);
00212     printf ("seed=%lu\n", optimize->seed);
00213
00214     // Checking arguments
00215 #if DEBUG_MAIN
00216     fprintf (stderr, "main: checking arguments\n");
00217 #endif
00218     if (argn > 4 || argn < 2)
00219     {
00220         printf ("The syntax is:\n"
00221             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00222             "[variables_file]\n");
00223         return 1;
00224     }
00225     if (argn > 2)
00226         input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00227     if (argn == 4)
00228         input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00229
00230     // Making optimization
00231 #if DEBUG_MAIN
00232     fprintf (stderr, "main: making optimization\n");
00233 #endif
00234     if (input_open (argc[1]))
00235         optimize_open ();
00236
00237     // Freeing memory
00238 #if DEBUG_MAIN
00239     fprintf (stderr, "main: freeing memory and closing\n");
00240 #endif

```

```
00241     optimize_free ();
00242
00243 #endif
00244
00245 // Closing MPI
00246 #if HAVE_MPI
00247     MPI_Finalize ();
00248 #endif
00249
00250 // Freeing memory
00251     gsl_rng_free (optimize->rng);
00252
00253 // Closing
00254     return 0;
00255 }
```

Here is the call graph for this function:



## 5.16 main.c

```

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

```

```

00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <locale.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>
00042 #include <glib.h>
00043 #include <json-glib/json-glib.h>
00044 #ifdef G_OS_WIN32
00045 #include <windows.h>
00046 #endif
00047 #if HAVE_MPI
00048 #include <mpi.h>
00049 #endif
00050 #if HAVE_GTK
00051 #include <gio/gio.h>
00052 #include <gtk/gtk.h>
00053 #endif
00054 #include "genetic/genetic.h"
00055 #include "utils.h"
00056 #include "experiment.h"
00057 #include "variable.h"
00058 #include "input.h"
00059 #include "optimize.h"
00060 #if HAVE_GTK
00061 #include "interface.h"
00062 #endif
00063
00064 #define DEBUG_MAIN 0
00065
00066 #if EXTERNAL_LIBRARY
00067 int
00068 mpcotool (int argn, char **argc)
00069 #else
00070 int
00071 main (int argn, char **argc)
00072 #endif
00073 {
00074     #if HAVE_GTK
00075     GtkApplication *application;
00076     char *buffer;
00077     #endif
00078
00079     // Starting pseudo-random numbers generator
00080     #if DEBUG_MAIN
00081     fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00082     #endif
00083     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00084
00085     // Allowing spaces in the XML data file
00086     #if DEBUG_MAIN
00087     fprintf (stderr, "main: allowing spaces in the XML data file\n");
00088     #endif
00089     xmlKeepBlanksDefault (0);
00090
00091     // Starting MPI
00092     #if HAVE_MPI
00093     #if DEBUG_MAIN
00094     fprintf (stderr, "main: starting MPI\n");
00095     #endif
00096     MPI_Init (&argn, &argc);
00097     MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00098     MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00099     printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00100     #else
00101     ntasks = 1;
00102     #endif
00103
00104     // Resetting result and variables file names
00105     #if DEBUG_MAIN
00106     fprintf (stderr, "main: resetting result and variables file names\n");
00107     #endif
00108     input->result = input->variables = NULL;
00109
00110     #if HAVE_GTK
00111
00112     // Getting threads number and pseudo-random numbers generator seed
00113     nthreads_direction = nthreads = cores_number ();

```

```

00129     optimize->seed = DEFAULT_RANDOM_SEED;
00130
00131     // Setting local language and international floating point numbers notation
00132     setlocale (LC_ALL, "");
00133     setlocale (LC_NUMERIC, "C");
00134     window->application_directory = g_get_current_dir ();
00135     buffer = g_build_filename (window->application_directory,
00136                               LOCALE_DIR, NULL);
00136     bindtextdomain (PROGRAM_INTERFACE, buffer);
00137     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00138     textdomain (PROGRAM_INTERFACE);
00139
00140     // Initing GTK+
00141     #if !EXTERNAL_LIBRARY
00142         show_pending = process_pending;
00143     #endif
00144     gtk_disable_setlocale ();
00145     application = gtk_application_new ("es.csic.eead.auladei.sprinkler",
00146                                       G_APPLICATION_FLAGS_NONE);
00147     g_signal_connect (application, "activate", G_CALLBACK (window_new), NULL);
00148
00149     // Opening the main window
00150     g_application_run (G_APPLICATION (application), 0, NULL);
00151
00152     // Freeing memory
00153     input_free ();
00154     g_free (buffer);
00155     gtk_widget_destroy (GTK_WIDGET (window->window));
00156     g_object_unref (application);
00157     g_free (window->application_directory);
00158
00159     #else
00160
00161     // Checking syntax
00162     if (argn < 2)
00163     {
00164         printf ("The syntax is:\n"
00165                "  ./mpccotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00166                "[variables_file]\n");
00167         return 1;
00168     }
00169
00170     // Getting threads number and pseudo-random numbers generator seed
00171     #if DEBUG_MAIN
00172         fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00173                "generator seed\n");
00174     #endif
00175     nthreads_direction = nthreads = cores_number ();
00176     optimize->seed = DEFAULT_RANDOM_SEED;
00177     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00178     {
00179         nthreads_direction = nthreads = atoi (argc[2]);
00180         if (!nthreads)
00181         {
00182             printf ("Bad threads number\n");
00183             return 2;
00184         }
00185         argc += 2;
00186         argn -= 2;
00187         if (argn > 2 && !strcmp (argc[1], "-seed"))
00188         {
00189             optimize->seed = atoi (argc[2]);
00190             argc += 2;
00191             argn -= 2;
00192         }
00193     }
00194     else if (argn > 2 && !strcmp (argc[1], "-seed"))
00195     {
00196         optimize->seed = atoi (argc[2]);
00197         argc += 2;
00198         argn -= 2;
00199         if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00200         {
00201             nthreads_direction = nthreads = atoi (argc[2]);
00202             if (!nthreads)
00203             {
00204                 printf ("Bad threads number\n");
00205                 return 2;
00206             }
00207             argc += 2;
00208             argn -= 2;
00209         }
00210     }
00211     printf ("nthreads=%u\n", nthreads);
00212     printf ("seed=%lu\n", optimize->seed);
00213
00214     // Checking arguments

```

```

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

```

## 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"

```



```

graph TD
    genetics[genetics] --> config[config.h]
    genetics --> stdio[stdio.h]
    genetics --> stdlib[stdlib.h]
    genetics --> string[string.h]
    genetics --> math[math.h]
    genetics --> sysparam[sysparam.h]
    genetics --> glib_glib[glib/glib.h]
    genetics --> libintl_param[libintl/param.h]
    genetics --> libintl[libintl.h]
    genetics --> glib[glib.h]
    genetics --> glib_glib[glib/glib.h]
    genetics --> jansson_jansson[jansson/jansson.h]
    genetics --> libintl[libintl.h]
    genetics --> rpl[rpl.h]
    genetics --> genetic_genetic[genetic/genetic.h]
    genetics --> libintl[libintl.h]
    genetics --> segment[segment.h]
    genetics --> libintl[libintl.h]
    genetics --> libintl[libintl.h]

```

- `#define GNU_SOURCE`
- `#define DEBUG_OPTIMIZE 0`  
*Macro to debug optimize functions.*
- `#define RM "rm"`  
*Macro to define the shell remove command.*

- 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 468 of file [optimize.c](#).

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

```

00495         else
00496             break;
00497     }
00498 }
00499 #if DEBUG_OPTIMIZE
00500 fprintf (stderr, "optimize_best: end\n");
00501 #endif
00502 }

```

### 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 795 of file [optimize.c](#).

```

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

```

### 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.
-------------------	--------------------

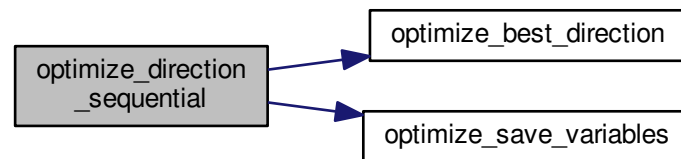
Definition at line 825 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



#### 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

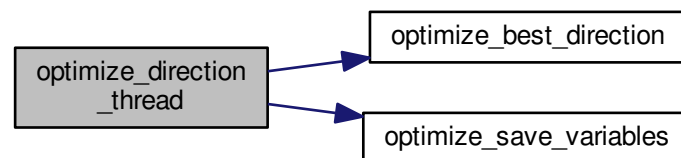
Definition at line 863 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



#### 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 937 of file [optimize.c](#).

```

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

```

#### 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 910 of file [optimize.c](#).

```

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

```

#### 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 1104 of file [optimize.c](#).

```

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

```

**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>	<a href="#">Input</a> 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,
00125         content);
00126 #endif
00127 file = g_fopen (input, "w");
00128
00129 // Parsing template
00130 for (i = 0; i < optimize->nvariables; ++i)
00131 {
00132 #if DEBUG_OPTIMIZE
00133     fprintf (stderr, "optimize_input: variable=%u\n", i);
00134 #endif
00135     snprintf (buffer, 32, "@variable%u@", i + 1);
00136     regex = g_regex_new (buffer, 0, 0, NULL);
00137     if (i == 0)
00138     {
00139         buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                         optimize->label[i], 0, NULL);
00141 #if DEBUG_OPTIMIZE
00142         fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143 #endif
00144     }
00145     else
00146     {
00147         length = strlen (buffer3);
00148         buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                         optimize->label[i], 0, NULL);
00150         g_free (buffer3);
00151     }
00152     g_regex_unref (regex);
00153     length = strlen (buffer2);
00154     snprintf (buffer, 32, "@value%u@", i + 1);
00155     regex = g_regex_new (buffer, 0, 0, NULL);
00156     snprintf (value, 32, format[optimize->precision[i]],
00157             optimize->value[simulation * optimize->
nvariables + i]);
00158 #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 591 of file [optimize.c](#).

```

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

```

#### 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.

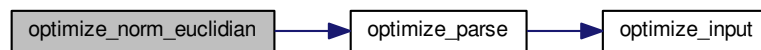
Definition at line 300 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



#### 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.

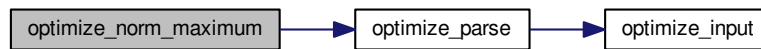
Definition at line 329 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



#### 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

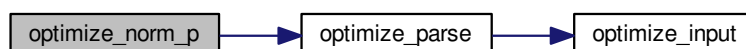
P error norm.

Definition at line 357 of file [optimize.c](#).

```

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

Here is the call graph for this function:



## 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.
-------------------	--------------------

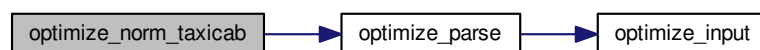
## Returns

Taxicab error norm.

Definition at line 386 of file [optimize.c](#).

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

Here is the call graph for this function:



## 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>	<a href="#">Experiment</a> number.

## Returns

Objective function value.

Definition at line 191 of file [optimize.c](#).

```

00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
00198
00199     #if DEBUG_OPTIMIZE
00200     fprintf (stderr, "optimize_parse: start\n");
00201     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202             simulation, experiment);
00203     #endif
00204
00205     // Opening input files
00206     for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209                 experiment);
00210     #if DEBUG_OPTIMIZE
00211         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212     #endif
00213         optimize_input (simulation, &input[i][0],
00214                         optimize->file[i][experiment]);
00215     }
00216     for (; i < MAX_NINPUTS; ++i)
00217         strcpy (&input[i][0], "");
00218     #if DEBUG_OPTIMIZE
00219     fprintf (stderr, "optimize_parse: parsing end\n");
00220     #endif
00221
00222     // Performing the simulation
00223     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224     buffer2 = g_path_get_dirname (optimize->simulator);
00225     buffer3 = g_path_get_basename (optimize->simulator);
00226     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
00228             buffer4, input[0], input[1], input[2], input[3], input[4],
00229             input[5], input[6], input[7], output);
00230     g_free (buffer4);
00231     g_free (buffer3);
00232     g_free (buffer2);
00233     #if DEBUG_OPTIMIZE
00234     fprintf (stderr, "optimize_parse: %s\n", buffer);
00235     #endif
00236     system (buffer);
00237
00238     // Checking the objective value function
00239     if (optimize->evaluator)
00240     {
00241         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242         buffer2 = g_path_get_dirname (optimize->evaluator);
00243         buffer3 = g_path_get_basename (optimize->evaluator);
00244         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245         snprintf (buffer, 512, "%s\ " %s %s %s",
00246                 buffer4, output, optimize->experiment[experiment], result);
00247         g_free (buffer4);
00248         g_free (buffer3);
00249         g_free (buffer2);
00250     #if DEBUG_OPTIMIZE
00251         fprintf (stderr, "optimize_parse: %s\n", buffer);
00252     #endif
00253         system (buffer);
00254         file_result = g_fopen (result, "r");
00255         e = atof (fgets (buffer, 512, file_result));
00256         fclose (file_result);
00257     }
00258     else
00259     {
00260         strcpy (result, "");
00261         file_result = g_fopen (output, "r");
00262         e = atof (fgets (buffer, 512, file_result));
00263         fclose (file_result);
00264     }
00265
00266     // Removing files
00267     #if !DEBUG_OPTIMIZE
00268     for (i = 0; i < optimize->ninputs; ++i)
00269     {

```

```

00270     if (optimize->file[i][0])
00271     {
00272         snprintf (buffer, 512, RM " %s", &input[i][0]);
00273         system (buffer);
00274     }
00275 }
00276 snprintf (buffer, 512, RM " %s %s", output, result);
00277 system (buffer);
00278 #endif
00279 // Processing pending events
00280 if (show_pending)
00281     show_pending ();
00282
00283 #if DEBUG_OPTIMIZE
00284 fprintf (stderr, "optimize_parse: end\n");
00285 #endif
00286 // Returning the objective function
00287 return e * optimize->weight[experiment];
00288 }

```

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 439 of file [optimize.c](#).

```

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

```

```

00454 #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_save_variables: end\n");
00456 #endif
00457 }

```

### 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 968 of file [optimize.c](#).

```

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

```

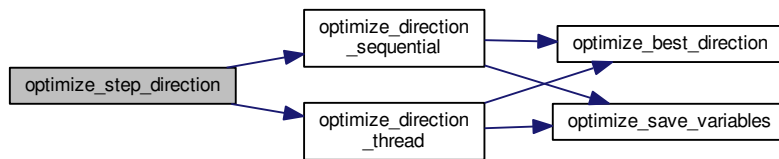


```

01021         data[i].thread = i;
01022         thread[i] = g_thread_new
01023             (NULL, (void (*)(void*)) optimize_direction_thread, &data[i]);
01024     }
01025     for (i = 0; i < nthreads_direction; ++i)
01026         g_thread_join (thread[i]);
01027 }
01028 #if DEBUG_OPTIMIZE
01029 fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031 }

```

Here is the call graph for this function:



#### 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 545 of file [optimize.c](#).

```

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

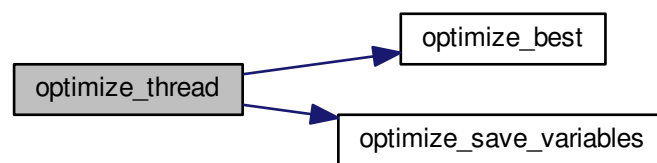
```

```

00565     g_mutex_unlock (mutex);
00566     if (optimize->stop)
00567         break;
00568 #if DEBUG_OPTIMIZE
00569     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570 #endif
00571 }
00572 #if DEBUG_OPTIMIZE
00573     fprintf (stderr, "optimize_thread: end\n");
00574 #endif
00575     g_thread_exit (NULL);
00576     return NULL;
00577 }

```

Here is the call graph for this function:



## 5.18 optimize.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <sys/param.h>
00039 #include <gsl/gsl_rng.h>
00040 #include <libxml/parser.h>
00041 #include <libintl.h>

```

```

00048 #include <glib.h>
00049 #include <glib/gstdio.h>
00050 #include <json-glib/json-glib.h>
00051 #ifdef G_OS_WIN32
00052 #include <windows.h>
00053 #elif !defined(__BSD_VISIBLE) && !defined(NetBSD)
00054 #include <alloca.h>
00055 #endif
00056 #if HAVE_MPI
00057 #include <mpi.h>
00058 #endif
00059 #include "genetic/genetic.h"
00060 #include "utils.h"
00061 #include "experiment.h"
00062 #include "variable.h"
00063 #include "input.h"
00064 #include "optimize.h"
00065
00066 #define DEBUG_OPTIMIZE 0
00067
00068
00072 #ifdef G_OS_WIN32
00073 #define RM "del"
00074 #else
00075 #define RM "rm"
00076 #endif
00077
00078 int ntasks;
00079 unsigned int nthreads;
00080 unsigned int nthreads_direction;
00082 GMutex mutex[1];
00083 void (*optimize_algorithm) ();
00085 double (*optimize_estimate_direction) (unsigned int variable,
00086                                       unsigned int estimate);
00088 double (*optimize_norm) (unsigned int simulation);
00090 Optimize optimize[1];
00091
00103 void
00104 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00105 {
00106     unsigned int i;
00107     char buffer[32], value[32], *buffer2, *buffer3, *content;
00108     FILE *file;
00109     gsize length;
00110     GRegex *regex;
00111
00112     #if DEBUG_OPTIMIZE
00113     fprintf (stderr, "optimize_input: start\n");
00114     #endif
00115
00116     // Checking the file
00117     if (!template)
00118         goto optimize_input_end;
00119
00120     // Opening template
00121     content = g_mapped_file_get_contents (template);
00122     length = g_mapped_file_get_length (template);
00123     #if DEBUG_OPTIMIZE
00124     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length,
00125             content);
00126     #endif
00127     file = g_fopen (input, "w");
00128
00129     // Parsing template
00130     for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132         #if DEBUG_OPTIMIZE
00133         fprintf (stderr, "optimize_input: variable=%u\n", i);
00134         #endif
00135         snprintf (buffer, 32, "@variable%u@", i + 1);
00136         regex = g_regex_new (buffer, 0, 0, NULL);
00137         if (i == 0)
00138         {
00139             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00140                                             optimize->label[i], 0, NULL);
00141             #if DEBUG_OPTIMIZE
00142             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143             #endif
00144         }
00145         else
00146         {
00147             length = strlen (buffer3);
00148             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                             optimize->label[i], 0, NULL);
00150             g_free (buffer3);
00151         }
00152         g_regex_unref (regex);

```

```

00153     length = strlen (buffer2);
00154     snprintf (buffer, 32, "@value%u@", i + 1);
00155     regex = g_regex_new (buffer, 0, 0, NULL);
00156     snprintf (value, 32, format[optimize->precision[i]],
00157         optimize->value[simulation * optimize->nvariables + i]);
00158
00159 #if DEBUG_OPTIMIZE
00160     fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162     buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163         0, NULL);
00164     g_free (buffer2);
00165     g_regex_unref (regex);
00166 }
00167
00168 // Saving input file
00169 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170 g_free (buffer3);
00171 fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }
00179
00190 double
00191 optimize_parse (unsigned int simulation, unsigned int experiment)
00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
00198
00199 #if DEBUG_OPTIMIZE
00200     fprintf (stderr, "optimize_parse: start\n");
00201     fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202         simulation, experiment);
00203 #endif
00204
00205     // Opening input files
00206     for (i = 0; i < optimize->ninputs; ++i)
00207     {
00208         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209             experiment);
00210 #if DEBUG_OPTIMIZE
00211         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
00213         optimize_input (simulation, &input[i][0],
00214             optimize->file[i][experiment]);
00215     }
00216     for (; i < MAX_NINPUTS; ++i)
00217         strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219     fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222     // Performing the simulation
00223     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224     buffer2 = g_path_get_dirname (optimize->simulator);
00225     buffer3 = g_path_get_basename (optimize->simulator);
00226     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227     snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
00228         buffer4, input[0], input[1], input[2], input[3], input[4],
00229         input[5], input[6], input[7], output);
00230     g_free (buffer4);
00231     g_free (buffer3);
00232     g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234     fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236     system (buffer);
00237
00238     // Checking the objective value function
00239     if (optimize->evaluator)
00240     {
00241         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242         buffer2 = g_path_get_dirname (optimize->evaluator);
00243         buffer3 = g_path_get_basename (optimize->evaluator);
00244         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245         snprintf (buffer, 512, "%s\ " %s %s %s",
00246             buffer4, output, optimize->experiment[experiment], result);
00247         g_free (buffer4);
00248         g_free (buffer3);
00249         g_free (buffer2);

```

```

00250 #if DEBUG_OPTIMIZE
00251     fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253     system (buffer);
00254     file_result = g_fopen (result, "r");
00255     e = atof (fgets (buffer, 512, file_result));
00256     fclose (file_result);
00257 }
00258 else
00259 {
00260     strcpy (result, "");
00261     file_result = g_fopen (output, "r");
00262     e = atof (fgets (buffer, 512, file_result));
00263     fclose (file_result);
00264 }
00265
00266 // Removing files
00267 #if !DEBUG_OPTIMIZE
00268 for (i = 0; i < optimize->ninputs; ++i)
00269 {
00270     if (optimize->file[i][0])
00271     {
00272         snprintf (buffer, 512, RM " %s", &input[i][0]);
00273         system (buffer);
00274     }
00275 }
00276 snprintf (buffer, 512, RM " %s %s", output, result);
00277 system (buffer);
00278 #endif
00279
00280 // Processing pending events
00281 if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE
00285     fprintf (stderr, "optimize_parse: end\n");
00286 #endif
00287
00288 // Returning the objective function
00289 return e * optimize->weight[experiment];
00290 }
00291
00292 double
00300 optimize_norm_euclidian (unsigned int simulation)
00301 {
00302     double e, ei;
00303     unsigned int i;
00304     #if DEBUG_OPTIMIZE
00305     fprintf (stderr, "optimize_norm_euclidian: start\n");
00306     #endif
00307     e = 0.;
00308     for (i = 0; i < optimize->nexperiments; ++i)
00309     {
00310         ei = optimize_parse (simulation, i);
00311         e += ei * ei;
00312     }
00313     e = sqrt (e);
00314     #if DEBUG_OPTIMIZE
00315     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316     fprintf (stderr, "optimize_norm_euclidian: end\n");
00317     #endif
00318     return e;
00319 }
00320
00321 double
00329 optimize_norm_maximum (unsigned int simulation)
00330 {
00331     double e, ei;
00332     unsigned int i;
00333     #if DEBUG_OPTIMIZE
00334     fprintf (stderr, "optimize_norm_maximum: start\n");
00335     #endif
00336     e = 0.;
00337     for (i = 0; i < optimize->nexperiments; ++i)
00338     {
00339         ei = fabs (optimize_parse (simulation, i));
00340         e = fmax (e, ei);
00341     }
00342     #if DEBUG_OPTIMIZE
00343     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00344     fprintf (stderr, "optimize_norm_maximum: end\n");
00345     #endif
00346     return e;
00347 }
00348
00349 double
00357 optimize_norm_p (unsigned int simulation)

```

```

00358 {
00359     double e, ei;
00360     unsigned int i;
00361     #if DEBUG_OPTIMIZE
00362     fprintf (stderr, "optimize_norm_p: start\n");
00363     #endif
00364     e = 0.;
00365     for (i = 0; i < optimize->nexperiments; ++i)
00366     {
00367         ei = fabs (optimize_parse (simulation, i));
00368         e += pow (ei, optimize->p);
00369     }
00370     e = pow (e, 1. / optimize->p);
00371     #if DEBUG_OPTIMIZE
00372     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00373     fprintf (stderr, "optimize_norm_p: end\n");
00374     #endif
00375     return e;
00376 }
00377
00385 double
00386 optimize_norm_taxicab (unsigned int simulation)
00387 {
00388     double e;
00389     unsigned int i;
00390     #if DEBUG_OPTIMIZE
00391     fprintf (stderr, "optimize_norm_taxicab: start\n");
00392     #endif
00393     e = 0.;
00394     for (i = 0; i < optimize->nexperiments; ++i)
00395         e += fabs (optimize_parse (simulation, i));
00396     #if DEBUG_OPTIMIZE
00397     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00398     fprintf (stderr, "optimize_norm_taxicab: end\n");
00399     #endif
00400     return e;
00401 }
00402
00407 void
00408 optimize_print ()
00409 {
00410     unsigned int i;
00411     char buffer[512];
00412     #if HAVE_MPI
00413     if (optimize->mpi_rank)
00414         return;
00415     #endif
00416     printf ("%s\n", _("Best result"));
00417     fprintf (optimize->file_result, "%s\n", _("Best result"));
00418     printf ("error = %.15le\n", optimize->error_old[0]);
00419     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00420     for (i = 0; i < optimize->nvariables; ++i)
00421     {
00422         snprintf (buffer, 512, "%s = %s\n",
00423                 optimize->label[i], format[optimize->precision[i]]);
00424         printf (buffer, optimize->value_old[i]);
00425         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00426     }
00427     fflush (optimize->file_result);
00428 }
00429
00438 void
00439 optimize_save_variables (unsigned int simulation, double error)
00440 {
00441     unsigned int i;
00442     char buffer[64];
00443     #if DEBUG_OPTIMIZE
00444     fprintf (stderr, "optimize_save_variables: start\n");
00445     #endif
00446     for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449         fprintf (optimize->file_variables, buffer,
00450                 optimize->value[simulation * optimize->nvariables + i]);
00451     }
00452     fprintf (optimize->file_variables, "%.14le\n", error);
00453     fflush (optimize->file_variables);
00454     #if DEBUG_OPTIMIZE
00455     fprintf (stderr, "optimize_save_variables: end\n");
00456     #endif
00457 }
00458
00467 void
00468 optimize_best (unsigned int simulation, double value)
00469 {
00470     unsigned int i, j;

```

```

00471     double e;
00472     #if DEBUG_OPTIMIZE
00473     fprintf (stderr, "optimize_best: start\n");
00474     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475             optimize->nsaveds, optimize->nbest);
00476     #endif
00477     if (optimize->nsaveds < optimize->nbest
00478         || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480         if (optimize->nsaveds < optimize->nbest)
00481             ++optimize->nsaveds;
00482         optimize->error_best[optimize->nsaveds - 1] = value;
00483         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484         for (i = optimize->nsaveds; --i;)
00485         {
00486             if (optimize->error_best[i] < optimize->error_best[i - 1])
00487             {
00488                 j = optimize->simulation_best[i];
00489                 e = optimize->error_best[i];
00490                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00491                 optimize->error_best[i] = optimize->error_best[i - 1];
00492                 optimize->simulation_best[i - 1] = j;
00493                 optimize->error_best[i - 1] = e;
00494             }
00495             else
00496                 break;
00497         }
00498     }
00499     #if DEBUG_OPTIMIZE
00500     fprintf (stderr, "optimize_best: end\n");
00501     #endif
00502 }
00503
00508 void
00509 optimize_sequential ()
00510 {
00511     unsigned int i;
00512     double e;
00513     #if DEBUG_OPTIMIZE
00514     fprintf (stderr, "optimize_sequential: start\n");
00515     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00516             optimize->nstart, optimize->nend);
00517     #endif
00518     for (i = optimize->nstart; i < optimize->nend; ++i)
00519     {
00520         e = optimize_norm (i);
00521         optimize_best (i, e);
00522         optimize_save_variables (i, e);
00523         if (e < optimize->threshold)
00524         {
00525             optimize->stop = 1;
00526             break;
00527         }
00528     }
00529     #if DEBUG_OPTIMIZE
00529     fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00530     #endif
00531 }
00532 #if DEBUG_OPTIMIZE
00533 fprintf (stderr, "optimize_sequential: end\n");
00534 #endif
00535 }
00536
00544 void *
00545 optimize_thread (ParallelData * data)
00546 {
00547     unsigned int i, thread;
00548     double e;
00549     #if DEBUG_OPTIMIZE
00550     fprintf (stderr, "optimize_thread: start\n");
00551     #endif
00552     thread = data->thread;
00553     #if DEBUG_OPTIMIZE
00554     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00555             optimize->thread[thread], optimize->thread[thread + 1]);
00556     #endif
00557     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00558     {
00559         e = optimize_norm (i);
00560         g_mutex_lock (mutex);
00561         optimize_best (i, e);
00562         optimize_save_variables (i, e);
00563         if (e < optimize->threshold)
00564             optimize->stop = 1;
00565         g_mutex_unlock (mutex);
00566         if (optimize->stop)
00567             break;

```

```

00568 #if DEBUG_OPTIMIZE
00569     fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00570 #endif
00571 }
00572 #if DEBUG_OPTIMIZE
00573     fprintf (stderr, "optimize_thread: end\n");
00574 #endif
00575     g_thread_exit (NULL);
00576     return NULL;
00577 }
00578
00590 void
00591 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00592                 double *error_best)
00593 {
00594     unsigned int i, j, k, s[optimize->nbest];
00595     double e[optimize->nbest];
00596     #if DEBUG_OPTIMIZE
00597         fprintf (stderr, "optimize_merge: start\n");
00598     #endif
00599     i = j = k = 0;
00600     do
00601     {
00602         if (i == optimize->nsaveds)
00603         {
00604             s[k] = simulation_best[j];
00605             e[k] = error_best[j];
00606             ++j;
00607             ++k;
00608             if (j == nsaveds)
00609                 break;
00610         }
00611         else if (j == nsaveds)
00612         {
00613             s[k] = optimize->simulation_best[i];
00614             e[k] = optimize->error_best[i];
00615             ++i;
00616             ++k;
00617             if (i == optimize->nsaveds)
00618                 break;
00619         }
00620         else if (optimize->error_best[i] > error_best[j])
00621         {
00622             s[k] = simulation_best[j];
00623             e[k] = error_best[j];
00624             ++j;
00625             ++k;
00626         }
00627         else
00628         {
00629             s[k] = optimize->simulation_best[i];
00630             e[k] = optimize->error_best[i];
00631             ++i;
00632             ++k;
00633         }
00634     }
00635     while (k < optimize->nbest);
00636     optimize->nsaveds = k;
00637     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00638     memcpy (optimize->error_best, e, k * sizeof (double));
00639     #if DEBUG_OPTIMIZE
00640         fprintf (stderr, "optimize_merge: end\n");
00641     #endif
00642 }
00643
00648 #if HAVE_MPI
00649 void
00650 optimize_synchronise ()
00651 {
00652     unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00653     double error_best[optimize->nbest];
00654     MPI_Status mpi_stat;
00655     #if DEBUG_OPTIMIZE
00656         fprintf (stderr, "optimize_synchronise: start\n");
00657     #endif
00658     if (optimize->mpi_rank == 0)
00659     {
00660         for (i = 1; i < ntasks; ++i)
00661         {
00662             MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00663             MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00664                      MPI_COMM_WORLD, &mpi_stat);
00665             MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00666                      MPI_COMM_WORLD, &mpi_stat);
00667             optimize_merge (nsaveds, simulation_best, error_best);
00668             MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00669             if (stop)

```



```

00670         optimize->stop = 1;
00671     }
00672     for (i = 1; i < ntasks; ++i)
00673         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00674     }
00675     else
00676     {
00677         MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00678         MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00679                 MPI_COMM_WORLD);
00680         MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00681                 MPI_COMM_WORLD);
00682         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00683         MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00684         if (stop)
00685             optimize->stop = 1;
00686     }
00687     #if DEBUG_OPTIMIZE
00688     fprintf (stderr, "optimize_synchronise: end\n");
00689     #endif
00690 }
00691 #endif
00692
00693 void
00694 optimize_sweep ()
00695 {
00696     unsigned int i, j, k, l;
00697     double e;
00698     GThread *thread[nthreads];
00699     ParallelData data[nthreads];
00700     #if DEBUG_OPTIMIZE
00701     fprintf (stderr, "optimize_sweep: start\n");
00702     #endif
00703     for (i = 0; i < optimize->nsimulations; ++i)
00704     {
00705         k = i;
00706         for (j = 0; j < optimize->nvariables; ++j)
00707         {
00708             l = k % optimize->nsweeps[j];
00709             k /= optimize->nsweeps[j];
00710             e = optimize->rangemin[j];
00711             if (optimize->nsweeps[j] > 1)
00712                 e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00713                     / (optimize->nsweeps[j] - 1);
00714             optimize->value[i * optimize->nvariables + j] = e;
00715         }
00716     }
00717     optimize->nsaveds = 0;
00718     if (nthreads <= 1)
00719         optimize_sequential ();
00720     else
00721     {
00722         for (i = 0; i < nthreads; ++i)
00723         {
00724             data[i].thread = i;
00725             thread[i] =
00726                 g_thread_new (NULL, (void (*) ) optimize_thread, &data[i]);
00727         }
00728         for (i = 0; i < nthreads; ++i)
00729             g_thread_join (thread[i]);
00730     }
00731     #if HAVE_MPI
00732     // Communicating tasks results
00733     optimize_synchronise ();
00734     #endif
00735     #if DEBUG_OPTIMIZE
00736     fprintf (stderr, "optimize_sweep: end\n");
00737     #endif
00738 }
00739
00740 void
00741 optimize_MonteCarlo ()
00742 {
00743     unsigned int i, j;
00744     GThread *thread[nthreads];
00745     ParallelData data[nthreads];
00746     #if DEBUG_OPTIMIZE
00747     fprintf (stderr, "optimize_MonteCarlo: start\n");
00748     #endif
00749     for (i = 0; i < optimize->nsimulations; ++i)
00750     {
00751         for (j = 0; j < optimize->nvariables; ++j)
00752             optimize->value[i * optimize->nvariables + j]
00753                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00754                     * (optimize->rangemax[j] - optimize->rangemin[j]);
00755         optimize->nsaveds = 0;
00756         if (nthreads <= 1)
00757             optimize_sequential ();
00758     }

```

```

00765     else
00766     {
00767         for (i = 0; i < nthreads; ++i)
00768         {
00769             data[i].thread = i;
00770             thread[i] =
00771                 g_thread_new (NULL, (void (*)(void*)) optimize_thread, &data[i]);
00772         }
00773         for (i = 0; i < nthreads; ++i)
00774             g_thread_join (thread[i]);
00775     }
00776 #if HAVE_MPI
00777     // Communicating tasks results
00778     optimize_synchronise ();
00779 #endif
00780 #if DEBUG_OPTIMIZE
00781     fprintf (stderr, "optimize_MonteCarlo: end\n");
00782 #endif
00783 }
00784
00794 void
00795 optimize_best_direction (unsigned int simulation, double value)
00796 {
00797     #if DEBUG_OPTIMIZE
00798     fprintf (stderr, "optimize_best_direction: start\n");
00799     fprintf (stderr,
00800             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00801             simulation, value, optimize->error_best[0]);
00802     #endif
00803     if (value < optimize->error_best[0])
00804     {
00805         optimize->error_best[0] = value;
00806         optimize->simulation_best[0] = simulation;
00807     #if DEBUG_OPTIMIZE
00808         fprintf (stderr,
00809                 "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00810                 simulation, value);
00811     #endif
00812     }
00813     #if DEBUG_OPTIMIZE
00814     fprintf (stderr, "optimize_best_direction: end\n");
00815     #endif
00816 }
00817
00824 void
00825 optimize_direction_sequential (unsigned int simulation)
00826 {
00827     unsigned int i, j;
00828     double e;
00829     #if DEBUG_OPTIMIZE
00830     fprintf (stderr, "optimize_direction_sequential: start\n");
00831     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00832             "nend_direction=%u\n",
00833             optimize->nstart_direction, optimize->nend_direction);
00834     #endif
00835     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00836     {
00837         j = simulation + i;
00838         e = optimize_norm (j);
00839         optimize_best_direction (j, e);
00840         optimize_save_variables (j, e);
00841         if (e < optimize->threshold)
00842         {
00843             optimize->stop = 1;
00844             break;
00845         }
00846     #if DEBUG_OPTIMIZE
00847     fprintf (stderr, "optimize_direction_sequential: i=%u e=%.14le\n", i, e);
00848     #endif
00849     }
00850     #if DEBUG_OPTIMIZE
00851     fprintf (stderr, "optimize_direction_sequential: end\n");
00852     #endif
00853 }
00854
00862 void *
00863 optimize_direction_thread (ParallelData * data)
00864 {
00865     unsigned int i, thread;
00866     double e;
00867     #if DEBUG_OPTIMIZE
00868     fprintf (stderr, "optimize_direction_thread: start\n");
00869     #endif
00870     thread = data->thread;
00871     #if DEBUG_OPTIMIZE
00872     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00873             thread,

```

```

00874         optimize->thread_direction[thread],
00875         optimize->thread_direction[thread + 1]);
00876 #endif
00877     for (i = optimize->thread_direction[thread];
00878          i < optimize->thread_direction[thread + 1]; ++i)
00879     {
00880         e = optimize_norm (i);
00881         g_mutex_lock (mutex);
00882         optimize_best_direction (i, e);
00883         optimize_save_variables (i, e);
00884         if (e < optimize->threshold)
00885             optimize->stop = 1;
00886         g_mutex_unlock (mutex);
00887         if (optimize->stop)
00888             break;
00889 #if DEBUG_OPTIMIZE
00890         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00891 #endif
00892     }
00893 #if DEBUG_OPTIMIZE
00894     fprintf (stderr, "optimize_direction_thread: end\n");
00895 #endif
00896     g_thread_exit (NULL);
00897     return NULL;
00898 }
00899
00900 double
00901 optimize_estimate_direction_random (unsigned int variable,
00902                                    unsigned int estimate)
00903 {
00904     double x;
00905     #if DEBUG_OPTIMIZE
00906     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00907     #endif
00908     x = optimize->direction[variable]
00909         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00909     #if DEBUG_OPTIMIZE
00910     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00911             variable, x);
00912     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00913     #endif
00914     return x;
00915 }
00916
00917 double
00918 optimize_estimate_direction_coordinates (unsigned int variable,
00919                                         unsigned int estimate)
00920 {
00921     double x;
00922     #if DEBUG_OPTIMIZE
00923     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00924     #endif
00925     x = optimize->direction[variable];
00926     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00927     {
00928         if (estimate & 1)
00929             x += optimize->step[variable];
00930         else
00931             x -= optimize->step[variable];
00932     }
00933     #if DEBUG_OPTIMIZE
00934     fprintf (stderr,
00935             "optimize_estimate_direction_coordinates: direction%u=%lg\n",
00936             variable, x);
00937     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00938     #endif
00939     return x;
00940 }
00941
00942 void
00943 optimize_step_direction (unsigned int simulation)
00944 {
00945     GThread *thread[nthreads_direction];
00946     ParallelData data[nthreads_direction];
00947     unsigned int i, j, k, b;
00948     #if DEBUG_OPTIMIZE
00949     fprintf (stderr, "optimize_step_direction: start\n");
00950     #endif
00951     for (i = 0; i < optimize->nestimates; ++i)
00952     {
00953         k = (simulation + i) * optimize->nvariables;
00954         b = optimize->simulation_best[0] * optimize->nvariables;
00955         #if DEBUG_OPTIMIZE
00956         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00957                 simulation + i, optimize->simulation_best[0]);
00958         #endif
00959         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)

```

```

00985     {
00986 #if DEBUG_OPTIMIZE
00987     fprintf (stderr,
00988             "optimize_step_direction: estimate=%u best%u=%.14le\n",
00989             i, j, optimize->value[b]);
00990 #endif
00991     optimize->value[k]
00992     = optimize->value[b] + optimize_estimate_direction (j, i);
00993     optimize->value[k] = fmin (fmax (optimize->value[k],
00994                                   optimize->rangeminabs[j]),
00995                               optimize->rangemaxabs[j]);
00996 #if DEBUG_OPTIMIZE
00997     fprintf (stderr,
00998             "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00999             i, j, optimize->value[k]);
01000 #endif
01001     }
01002 }
01003 if (nthreads_direction == 1)
01004     optimize_direction_sequential (simulation);
01005 else
01006     {
01007         for (i = 0; i <= nthreads_direction; ++i)
01008         {
01009             optimize->thread_direction[i]
01010             = simulation + optimize->nstart_direction
01011             + i * (optimize->nend_direction - optimize->
01012                   nstart_direction)
01013             / nthreads_direction;
01014 #if DEBUG_OPTIMIZE
01015             fprintf (stderr,
01016                     "optimize_step_direction: i=%u thread_direction=%u\n",
01017                     i, optimize->thread_direction[i]);
01018 #endif
01019         }
01020         for (i = 0; i < nthreads_direction; ++i)
01021         {
01022             data[i].thread = i;
01023             thread[i] = g_thread_new
01024             (NULL, (void (*)) optimize_direction_thread, &data[i]);
01025         }
01026         for (i = 0; i < nthreads_direction; ++i)
01027             g_thread_join (thread[i]);
01028 #if DEBUG_OPTIMIZE
01029         fprintf (stderr, "optimize_step_direction: end\n");
01030 #endif
01031     }
01032
01033 void
01034 optimize_direction ()
01035 {
01036     unsigned int i, j, k, b, s, adjust;
01037 #if DEBUG_OPTIMIZE
01038     fprintf (stderr, "optimize_direction: start\n");
01039 #endif
01040     for (i = 0; i < optimize->nvariables; ++i)
01041         optimize->direction[i] = 0.;
01042     b = optimize->simulation_best[0] * optimize->nvariables;
01043     s = optimize->nsimulations;
01044     adjust = 1;
01045     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01046     {
01047         #if DEBUG_OPTIMIZE
01048             fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01049                     i, optimize->simulation_best[0]);
01050         #endif
01051         optimize_step_direction (s);
01052         k = optimize->simulation_best[0] * optimize->nvariables;
01053         #if DEBUG_OPTIMIZE
01054             fprintf (stderr, "optimize_direction: step=%u best=%u\n",
01055                     i, optimize->simulation_best[0]);
01056         #endif
01057         if (k == b)
01058         {
01059             if (adjust)
01060             {
01061                 for (j = 0; j < optimize->nvariables; ++j)
01062                     optimize->step[j] *= 0.5;
01063                 for (j = 0; j < optimize->nvariables; ++j)
01064                     optimize->direction[j] = 0.;
01065                 adjust = 1;
01066             }
01067             else
01068             {
01069                 for (j = 0; j < optimize->nvariables; ++j)
01070                 {
01071                     #if DEBUG_OPTIMIZE

```

```

01075         fprintf (stderr,
01076                 "optimize_direction: best%u=%.14le old%u=%.14le\n",
01077                 j, optimize->value[k + j], j, optimize->value[b + j]);
01078     #endif
01079     optimize->direction[j]
01080     = (1. - optimize->relaxation) * optimize->direction[j]
01081     + optimize->relaxation
01082     * (optimize->value[k + j] - optimize->value[b + j]);
01083     #if DEBUG_OPTIMIZE
01084     fprintf (stderr, "optimize_direction: direction%u=%.14le\n",
01085             j, optimize->direction[j]);
01086     #endif
01087     }
01088     adjust = 0;
01089     }
01090     }
01091     #if DEBUG_OPTIMIZE
01092     fprintf (stderr, "optimize_direction: end\n");
01093     #endif
01094     }
01095     }
01103     double
01104     optimize_genetic_objective (Entity * entity)
01105     {
01106     unsigned int j;
01107     double objective;
01108     char buffer[64];
01109     #if DEBUG_OPTIMIZE
01110     fprintf (stderr, "optimize_genetic_objective: start\n");
01111     #endif
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         optimize->value[entity->id * optimize->nvariables + j]
01115         = genetic_get_variable (entity, optimize->genetic_variable + j);
01116     }
01117     objective = optimize_norm (entity->id);
01118     g_mutex_lock (mutex);
01119     for (j = 0; j < optimize->nvariables; ++j)
01120     {
01121         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01122         fprintf (optimize->file_variables, buffer,
01123                 genetic_get_variable (entity, optimize->genetic_variable + j));
01124     }
01125     fprintf (optimize->file_variables, "%.14le\n", objective);
01126     g_mutex_unlock (mutex);
01127     #if DEBUG_OPTIMIZE
01128     fprintf (stderr, "optimize_genetic_objective: end\n");
01129     #endif
01130     return objective;
01131     }
01132     void
01133     optimize_genetic ()
01134     {
01135     char *best_genome;
01136     double best_objective, *best_variable;
01137     #if DEBUG_OPTIMIZE
01138     fprintf (stderr, "optimize_genetic: start\n");
01139     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01140             nthreads);
01141     fprintf (stderr,
01142             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01143             optimize->nvariables, optimize->nsimulations,
01144             optimize->niterations);
01145     fprintf (stderr,
01146             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01147             optimize->mutation_ratio, optimize->reproduction_ratio,
01148             optimize->adaptation_ratio);
01149     #endif
01150     genetic_algorithm_default (optimize->nvariables,
01151                             optimize->genetic_variable,
01152                             optimize->nsimulations,
01153                             optimize->niterations,
01154                             optimize->mutation_ratio,
01155                             optimize->reproduction_ratio,
01156                             optimize->adaptation_ratio,
01157                             optimize->seed,
01158                             optimize->threshold,
01159                             &optimize_genetic_objective,
01160                             &best_genome, &best_variable, &best_objective);
01161     #if DEBUG_OPTIMIZE
01162     fprintf (stderr, "optimize_genetic: the best\n");
01163     #endif
01164     optimize->error_old = (double *) g_malloc (sizeof (double));
01165     optimize->value_old
01166     = (double *) g_malloc (optimize->nvariables * sizeof (double));
01167     optimize->error_old[0] = best_objective;

```

```

01173     memcpy (optimize->value_old, best_variable,
01174             optimize->nvariables * sizeof (double));
01175     g_free (best_genome);
01176     g_free (best_variable);
01177     optimize_print ();
01178     #if DEBUG_OPTIMIZE
01179     fprintf (stderr, "optimize_genetic: end\n");
01180     #endif
01181 }
01182
01183 void
01184 optimize_save_old ()
01185 {
01186     unsigned int i, j;
01187     #if DEBUG_OPTIMIZE
01188     fprintf (stderr, "optimize_save_old: start\n");
01189     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01190     #endif
01191     memcpy (optimize->error_old, optimize->error_best,
01192             optimize->nbest * sizeof (double));
01193     for (i = 0; i < optimize->nbest; ++i)
01194     {
01195         j = optimize->simulation_best[i];
01196         #if DEBUG_OPTIMIZE
01197         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01198         #endif
01199         memcpy (optimize->value_old + i * optimize->nvariables,
01200                 optimize->value + j * optimize->nvariables,
01201                 optimize->nvariables * sizeof (double));
01202     }
01203     #if DEBUG_OPTIMIZE
01204     for (i = 0; i < optimize->nvariables; ++i)
01205         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01206                 i, optimize->value_old[i]);
01207     fprintf (stderr, "optimize_save_old: end\n");
01208     #endif
01209 }
01210
01211 void
01212 optimize_merge_old ()
01213 {
01214     unsigned int i, j, k;
01215     double v[optimize->nbest * optimize->nvariables], e[optimize->
01216 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->
01231 nvariables,
01232                     optimize->nvariables * sizeof (double));
01233             e[k] = *enew;
01234             ++k;
01235             ++enew;
01236             ++i;
01237         }
01238         else
01239         {
01240             memcpy (v + k * optimize->nvariables,
01241                     optimize->value_old + j * optimize->nvariables,
01242                     optimize->nvariables * sizeof (double));
01243             e[k] = *eold;
01244             ++k;
01245             ++eold;
01246             ++j;
01247         }
01248     }
01249     while (k < optimize->nbest);
01250     memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01251     memcpy (optimize->error_old, e, k * sizeof (double));
01252     #if DEBUG_OPTIMIZE
01253     fprintf (stderr, "optimize_merge_old: end\n");
01254     #endif
01255 }
01256
01257 void
01258 optimize_refine ()
01259 {

```

```

01272     unsigned int i, j;
01273     double d;
01274     #if HAVE_MPI
01275     MPI_Status mpi_stat;
01276     #endif
01277     #if DEBUG_OPTIMIZE
01278     fprintf (stderr, "optimize_refine: start\n");
01279     #endif
01280     #if HAVE_MPI
01281     if (!optimize->mpi_rank)
01282     {
01283     #endif
01284         for (j = 0; j < optimize->nvariables; ++j)
01285         {
01286             optimize->rangemin[j] = optimize->rangemax[j]
01287             = optimize->value_old[j];
01288         }
01289         for (i = 0; ++i < optimize->nbest;)
01290         {
01291             for (j = 0; j < optimize->nvariables; ++j)
01292             {
01293                 optimize->rangemin[j]
01294                 = fmin (optimize->rangemin[j],
01295                     optimize->value_old[i * optimize->nvariables + j]);
01296                 optimize->rangemax[j]
01297                 = fmax (optimize->rangemax[j],
01298                     optimize->value_old[i * optimize->nvariables + j]);
01299             }
01300         }
01301         for (j = 0; j < optimize->nvariables; ++j)
01302         {
01303             d = optimize->tolerance
01304             * (optimize->rangemax[j] - optimize->rangemin[j]);
01305             switch (optimize->algorithm)
01306             {
01307             case ALGORITHM_MONTE_CARLO:
01308                 d *= 0.5;
01309                 break;
01310             default:
01311                 if (optimize->nsweeps[j] > 1)
01312                     d /= optimize->nsweeps[j] - 1;
01313                 else
01314                     d = 0.;
01315             }
01316             optimize->rangemin[j] -= d;
01317             optimize->rangemin[j]
01318             = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01319             optimize->rangemax[j] += d;
01320             optimize->rangemax[j]
01321             = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01322             printf ("%s min=%lg max=%lg\n", optimize->label[j],
01323                 optimize->rangemin[j], optimize->rangemax[j]);
01324             fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01325                 optimize->label[j], optimize->rangemin[j],
01326                 optimize->rangemax[j]);
01327         }
01328     #if HAVE_MPI
01329         for (i = 1; i < ntasks; ++i)
01330         {
01331             MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01332                 1, MPI_COMM_WORLD);
01333             MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01334                 1, MPI_COMM_WORLD);
01335         }
01336     }
01337     else
01338     {
01339         MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01340             MPI_COMM_WORLD, &mpi_stat);
01341         MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01342             MPI_COMM_WORLD, &mpi_stat);
01343     }
01344     #endif
01345     #if DEBUG_OPTIMIZE
01346     fprintf (stderr, "optimize_refine: end\n");
01347     #endif
01348 }
01349
01350 void
01351 optimize_step ()
01352 {
01353     #if DEBUG_OPTIMIZE
01354     fprintf (stderr, "optimize_step: start\n");
01355     #endif
01356     optimize_algorithm ();
01357     if (optimize->nsteps)
01358         optimize_direction ();

```

```

01363 #if DEBUG_OPTIMIZE
01364 fprintf (stderr, "optimize_step: end\n");
01365 #endif
01366 }
01367
01372 void
01373 optimize_iterate ()
01374 {
01375     unsigned int i;
01376     #if DEBUG_OPTIMIZE
01377     fprintf (stderr, "optimize_iterate: start\n");
01378     #endif
01379     optimize->error_old =
01380         (double *) g_malloc (optimize->nbest * sizeof (double));
01381     optimize->value_old =
01382         (double *) g_malloc (optimize->nbest * optimize->nvariables *
01383                             sizeof (double));
01384     optimize_step ();
01385     optimize_save_old ();
01386     optimize_refine ();
01387     optimize_print ();
01388     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01389     {
01390         optimize_step ();
01391         optimize_merge_old ();
01392         optimize_refine ();
01393         optimize_print ();
01394     }
01395     #if DEBUG_OPTIMIZE
01396     fprintf (stderr, "optimize_iterate: end\n");
01397     #endif
01398 }
01399
01404 void
01405 optimize_free ()
01406 {
01407     unsigned int i, j;
01408     #if DEBUG_OPTIMIZE
01409     fprintf (stderr, "optimize_free: start\n");
01410     #endif
01411     for (j = 0; j < optimize->ninputs; ++j)
01412     {
01413         for (i = 0; i < optimize->nexperiments; ++i)
01414             g_mapped_file_unref (optimize->file[j][i]);
01415         g_free (optimize->file[j]);
01416     }
01417     g_free (optimize->error_old);
01418     g_free (optimize->value_old);
01419     g_free (optimize->value);
01420     g_free (optimize->genetic_variable);
01421     #if DEBUG_OPTIMIZE
01422     fprintf (stderr, "optimize_free: end\n");
01423     #endif
01424 }
01425
01430 void
01431 optimize_open ()
01432 {
01433     GTimeZone *tz;
01434     GDateTime *t0, *t;
01435     unsigned int i, j;
01436
01437     #if DEBUG_OPTIMIZE
01438     char *buffer;
01439     fprintf (stderr, "optimize_open: start\n");
01440     #endif
01441
01442     // Getting initial time
01443     #if DEBUG_OPTIMIZE
01444     fprintf (stderr, "optimize_open: getting initial time\n");
01445     #endif
01446     tz = g_time_zone_new_utc ();
01447     t0 = g_date_time_new_now (tz);
01448
01449     // Obtaining and initing the pseudo-random numbers generator seed
01450     #if DEBUG_OPTIMIZE
01451     fprintf (stderr, "optimize_open: getting initial seed\n");
01452     #endif
01453     if (optimize->seed == DEFAULT_RANDOM_SEED)
01454         optimize->seed = input->seed;
01455     gsl_rng_set (optimize->rng, optimize->seed);
01456
01457     // Replacing the working directory
01458     #if DEBUG_OPTIMIZE
01459     fprintf (stderr, "optimize_open: replacing the working directory\n");
01460     #endif
01461     g_chdir (input->directory);

```



```

01462
01463 // Getting results file names
01464 optimize->result = input->result;
01465 optimize->variables = input->variables;
01466
01467 // Obtaining the simulator file
01468 optimize->simulator = input->simulator;
01469
01470 // Obtaining the evaluator file
01471 optimize->evaluator = input->evaluator;
01472
01473 // Reading the algorithm
01474 optimize->algorithm = input->algorithm;
01475 switch (optimize->algorithm)
01476 {
01477     case ALGORITHM_MONTE_CARLO:
01478         optimize_algorithm = optimize_MonteCarlo;
01479         break;
01480     case ALGORITHM_SWEEP:
01481         optimize_algorithm = optimize_sweep;
01482         break;
01483     default:
01484         optimize_algorithm = optimize_genetic;
01485         optimize->mutation_ratio = input->mutation_ratio;
01486         optimize->reproduction_ratio = input->
reproduction_ratio;
01487         optimize->adaptation_ratio = input->adaptation_ratio;
01488     }
01489 optimize->nvariables = input->nvariables;
01490 optimize->nsimulations = input->nsimulations;
01491 optimize->niterations = input->niterations;
01492 optimize->nbest = input->nbest;
01493 optimize->tolerance = input->tolerance;
01494 optimize->nsteps = input->nsteps;
01495 optimize->nestimates = 0;
01496 optimize->threshold = input->threshold;
01497 optimize->stop = 0;
01498 if (input->nsteps)
01499 {
01500     optimize->relaxation = input->relaxation;
01501     switch (input->direction)
01502     {
01503         case DIRECTION_METHOD_COORDINATES:
01504             optimize->nestimates = 2 * optimize->nvariables;
01505             optimize_estimate_direction =
optimize_estimate_direction_coordinates;
01506             break;
01507         default:
01508             optimize->nestimates = input->nestimates;
01509             optimize_estimate_direction =
optimize_estimate_direction_random;
01510     }
01511 }
01512 }
01513
01514 #if DEBUG_OPTIMIZE
01515 fprintf (stderr, "optimize_open: nbest=%u\n", optimize->nbest);
01516 #endif
01517 optimize->simulation_best
01518 = (unsigned int *) alloca (optimize->nbest * sizeof (unsigned int));
01519 optimize->error_best =
01520 (double *) alloca (optimize->nbest * sizeof (double));
01521
01522 // Reading the experimental data
01523 #if DEBUG_OPTIMIZE
01524 buffer = g_get_current_dir ();
01525 fprintf (stderr, "optimize_open: current directory=%s\n", buffer);
01526 g_free (buffer);
01527 #endif
01528 optimize->nexperiments = input->nexperiments;
01529 optimize->ninputs = input->experiment->ninputs;
01530 optimize->experiment
01531 = (char **) alloca (input->nexperiments * sizeof (char *));
01532 optimize->weight =
01533 (double *) alloca (input->nexperiments * sizeof (double));
01534 for (i = 0; i < input->experiment->ninputs; ++i)
01535     optimize->file[i] = (GMappedFile **)
01536 g_malloc (input->nexperiments * sizeof (GMappedFile *));
01537 for (i = 0; i < input->nexperiments; ++i)
01538 {
01539 #if DEBUG_OPTIMIZE
01540     fprintf (stderr, "optimize_open: i=%u\n", i);
01541 #endif
01542     optimize->experiment[i] = input->experiment[i].
name;
01543     optimize->weight[i] = input->experiment[i].weight;
01544 #if DEBUG_OPTIMIZE
01545     fprintf (stderr, "optimize_open: experiment=%s weight=%lg\n",

```

```

01546         optimize->experiment[i], optimize->weight[i]);
01547 #endif
01548     for (j = 0; j < input->experiment->ninputs; ++j)
01549     {
01550 #if DEBUG_OPTIMIZE
01551         fprintf(stderr, "optimize_open: template%u\n", j + 1);
01552 #endif
01553         optimize->file[j][i]
01554             = g_mapped_file_new (input->experiment[i].template[j], 0, NULL);
01555     }
01556 }
01557
01558 // Reading the variables data
01559 #if DEBUG_OPTIMIZE
01560 fprintf(stderr, "optimize_open: reading variables\n");
01561 #endif
01562 optimize->label = (char **) alloca (input->nvariables * sizeof (char *));
01563 j = input->nvariables * sizeof (double);
01564 optimize->rangemin = (double *) alloca (j);
01565 optimize->rangeminabs = (double *) alloca (j);
01566 optimize->rangemax = (double *) alloca (j);
01567 optimize->rangemaxabs = (double *) alloca (j);
01568 optimize->step = (double *) alloca (j);
01569 j = input->nvariables * sizeof (unsigned int);
01570 optimize->precision = (unsigned int *) alloca (j);
01571 optimize->nsweeps = (unsigned int *) alloca (j);
01572 optimize->nbits = (unsigned int *) alloca (j);
01573 for (i = 0; i < input->nvariables; ++i)
01574 {
01575     optimize->label[i] = input->variable[i].name;
01576     optimize->rangemin[i] = input->variable[i].rangemin;
01577     optimize->rangeminabs[i] = input->variable[i].
rangeminabs;
01578     optimize->rangemax[i] = input->variable[i].rangemax;
01579     optimize->rangemaxabs[i] = input->variable[i].
rangemaxabs;
01580     optimize->precision[i] = input->variable[i].
precision;
01581     optimize->step[i] = input->variable[i].step;
01582     optimize->nsweeps[i] = input->variable[i].nsweeps;
01583     optimize->nbits[i] = input->variable[i].nbits;
01584 }
01585 if (input->algorithm == ALGORITHM_SWEEP)
01586 {
01587     optimize->nsimulations = 1;
01588     for (i = 0; i < input->nvariables; ++i)
01589     {
01590         if (input->algorithm == ALGORITHM_SWEEP)
01591         {
01592             optimize->nsimulations *= optimize->nsweeps[i];
01593 #if DEBUG_OPTIMIZE
01594             fprintf(stderr, "optimize_open: nsweeps=%u nsimulations=%u\n",
01595                 optimize->nsweeps[i], optimize->nsimulations);
01596 #endif
01597         }
01598     }
01599 }
01600 if (optimize->nsteps)
01601     optimize->direction
01602         = (double *) alloca (optimize->nvariables * sizeof (double));
01603
01604 // Setting error norm
01605 switch (input->norm)
01606 {
01607     case ERROR_NORM_EUCLIDIAN:
01608         optimize_norm = optimize_norm_euclidian;
01609         break;
01610     case ERROR_NORM_MAXIMUM:
01611         optimize_norm = optimize_norm_maximum;
01612         break;
01613     case ERROR_NORM_P:
01614         optimize_norm = optimize_norm_p;
01615         optimize->p = input->p;
01616         break;
01617     default:
01618         optimize_norm = optimize_norm_taxicab;
01619 }
01620
01621 // Allocating values
01622 #if DEBUG_OPTIMIZE
01623 fprintf(stderr, "optimize_open: allocating variables\n");
01624 fprintf(stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01625     optimize->nvariables, optimize->algorithm);
01626 #endif
01627 optimize->genetic_variable = NULL;
01628 if (optimize->algorithm == ALGORITHM_GENETIC)
01629 {

```

```

01630     optimize->genetic_variable = (GeneticVariable *)
01631     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01632     for (i = 0; i < optimize->nvariables; ++i)
01633     {
01634     #if DEBUG_OPTIMIZE
01635         fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01636             i, optimize->rangemin[i], optimize->rangemax[i],
01637             optimize->nbits[i]);
01638     #endif
01639         optimize->genetic_variable[i].minimum = optimize->
01640         rangemin[i];
01641         optimize->genetic_variable[i].maximum = optimize->
01642         rangemax[i];
01643         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01644     }
01645     #if DEBUG_OPTIMIZE
01646     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01647         optimize->nvariables, optimize->nsimulations);
01648     #endif
01649     optimize->value = (double *)
01650     g_malloc ((optimize->nsimulations
01651         + optimize->nestimates * optimize->nsteps)
01652         * optimize->nvariables * sizeof (double));
01653     // Calculating simulations to perform for each task
01654     #if HAVE_MPI
01655     #if DEBUG_OPTIMIZE
01656     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01657         optimize->mpi_rank, ntasks);
01658     #endif
01659     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01660     ntasks;
01661     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01662     ntasks;
01663     if (optimize->nsteps)
01664     {
01665         optimize->nstart_direction
01666         = optimize->mpi_rank * optimize->nestimates / ntasks;
01667         optimize->nend_direction
01668         = (1 + optimize->mpi_rank) * optimize->nestimates /
01669         ntasks;
01670     }
01671     #else
01672     optimize->nstart = 0;
01673     optimize->nend = optimize->nsimulations;
01674     if (optimize->nsteps)
01675     {
01676         optimize->nstart_direction = 0;
01677         optimize->nend_direction = optimize->nestimates;
01678     }
01679     #endif
01680     #if DEBUG_OPTIMIZE
01681     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01682         optimize->nend);
01683     #endif
01684     // Calculating simulations to perform for each thread
01685     optimize->thread = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01686     for (i = 0; i <= nthreads; ++i)
01687     {
01688         optimize->thread[i] = optimize->nstart
01689         + i * (optimize->nend - optimize->nstart) / nthreads;
01690     }
01691     #if DEBUG_OPTIMIZE
01692     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01693         optimize->thread[i]);
01694     #endif
01695     if (optimize->nsteps)
01696     {
01697         optimize->thread_direction = (unsigned int *)
01698         alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01699     }
01700     // Opening result files
01701     optimize->file_result = g_fopen (optimize->result, "w");
01702     optimize->file_variables = g_fopen (optimize->variables, "w");
01703     // Performing the algorithm
01704     switch (optimize->algorithm)
01705     {
01706     // Genetic algorithm
01707     case ALGORITHM_GENETIC:
01708         optimize_genetic ();
01709         break;
01710     // Iterative algorithm
01711     default:

```

```

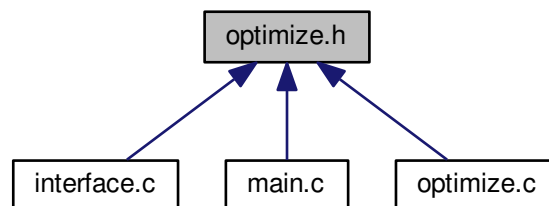
01712     optimize_iterate ();
01713 }
01714
01715 // Getting calculation time
01716 t = g_date_time_new_now (tz);
01717 optimize->calculation_time = 0.000001 * g_date_time_difference (t, t0);
01718 g_date_time_unref (t);
01719 g_date_time_unref (t0);
01720 g_time_zone_unref (tz);
01721 printf ("%s = %.6lg s\n",
01722         _("Calculation time"), optimize->calculation_time);
01723 fprintf (optimize->file_result, "%s = %.6lg s\n",
01724         _("Calculation time"), optimize->calculation_time);
01725
01726 // Closing result files
01727 fclose (optimize->file_variables);
01728 fclose (optimize->file_result);
01729
01730 #if DEBUG_OPTIMIZE
01731 fprintf (stderr, "optimize_open: end\n");
01732 #endif
01733 }

```

## 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 468 of file `optimize.c`.

```

00469 {
00470     unsigned int i, j;
00471     double e;
00472     #if DEBUG_OPTIMIZE
00473     fprintf (stderr, "optimize_best: start\n");
00474     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00475             optimize->nsaveds, optimize->nbest);
00476     #endif
00477     if (optimize->nsaveds < optimize->nbest
00478         || value < optimize->error_best[optimize->nsaveds - 1])
00479     {
00480         if (optimize->nsaveds < optimize->nbest)
00481             ++optimize->nsaveds;
00482         optimize->error_best[optimize->nsaveds - 1] = value;
00483         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00484         for (i = optimize->nsaveds; --i;)
00485         {
00486             if (optimize->error_best[i] < optimize->
00487                 error_best[i - 1])
00488             {
00489                 j = optimize->simulation_best[i];
00490                 e = optimize->error_best[i];
00491                 optimize->simulation_best[i] = optimize->
00492                     simulation_best[i - 1];
00493                 optimize->error_best[i] = optimize->
00494                     error_best[i - 1];
00495                 optimize->simulation_best[i - 1] = j;
00496                 optimize->error_best[i - 1] = e;
00497             }
00498             else
00499                 break;
00500         }
00501     }
00502     #if DEBUG_OPTIMIZE
00503     fprintf (stderr, "optimize_best: end\n");
00504     #endif
00505 }
```

## 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 795 of file `optimize.c`.

```

00796 {
00797     #if DEBUG_OPTIMIZE
00798     fprintf (stderr, "optimize_best_direction: start\n");
00799     fprintf (stderr,
00800             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
```

```

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

```

### 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 825 of file [optimize.c](#).

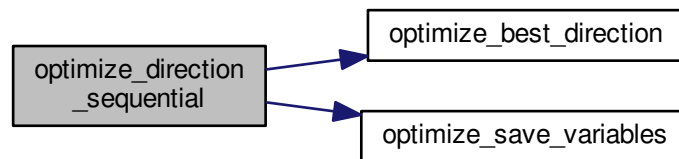
```

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

```



Here is the call graph for this function:



#### 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 863 of file [optimize.c](#).

```

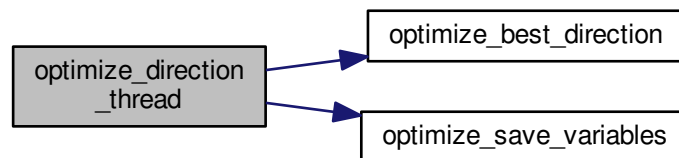
00864 {
00865     unsigned int i, thread;
00866     double e;
00867     #if DEBUG_OPTIMIZE
00868         fprintf (stderr, "optimize_direction_thread: start\n");
00869     #endif
00870     thread = data->thread;
00871     #if DEBUG_OPTIMIZE
00872         fprintf (stderr, "optimize_direction_thread: thread=%u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

```

00895 #endif
00896     g_thread_exit (NULL);
00897     return NULL;
00898 }

```

Here is the call graph for this function:



#### 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 937 of file [optimize.c](#).

```

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

```

## 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 910 of file [optimize.c](#).

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

## 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 1104 of file [optimize.c](#).

```
01105 {
01106     unsigned int j;
01107     double objective;
01108     char buffer[64];
01109     #if DEBUG_OPTIMIZE
01110     fprintf (stderr, "optimize_genetic_objective: start\n");
01111     #endif
01112     for (j = 0; j < optimize->nvariables; ++j)
01113     {
01114         optimize->value[entity->id * optimize->nvariables + j]
```

```

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

```

### 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,
00125         content);
00126     #endif
00127     file = g_fopen (input, "w");
00128
00129     // Parsing template
00130     for (i = 0; i < optimize->nvariables; ++i)
00131     {
00132         #if DEBUG_OPTIMIZE
00133         fprintf (stderr, "optimize_input: variable=%u\n", i);
00134         #endif
00135         snprintf (buffer, 32, "@variable%u@", i + 1);
00136         regex = g_regex_new (buffer, 0, 0, NULL);
00137         if (i == 0)
00138         {
00139             buffer2 = g_regex_replace_literal (regex, content, length, 0,

```

```

00140                                     optimize->label[i], 0, NULL);
00141 #if DEBUG_OPTIMIZE
00142     fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00143 #endif
00144 }
00145 else
00146 {
00147     length = strlen (buffer3);
00148     buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00149                                     optimize->label[i], 0, NULL);
00150     g_free (buffer3);
00151 }
00152 g_regex_unref (regex);
00153 length = strlen (buffer2);
00154 snprintf (buffer, 32, "@value%u@", i + 1);
00155 regex = g_regex_new (buffer, 0, 0, NULL);
00156 snprintf (value, 32, format[optimize->precision[i]],
00157         optimize->value[simulation * optimize->
nvariables + i]);
00158
00159 #if DEBUG_OPTIMIZE
00160     fprintf (stderr, "optimize_input: value=%s\n", value);
00161 #endif
00162 buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00163                                 0, NULL);
00164 g_free (buffer2);
00165 g_regex_unref (regex);
00166 }
00167
00168 // Saving input file
00169 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00170 g_free (buffer3);
00171 fclose (file);
00172
00173 optimize_input_end:
00174 #if DEBUG_OPTIMIZE
00175     fprintf (stderr, "optimize_input: end\n");
00176 #endif
00177     return;
00178 }

```

### 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 591 of file [optimize.c](#).

```

00593 {
00594     unsigned int i, j, k, s[optimize->nbest];
00595     double e[optimize->nbest];
00596     #if DEBUG_OPTIMIZE
00597         fprintf (stderr, "optimize_merge: start\n");
00598     #endif
00599     i = j = k = 0;
00600     do
00601     {
00602         if (i == optimize->nsaveds)
00603         {
00604             s[k] = simulation_best[j];

```

```

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

```

#### 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 300 of file [optimize.c](#).

```

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

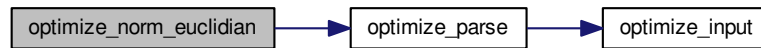
```

```

00315     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00316     fprintf (stderr, "optimize_norm_euclidian: end\n");
00317 #endif
00318     return e;
00319 }

```

Here is the call graph for this function:



#### 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.

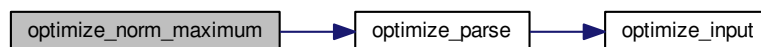
Definition at line 329 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:



### 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.
-------------------	--------------------

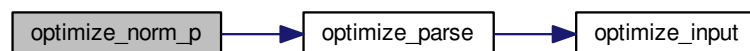
#### Returns

P error norm.

Definition at line [357](#) of file [optimize.c](#).

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

Here is the call graph for this function:



### 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**

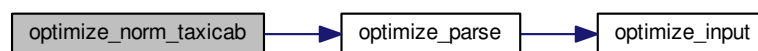
Taxicab error norm.

Definition at line 386 of file [optimize.c](#).

```

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

Here is the call graph for this function:

**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>	<a href="#">Experiment</a> number.

**Returns**

Objective function value.

Definition at line 191 of file [optimize.c](#).

```

00192 {
00193     unsigned int i;
00194     double e;
00195     char buffer[512], input[MAX\_NINPUTS][32], output[32], result[32], *buffer2,
00196         *buffer3, *buffer4;
00197     FILE *file_result;
```

```

00198
00199 #if DEBUG_OPTIMIZE
00200 fprintf (stderr, "optimize_parse: start\n");
00201 fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n",
00202         simulation, experiment);
00203 #endif
00204
00205 // Opening input files
00206 for (i = 0; i < optimize->ninputs; ++i)
00207 {
00208     snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation,
00209             experiment);
00210 #if DEBUG_OPTIMIZE
00211     fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00212 #endif
00213     optimize_input (simulation, &input[i][0],
00214             optimize->file[i][experiment]);
00215 }
00216 for (; i < MAX_NINPUTS; ++i)
00217     strcpy (&input[i][0], "");
00218 #if DEBUG_OPTIMIZE
00219 fprintf (stderr, "optimize_parse: parsing end\n");
00220 #endif
00221
00222 // Performing the simulation
00223 snprintf (output, 32, "output-%u-%u", simulation, experiment);
00224 buffer2 = g_path_get_dirname (optimize->simulator);
00225 buffer3 = g_path_get_basename (optimize->simulator);
00226 buffer4 = g_build_filename (buffer2, buffer3, NULL);
00227 snprintf (buffer, 512, "%s\ " %s %s %s %s %s %s %s %s %s",
00228         buffer4, input[0], input[1], input[2], input[3], input[4],
00229         input[5], input[6], input[7], output);
00230 g_free (buffer4);
00231 g_free (buffer3);
00232 g_free (buffer2);
00233 #if DEBUG_OPTIMIZE
00234 fprintf (stderr, "optimize_parse: %s\n", buffer);
00235 #endif
00236 system (buffer);
00237
00238 // Checking the objective value function
00239 if (optimize->evaluator)
00240 {
00241     snprintf (result, 32, "result-%u-%u", simulation, experiment);
00242     buffer2 = g_path_get_dirname (optimize->evaluator);
00243     buffer3 = g_path_get_basename (optimize->evaluator);
00244     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00245     snprintf (buffer, 512, "%s\ " %s %s %s",
00246             buffer4, output, optimize->experiment[experiment], result);
00247     g_free (buffer4);
00248     g_free (buffer3);
00249     g_free (buffer2);
00250 #if DEBUG_OPTIMIZE
00251     fprintf (stderr, "optimize_parse: %s\n", buffer);
00252 #endif
00253     system (buffer);
00254     file_result = g_fopen (result, "r");
00255     e = atof (fgets (buffer, 512, file_result));
00256     fclose (file_result);
00257 }
00258 else
00259 {
00260     strcpy (result, "");
00261     file_result = g_fopen (output, "r");
00262     e = atof (fgets (buffer, 512, file_result));
00263     fclose (file_result);
00264 }
00265
00266 // Removing files
00267 #if !DEBUG_OPTIMIZE
00268 for (i = 0; i < optimize->ninputs; ++i)
00269 {
00270     if (optimize->file[i][0])
00271     {
00272         snprintf (buffer, 512, RM " %s", &input[i][0]);
00273         system (buffer);
00274     }
00275 }
00276     snprintf (buffer, 512, RM " %s %s", output, result);
00277     system (buffer);
00278 #endif
00279
00280 // Processing pending events
00281 if (show_pending)
00282     show_pending ();
00283
00284 #if DEBUG_OPTIMIZE

```

```

00285     fprintf (stderr, "optimize_parse: end\n");
00286 #endif
00287
00288     // Returning the objective function
00289     return e * optimize->weight[experiment];
00290 }

```

Here is the call graph for this function:



#### 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 439 of file [optimize.c](#).

```

00440 {
00441     unsigned int i;
00442     char buffer[64];
00443     #if DEBUG_OPTIMIZE
00444     fprintf (stderr, "optimize_save_variables: start\n");
00445     #endif
00446     for (i = 0; i < optimize->nvariables; ++i)
00447     {
00448         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00449         fprintf (optimize->file_variables, buffer,
00450             optimize->value[simulation * optimize->
00451                 nvariables + i]);
00452     }
00453     fprintf (optimize->file_variables, "%.14le\n", error);
00454     fflush (optimize->file_variables);
00455     #if DEBUG_OPTIMIZE
00456     fprintf (stderr, "optimize_save_variables: end\n");
00457     #endif
00458 }

```

#### 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.
-------------------	--------------------

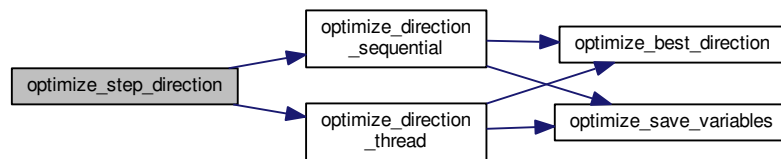
Definition at line 968 of file `optimize.c`.

```

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

```

Here is the call graph for this function:



#### 5.19.2.17 optimize\_thread()

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

Function to optimize on a thread.

##### Parameters

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

##### Returns

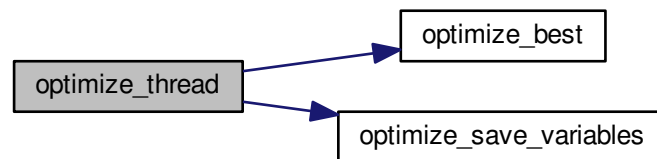
NULL

Definition at line 545 of file [optimize.c](#).

```

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

Here is the call graph for this function:



## 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

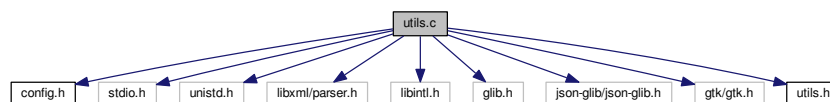
Source file to define some useful functions.

```

#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\\_message](#) (char \*title, char \*msg, int type)  
*Function to show a dialog with a message.*
- void [show\\_error](#) (char \*msg)  
*Function to show a dialog with an error message.*
- int [xml\\_node\\_get\\_int](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an integer number of a XML node property.*
- unsigned int [xml\\_node\\_get\\_uint](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get an unsigned integer number of a XML node property.*
- unsigned int [xml\\_node\\_get\\_uint\\_with\\_default](#) (xmlNode \*node, const xmlChar \*prop, unsigned int default\_value, int \*error\_code)  
*Function to get an unsigned integer number of a XML node property with a default value.*
- double [xml\\_node\\_get\\_float](#) (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
*Function to get a floating point number of a XML node property.*
- double [xml\\_node\\_get\\_float\\_with\\_default](#) (xmlNode \*node, const xmlChar \*prop, double default\_value, int \*error\_code)  
*Function to get a floating point number of a XML node property with a default value.*
- void [xml\\_node\\_set\\_int](#) (xmlNode \*node, const xmlChar \*prop, int value)  
*Function to set an integer number in a XML node property.*
- void [xml\\_node\\_set\\_uint](#) (xmlNode \*node, const xmlChar \*prop, unsigned int value)  
*Function to set an unsigned integer number in a XML node property.*
- void [xml\\_node\\_set\\_float](#) (xmlNode \*node, const xmlChar \*prop, double value)



- Function to set a floating point number in a XML node property.*
- int [json\\_object\\_get\\_int](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get an integer number of a JSON object property.*
- unsigned int [json\\_object\\_get\\_uint](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get an unsigned integer number of a JSON object property.*
- unsigned int [json\\_object\\_get\\_uint\\_with\\_default](#) (JsonObject \*object, const char \*prop, unsigned int default\_value, int \*error\_code)
- Function to get an unsigned integer number of a JSON object property with a default value.*
- double [json\\_object\\_get\\_float](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get a floating point number of a JSON object property.*
- double [json\\_object\\_get\\_float\\_with\\_default](#) (JsonObject \*object, const char \*prop, double default\_value, int \*error\_code)
- Function to get a floating point number of a JSON object property with a default value.*
- void [json\\_object\\_set\\_int](#) (JsonObject \*object, const char \*prop, int value)
- Function to set an integer number in a JSON object property.*
- void [json\\_object\\_set\\_uint](#) (JsonObject \*object, const char \*prop, unsigned int value)
- Function to set an unsigned integer number in a JSON object property.*
- void [json\\_object\\_set\\_float](#) (JsonObject \*object, const char \*prop, double value)
- Function to set a floating point number in a JSON object property.*
- int [cores\\_number](#) ()
- Function to obtain the cores number.*
- void [process\\_pending](#) ()
- Function to process events on long computation.*
- unsigned int [gtk\\_array\\_get\\_active](#) (GtkRadioButton \*array[], unsigned int n)
- Function to get the active GtkRadioButton.*

## Variables

- GtkWidget \* [main\\_window](#)
- Main GtkWidget.*
- char \* [error\\_message](#)
- Error message.*
- void(\* [show\\_pending](#) )() = NULL
- Pointer to the function to show pending events.*

### 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 532 of file [utils.c](#).

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

### 5.21.2.2 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 567 of file [utils.c](#).

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

### 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 421 of file [utils.c](#).

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

### 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**

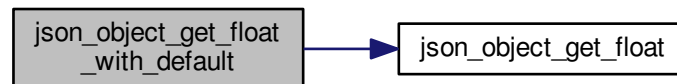
Floating point number value.

Definition at line [454](#) of file [utils.c](#).

```

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

Here is the call graph for this function:

**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 [330](#) of file [utils.c](#).

```

00331 {
00332     const char *buffer;
00333     int i = 0;
```

```

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

```

### 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 360 of file [utils.c](#).

```

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

```

### 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.

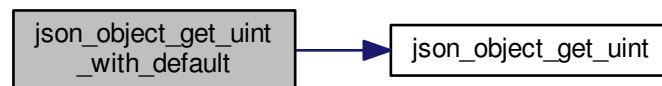
## Returns

Unsigned integer number value.

Definition at line 393 of file [utils.c](#).

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

Here is the call graph for this function:



### 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 519 of file [utils.c](#).

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

#### 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 480 of file [utils.c](#).

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

#### 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 499 of file [utils.c](#).

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

### 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 103 of file [utils.c](#).

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

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 73 of file [utils.c](#).

```
00074 {
00075     #if HAVE_GTK
00076     GtkMessageDialog *dlg;
00077
00078     // Creating the dialog
```



```

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

```

### 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 213 of file [utils.c](#).

```

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

```

### 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.

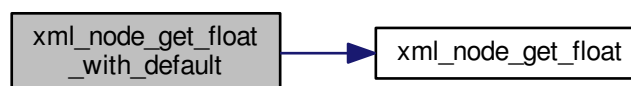
**Returns**

Floating point number value.

Definition at line 247 of file [utils.c](#).

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

Here is the call graph for this function:

**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 121 of file [utils.c](#).

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

### 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 152 of file [utils.c](#).

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

### 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.

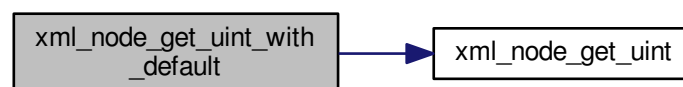
#### Returns

Unsigned integer number value.

Definition at line 186 of file [utils.c](#).

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

Here is the call graph for this function:



### 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 310 of file [utils.c](#).

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

#### 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 272 of file [utils.c](#).

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

#### 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 291 of file [utils.c](#).

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

## 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 void (*show_pending) () = NULL;
00054
00055 void
00056 show_message (char *title, char *msg, int type)
00057 {
00058     #if HAVE_GTK
00059         GtkMessageDialog *dlg;
00060
00061         // Creating the dialog
00062         dlg = (GtkMessageDialog *) gtk_message_dialog_new
00063             (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00064
00065         // Setting the dialog title
00066         gtk_window_set_title (GTK_WINDOW (dlg), title);
00067
00068         // Showing the dialog and waiting response
00069         gtk_dialog_run (GTK_DIALOG (dlg));
00070     #endif
00071 }
```

```

00088 // Closing and freeing memory
00089 gtk_widget_destroy (GTK_WIDGET (dlg));
00090
00091 #else
00092     printf ("%s: %s\n", title, msg);
00093 #endif
00094 }
00095
00102 void
00103 show_error (char *msg)
00104 {
00105     show_message (_("ERROR!"), msg, ERROR_TYPE);
00106 }
00107
00120 int
00121 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00122 {
00123     int i = 0;
00124     xmlChar *buffer;
00125     buffer = xmlGetProp (node, prop);
00126     if (!buffer)
00127         *error_code = 1;
00128     else
00129     {
00130         if (sscanf ((char *) buffer, "%d", &i) != 1)
00131             *error_code = 2;
00132         else
00133             *error_code = 0;
00134         xmlFree (buffer);
00135     }
00136     return i;
00137 }
00138
00151 unsigned int
00152 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00153 {
00154     unsigned int i = 0;
00155     xmlChar *buffer;
00156     buffer = xmlGetProp (node, prop);
00157     if (!buffer)
00158         *error_code = 1;
00159     else
00160     {
00161         if (sscanf ((char *) buffer, "%u", &i) != 1)
00162             *error_code = 2;
00163         else
00164             *error_code = 0;
00165         xmlFree (buffer);
00166     }
00167     return i;
00168 }
00169
00185 unsigned int
00186 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00187                                unsigned int default_value, int *error_code)
00188 {
00189     unsigned int i;
00190     if (xmlHasProp (node, prop))
00191         i = xml_node_get_uint (node, prop, error_code);
00192     else
00193     {
00194         i = default_value;
00195         *error_code = 0;
00196     }
00197     return i;
00198 }
00199
00212 double
00213 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00214 {
00215     double x = 0.;
00216     xmlChar *buffer;
00217     buffer = xmlGetProp (node, prop);
00218     if (!buffer)
00219         *error_code = 1;
00220     else
00221     {
00222         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00223             *error_code = 2;
00224         else
00225             *error_code = 0;
00226         xmlFree (buffer);
00227     }
00228     return x;
00229 }
00230
00246 double

```

```

00247 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00248                                   double default_value, int *error_code)
00249 {
00250     double x;
00251     if (xmlHasProp (node, prop))
00252         x = xml_node_get_float (node, prop, error_code);
00253     else
00254     {
00255         x = default_value;
00256         *error_code = 0;
00257     }
00258     return x;
00259 }
00260
00271 void
00272 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00273 {
00274     xmlChar buffer[64];
00275     snprintf ((char *) buffer, 64, "%d", value);
00276     xmlSetProp (node, prop, buffer);
00277 }
00278
00290 void
00291 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00292 {
00293     xmlChar buffer[64];
00294     snprintf ((char *) buffer, 64, "%u", value);
00295     xmlSetProp (node, prop, buffer);
00296 }
00297
00309 void
00310 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00311 {
00312     xmlChar buffer[64];
00313     snprintf ((char *) buffer, 64, "%.14lg", value);
00314     xmlSetProp (node, prop, buffer);
00315 }
00316
00329 int
00330 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00331 {
00332     const char *buffer;
00333     int i = 0;
00334     buffer = json_object_get_string_member (object, prop);
00335     if (!buffer)
00336         *error_code = 1;
00337     else
00338     {
00339         if (sscanf (buffer, "%d", &i) != 1)
00340             *error_code = 2;
00341         else
00342             *error_code = 0;
00343     }
00344     return i;
00345 }
00346
00359 unsigned int
00360 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00361 {
00362     const char *buffer;
00363     unsigned int i = 0;
00364     buffer = json_object_get_string_member (object, prop);
00365     if (!buffer)
00366         *error_code = 1;
00367     else
00368     {
00369         if (sscanf (buffer, "%u", &i) != 1)
00370             *error_code = 2;
00371         else
00372             *error_code = 0;
00373     }
00374     return i;
00375 }
00376
00392 unsigned int
00393 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00394                                   unsigned int default_value,
00395                                   int *error_code)
00396 {
00397     unsigned int i;
00398     if (json_object_get_member (object, prop))
00399         i = json_object_get_uint (object, prop, error_code);
00400     else
00401     {
00402         i = default_value;
00403         *error_code = 0;
00404     }

```



```
00405     return i;
00406 }
00407
00420 double
00421 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00422 {
00423     const char *buffer;
00424     double x = 0.;
00425     buffer = json_object_get_string_member (object, prop);
00426     if (!buffer)
00427         *error_code = 1;
00428     else
00429     {
00430         if (sscanf (buffer, "%lf", &x) != 1)
00431             *error_code = 2;
00432         else
00433             *error_code = 0;
00434     }
00435     return x;
00436 }
00437
00453 double
00454 json_object_get_float_with_default (JsonObject * object, const char *prop
00455 ,
                                double default_value, int *error_code)
00456 {
00457     double x;
00458     if (json_object_get_member (object, prop))
00459         x = json_object_get_float (object, prop, error_code);
00460     else
00461     {
00462         x = default_value;
00463         *error_code = 0;
00464     }
00465     return x;
00466 }
00467
00479 void
00480 json_object_set_int (JsonObject * object, const char *prop, int value)
00481 {
00482     char buffer[64];
00483     snprintf (buffer, 64, "%d", value);
00484     json_object_set_string_member (object, prop, buffer);
00485 }
00486
00498 void
00499 json_object_set_uint (JsonObject * object, const char *prop,
00500                     unsigned int value)
00501 {
00502     char buffer[64];
00503     snprintf (buffer, 64, "%u", value);
00504     json_object_set_string_member (object, prop, buffer);
00505 }
00506
00518 void
00519 json_object_set_float (JsonObject * object, const char *prop, double value)
00520 {
00521     char buffer[64];
00522     snprintf (buffer, 64, "%.14lg", value);
00523     json_object_set_string_member (object, prop, buffer);
00524 }
00525
00531 int
00532 cores_number ()
00533 {
00534     #ifdef G_OS_WIN32
00535         SYSTEM_INFO sysinfo;
00536         GetSystemInfo (&sysinfo);
00537         return sysinfo.dwNumberOfProcessors;
00538     #else
00539         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00540     #endif
00541 }
00542
00543 #if HAVE_GTK
00544
00549 void
00550 process_pending ()
00551 {
00552     while (gtk_events_pending ())
00553         gtk_main_iteration ();
00554 }
00555
00566 unsigned int
00567 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00568 {
00569     unsigned int i;
```

```

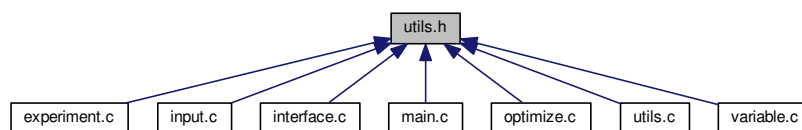
00570     for (i = 0; i < n; ++i)
00571         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00572             break;
00573     return i;
00574 }
00575
00576 #endif

```

## 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_message` (char \*title, char \*msg, int type)  
Function to show a dialog with a message.
- void `show_error` (char \*msg)  
Function to show a dialog with an error message.
- int `xml_node_get_int` (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
Function to get an integer number of a XML node property.
- unsigned int `xml_node_get_uint` (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
Function to get an unsigned integer number of a XML node property.
- unsigned int `xml_node_get_uint_with_default` (xmlNode \*node, const xmlChar \*prop, unsigned int default\_value, int \*error\_code)  
Function to get an unsigned integer number of a XML node property with a default value.
- double `xml_node_get_float` (xmlNode \*node, const xmlChar \*prop, int \*error\_code)  
Function to get a floating point number of a XML node property.
- double `xml_node_get_float_with_default` (xmlNode \*node, const xmlChar \*prop, double default\_value, int \*error\_code)  
Function to get a floating point number of a XML node property with a default value.
- void `xml_node_set_int` (xmlNode \*node, const xmlChar \*prop, int value)  
Function to set an integer number in a XML node property.
- void `xml_node_set_uint` (xmlNode \*node, const xmlChar \*prop, unsigned int value)

- Function to set an unsigned integer number in a XML node property.*
- void [xml\\_node\\_set\\_float](#) (xmlNode \*node, const xmlChar \*prop, double value)
- Function to set a floating point number in a XML node property.*
- int [json\\_object\\_get\\_int](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get an integer number of a JSON object property.*
- unsigned int [json\\_object\\_get\\_uint](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get an unsigned integer number of a JSON object property.*
- unsigned int [json\\_object\\_get\\_uint\\_with\\_default](#) (JsonObject \*object, const char \*prop, unsigned int default\_value, int \*error\_code)
- Function to get an unsigned integer number of a JSON object property with a default value.*
- double [json\\_object\\_get\\_float](#) (JsonObject \*object, const char \*prop, int \*error\_code)
- Function to get a floating point number of a JSON object property.*
- double [json\\_object\\_get\\_float\\_with\\_default](#) (JsonObject \*object, const char \*prop, double default\_value, int \*error\_code)
- Function to get a floating point number of a JSON object property with a default value.*
- void [json\\_object\\_set\\_int](#) (JsonObject \*object, const char \*prop, int value)
- Function to set an integer number in a JSON object property.*
- void [json\\_object\\_set\\_uint](#) (JsonObject \*object, const char \*prop, unsigned int value)
- Function to set an unsigned integer number in a JSON object property.*
- void [json\\_object\\_set\\_float](#) (JsonObject \*object, const char \*prop, double value)
- Function to set a floating point number in a JSON object property.*
- int [cores\\_number](#) ()
- Function to obtain the cores number.*
- void [process\\_pending](#) ()
- Function to process events on long computation.*
- unsigned int [gtk\\_array\\_get\\_active](#) (GtkRadioButton \*array[], unsigned int n)
- Function to get the active GtkRadioButton.*

## Variables

- GtkWidget \* [main\\_window](#)
- Main GtkWidget.*
- char \* [error\\_message](#)
- Error message.*
- void(\* [show\\_pending](#) )()
- Pointer to the function to show pending events.*

## 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 532 of file [utils.c](#).

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

### 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 567 of file [utils.c](#).

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

### 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 421 of file [utils.c](#).

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

### 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**

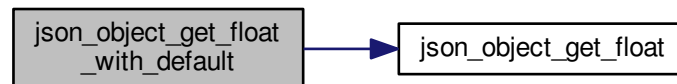
Floating point number value.

Definition at line [454](#) of file [utils.c](#).

```

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

Here is the call graph for this function:

**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 [330](#) of file [utils.c](#).

```

00331 {
00332     const char *buffer;
00333     int i = 0;
```

```

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

```

### 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 360 of file [utils.c](#).

```

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

```

### 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

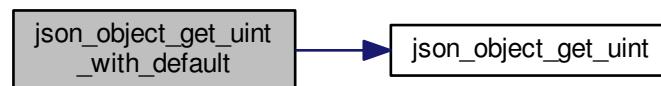
Unsigned integer number value.

Definition at line 393 of file [utils.c](#).

```

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

Here is the call graph for this function:



## 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 519 of file [utils.c](#).

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

#### 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 480 of file [utils.c](#).

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

#### 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 499 of file [utils.c](#).

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

### 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 103 of file [utils.c](#).

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

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 73 of file [utils.c](#).

```
00074 {
00075     #if HAVE_GTK
00076     GtkMessageDialog *dlg;
00077
00078     // Creating the dialog
```

```

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

```

### 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 213 of file [utils.c](#).

```

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

```

### 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**

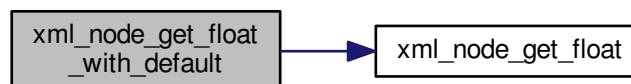
Floating point number value.

Definition at line 247 of file [utils.c](#).

```

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

Here is the call graph for this function:

**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 121 of file [utils.c](#).

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

### 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 152 of file [utils.c](#).

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

### 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.

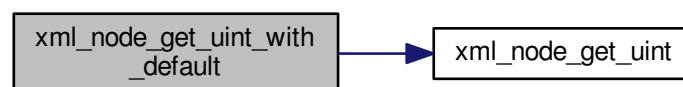
#### Returns

Unsigned integer number value.

Definition at line 186 of file [utils.c](#).

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

Here is the call graph for this function:



### 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 310 of file [utils.c](#).

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

#### 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 272 of file [utils.c](#).

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

#### 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 291 of file [utils.c](#).

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

## 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 extern void (*show_pending) ();
00046
00047 // Public functions
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,
00061                                         int *error_code);
00062 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00063 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00064                        unsigned int value);
00065 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00066 int json_object_get_int (JsonObject * object, const char *prop,
00067                        int *error_code);
00068 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00069                                   int *error_code);
```



```

00082 unsigned int json_object_get_uint_with_default (JsonObject * object,
00083                                                  const char *prop,
00084                                                  unsigned int default_value,
00085                                                  int *error_code);
00086 double json_object_get_float (JsonObject * object, const char *prop,
00087                               int *error_code);
00088 double json_object_get_float_with_default (JsonObject * object,
00089                                            const char *prop,
00090                                            double default_value,
00091                                            int *error_code);
00092 void json_object_set_int (JsonObject * object, const char *prop, int value);
00093 void json_object_set_uint (JsonObject * object, const char *prop,
00094                           unsigned int value);
00095 void json_object_set_float (JsonObject * object, const char *prop,
00096                           double value);
00097 int cores_number ();
00098 #if HAVE_GTK
00099 void process_pending ();
00100 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00101 #endif
00102
00103 #endif

```

## 5.25 variable.c File Reference

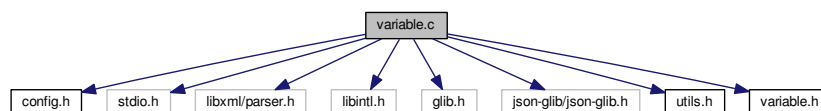
Source file to define the variable data.

```

#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, 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.*

### 5.25.1 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>	<a href="#">Variable</a> 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", _("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", _("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>	<a href="#">Variable</a> 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>	<a href="#">Variable</a> 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 303 of file [variable.c](#).

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

```

00351     = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352     if (error_code)
00353     {
00354         variable_error (variable, _("bad maximum"));
00355         goto exit_on_error;
00356     }
00357     variable->rangemaxabs
00358     = json_object_get_float_with_default (object,
00359     LABEL_ABSOLUTE_MAXIMUM,
00360     G_MAXDOUBLE, &error_code);
00361     if (error_code)
00362     {
00363         variable_error (variable, _("bad absolute maximum"));
00364         goto exit_on_error;
00365     }
00366     if (variable->rangemax > variable->rangemaxabs)
00367     {
00368         variable_error (variable, _("maximum range not allowed"));
00369         goto exit_on_error;
00370     }
00371     if (variable->rangemax < variable->rangemin)
00372     {
00373         variable_error (variable, _("bad range"));
00374         goto exit_on_error;
00375     }
00376     else
00377     {
00378         variable_error (variable, _("no maximum range"));
00379         goto exit_on_error;
00380     }
00381     variable->precision
00382     = json_object_get_uint_with_default (object,
00383     LABEL_PRECISION,
00384     DEFAULT_PRECISION, &error_code);
00385     if (error_code || variable->precision >= NPRECISIONS)
00386     {
00387         variable_error (variable, _("bad precision"));
00388         goto exit_on_error;
00389     }
00390     if (algorithm == ALGORITHM_SWEEP)
00391     {
00392         if (json_object_get_member (object, LABEL_NSWEEPS))
00393         {
00394             variable->nsweeps
00395             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00396             if (error_code || !variable->nsweeps)
00397             {
00398                 variable_error (variable, _("bad sweeps"));
00399                 goto exit_on_error;
00400             }
00401         }
00402         else
00403         {
00404             variable_error (variable, _("no sweeps number"));
00405             goto exit_on_error;
00406         }
00407     }
00408     #if DEBUG_VARIABLE
00409     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00410     #endif
00411     if (algorithm == ALGORITHM_GENETIC)
00412     {
00413         // Obtaining bits representing each variable
00414         if (json_object_get_member (object, LABEL_NBITS))
00415         {
00416             variable->nbits
00417             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00418             if (error_code || !variable->nbits)
00419             {
00420                 variable_error (variable, _("invalid bits number"));
00421                 goto exit_on_error;
00422             }
00423         }
00424         else
00425         {
00426             variable_error (variable, _("no bits number"));
00427             goto exit_on_error;
00428         }
00429     }
00430     else if (nsteps)
00431     {
00432         variable->step =
00433         json_object_get_float (object, LABEL_STEP, &error_code);
00434         if (error_code || variable->step < 0.)
00435         {
00436             variable_error (variable, _("bad step size"));

```

```

00436         goto exit_on_error;
00437     }
00438 }
00439
00440 #if DEBUG_VARIABLE
00441 fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443 return 1;
00444 exit_on_error:
00445     variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447 fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449 return 0;
00450 }

```

Here is the call graph for this function:



#### 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>	<a href="#">Variable</a> 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, _("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, _("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, _("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170         if (variable->rangemin < variable->rangeminabs)
00171         {
00172             variable_error (variable, _("minimum range not allowed"));
00173             goto exit_on_error;
00174         }
00175     }
00176     else
00177     {
00178         variable_error (variable, _("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, _("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, _("bad absolute maximum"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemax > variable->rangemaxabs)
00201         {
00202             variable_error (variable, _("maximum range not allowed"));
00203             goto exit_on_error;
00204         }
00205         if (variable->rangemax < variable->rangemin)
00206         {
00207             variable_error (variable, _("bad range"));
00208             goto exit_on_error;
00209         }
00210     }
00211     else
00212     {
00213         variable_error (variable, _("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, _("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,

```

```

00228                                     &error_code);
00229         if (error_code || !variable->nsweeps)
00230         {
00231             variable_error (variable, _("bad sweeps"));
00232             goto exit_on_error;
00233         }
00234     }
00235     else
00236     {
00237         variable_error (variable, _("no sweeps number"));
00238         goto exit_on_error;
00239     }
00240 #if DEBUG_VARIABLE
00241     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {
00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *)
00251 LABEL_NBITS,
00252                                     &error_code);
00253         if (error_code || !variable->nbits)
00254         {
00255             variable_error (variable, _("invalid bits number"));
00256             goto exit_on_error;
00257         }
00258     }
00259     else
00260     {
00261         variable_error (variable, _("no bits number"));
00262         goto exit_on_error;
00263     }
00264     else if (nsteps)
00265     {
00266         variable->step
00267         =
00268         xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269         if (error_code || variable->step < 0.)
00270         {
00271             variable_error (variable, _("bad step size"));
00272             goto exit_on_error;
00273         }
00274     }
00275 }
00276 #if DEBUG_VARIABLE
00277     fprintf (stderr, "variable_open_xml: end\n");
00278 #endif
00279     return 1;
00280 exit_on_error:
00281     variable_free (variable, INPUT_TYPE_XML);
00282 #if DEBUG_VARIABLE
00283     fprintf (stderr, "variable_open_xml: end\n");
00284 #endif
00285     return 0;
00286 }

```

Here is the call graph for this function:



## 5.25.3 Variable Documentation



## 5.25.3.1 format

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

**Initial value:**

```
= {
    "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
    "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
}
```

Array of C-strings with variable formats.

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

## 5.25.3.2 precision

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

**Initial value:**

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

Array of variable precisions.

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

## 5.26 variable.c

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

```

00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "utils.h"
00046 #include "variable.h"
00047
00048 #define DEBUG_VARIABLE 0
00049
00050 const char *format[NPRECISIONS] = {
00051     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00052     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00053 };
00054
00055 const double precision[NPRECISIONS] = {
00056     1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,
00057     1e-12, 1e-13, 1e-14
00058 };
00059
00060 void
00061 variable_new (Variable * variable)
00062 {
00063     #if DEBUG_VARIABLE
00064         fprintf (stderr, "variable_new: start\n");
00065     #endif
00066     variable->name = NULL;
00067     #if DEBUG_VARIABLE
00068         fprintf (stderr, "variable_new: end\n");
00069     #endif
00070 }
00071
00072 void
00073 variable_free (Variable * variable, unsigned int type)
00074 {
00075     #if DEBUG_VARIABLE
00076         fprintf (stderr, "variable_free: start\n");
00077     #endif
00078     if (type == INPUT_TYPE_XML)
00079         xmlFree (variable->name);
00080     else
00081         g_free (variable->name);
00082     #if DEBUG_VARIABLE
00083         fprintf (stderr, "variable_free: end\n");
00084     #endif
00085 }
00086
00087 void
00088 variable_error (Variable * variable, char *message)
00089 {
00090     char buffer[64];
00091     if (!variable->name)
00092         snprintf (buffer, 64, "%s: %s", _("Variable"), message);
00093     else
00094         snprintf (buffer, 64, "%s %s: %s", _("Variable"), variable->name,
00095                 message);
00096     error_message = g_strdup (buffer);
00097 }
00098
00099 int
00100 variable_open_xml (Variable * variable, xmlNode * node,
00101                  unsigned int algorithm, unsigned int nsteps)
00102 {
00103     int error_code;
00104     #if DEBUG_VARIABLE
00105         fprintf (stderr, "variable_open_xml: start\n");
00106     #endif
00107     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00108     if (!variable->name)
00109     {
00110         variable_error (variable, _("no name"));
00111         goto exit_on_error;
00112     }
00113     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00114     {
00115         variable->rangemin
00116             = xml_node_get_float (node, (const xmlChar *)
00117             LABEL_MINIMUM,
00118             &error_code);
00119         if (error_code)
00120         {
00121             variable_error (variable, _("bad minimum"));
00122             goto exit_on_error;
00123         }
00124     }
00125 }

```

```

00161     variable->rangeminabs = xml_node_get_float_with_default
00162     (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163     &error_code);
00164     if (error_code)
00165     {
00166         variable_error (variable, _("bad absolute minimum"));
00167         goto exit_on_error;
00168     }
00169     if (variable->rangemin < variable->rangeminabs)
00170     {
00171         variable_error (variable, _("minimum range not allowed"));
00172         goto exit_on_error;
00173     }
00174 }
00175 else
00176 {
00177     variable_error (variable, _("no minimum range"));
00178     goto exit_on_error;
00179 }
00180 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181 {
00182     variable->rangemax
00183     = xml_node_get_float (node, (const xmlChar *)
00184     LABEL_MAXIMUM,
00185     &error_code);
00186     if (error_code)
00187     {
00188         variable_error (variable, _("bad maximum"));
00189         goto exit_on_error;
00190     }
00191     variable->rangemaxabs = xml_node_get_float_with_default
00192     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00193     &error_code);
00194     if (error_code)
00195     {
00196         variable_error (variable, _("bad absolute maximum"));
00197         goto exit_on_error;
00198     }
00199     if (variable->rangemax > variable->rangemaxabs)
00200     {
00201         variable_error (variable, _("maximum range not allowed"));
00202         goto exit_on_error;
00203     }
00204     if (variable->rangemax < variable->rangemin)
00205     {
00206         variable_error (variable, _("bad range"));
00207         goto exit_on_error;
00208     }
00209 }
00210 else
00211 {
00212     variable_error (variable, _("no maximum range"));
00213     goto exit_on_error;
00214 }
00215 variable->precision
00216 = xml_node_get_uint_with_default (node, (const xmlChar *)
00217 LABEL_PRECISION,
00218     DEFAULT_PRECISION, &error_code);
00219 if (error_code || variable->precision >= NPRECISIONS)
00220 {
00221     variable_error (variable, _("bad precision"));
00222     goto exit_on_error;
00223 }
00224 if (algorithm == ALGORITHM_SWEEP)
00225 {
00226     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00227     {
00228         variable->nsweeps
00229         = xml_node_get_uint (node, (const xmlChar *)
00230 LABEL_NSWEEPS,
00231     &error_code);
00232         if (error_code || !variable->nsweeps)
00233         {
00234             variable_error (variable, _("bad sweeps"));
00235             goto exit_on_error;
00236         }
00237     }
00238 }
00239 else
00240 {
00241     variable_error (variable, _("no sweeps number"));
00242     goto exit_on_error;
00243 }
00244 #if DEBUG_VARIABLE
00245 fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00246 #endif
00247 }
00248 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, _("invalid bits number"));
00256                 goto exit_on_error;
00257             }
00258         }
00259         else if (nsteps)
00260         {
00261             variable->step
00262             =
00263             xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00264             if (error_code || variable->step < 0.)
00265             {
00266                 variable_error (variable, _("bad step size"));
00267                 goto exit_on_error;
00268             }
00269         }
00270     }
00271     #if DEBUG_VARIABLE
00272     fprintf (stderr, "variable_open_xml: end\n");
00273     #endif
00274     return 1;
00275 exit_on_error:
00276     variable_free (variable, INPUT_TYPE_XML);
00277     #if DEBUG_VARIABLE
00278     fprintf (stderr, "variable_open_xml: end\n");
00279     #endif
00280     return 0;
00281 }
00282
00283 int
00284 variable_open_json (Variable * variable, JsonNode * node,
00285                     unsigned int algorithm, unsigned int nsteps)
00286 {
00287     JsonObject *object;
00288     const char *label;
00289     int error_code;
00290     #if DEBUG_VARIABLE
00291     fprintf (stderr, "variable_open_json: start\n");
00292     #endif
00293     object = json_node_get_object (node);
00294     label = json_object_get_string_member (object, LABEL_NAME);
00295     if (!label)
00296     {
00297         variable_error (variable, _("no name"));
00298         goto exit_on_error;
00299     }
00300     variable->name = g_strdup (label);
00301     if (json_object_get_member (object, LABEL_MINIMUM))
00302     {
00303         variable->rangemin
00304         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00305         if (error_code)
00306         {
00307             variable_error (variable, _("bad minimum"));
00308             goto exit_on_error;
00309         }
00310         variable->rangeminabs
00311         = json_object_get_float_with_default (object,
00312 LABEL_ABSOLUTE_MINIMUM,
00313                                             -G_MAXDOUBLE, &error_code);
00314         if (error_code)
00315         {
00316             variable_error (variable, _("bad absolute minimum"));
00317             goto exit_on_error;
00318         }
00319         if (variable->rangemin < variable->rangeminabs)
00320         {
00321             variable_error (variable, _("minimum range not allowed"));
00322             goto exit_on_error;
00323         }
00324     }
00325     else

```

```

00344     {
00345         variable_error (variable, _("no minimum range"));
00346         goto exit_on_error;
00347     }
00348     if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350         variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352         if (error_code)
00353         {
00354             variable_error (variable, _("bad maximum"));
00355             goto exit_on_error;
00356         }
00357         variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
00359         LABEL_ABSOLUTE_MAXIMUM,
00360         G_MAXDOUBLE, &error_code);
00361         if (error_code)
00362         {
00363             variable_error (variable, _("bad absolute maximum"));
00364             goto exit_on_error;
00365         }
00366         if (variable->rangemax > variable->rangemaxabs)
00367         {
00368             variable_error (variable, _("maximum range not allowed"));
00369             goto exit_on_error;
00370         }
00371         if (variable->rangemax < variable->rangemin)
00372         {
00373             variable_error (variable, _("bad range"));
00374             goto exit_on_error;
00375         }
00376     }
00377     else
00378     {
00379         variable_error (variable, _("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, _("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
00396             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00397             if (error_code || !variable->nsweeps)
00398             {
00399                 variable_error (variable, _("bad sweeps"));
00400                 goto exit_on_error;
00401             }
00402         }
00403         else
00404         {
00405             variable_error (variable, _("no sweeps number"));
00406             goto exit_on_error;
00407         }
00408     }
00409     #if DEBUG_VARIABLE
00410     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00411     #endif
00412     if (algorithm == ALGORITHM_GENETIC)
00413     {
00414         // Obtaining bits representing each variable
00415         if (json_object_get_member (object, LABEL_NBITS))
00416         {
00417             variable->nbits
00418             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00419             if (error_code || !variable->nbits)
00420             {
00421                 variable_error (variable, _("invalid bits number"));
00422                 goto exit_on_error;
00423             }
00424         }
00425         else
00426         {
00427             variable_error (variable, _("no bits number"));
00428             goto exit_on_error;
00429         }
00430     }
00431 }

```

```

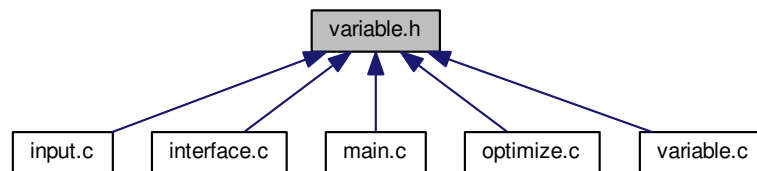
00429     else if (nsteps)
00430     {
00431         variable->step =
00432             json_object_get_float (object, LABEL_STEP, &error_code);
00433         if (error_code || variable->step < 0.)
00434         {
00435             variable_error (variable, _("bad step size"));
00436             goto exit_on_error;
00437         }
00438     }
00439
00440 #if DEBUG_VARIABLE
00441     fprintf (stderr, "variable_open_json: end\n");
00442 #endif
00443     return 1;
00444 exit_on_error:
00445     variable_free (variable, INPUT_TYPE_JSON);
00446 #if DEBUG_VARIABLE
00447     fprintf (stderr, "variable_open_json: end\n");
00448 #endif
00449     return 0;
00450 }

```

## 5.27 variable.h File Reference

Header file to define the variable data.

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



## Data Structures

- struct [Variable](#)

*Struct to define the variable data.*

## Enumerations

- enum [Algorithm](#) { [ALGORITHM\\_MONTE\\_CARLO](#) = 0, [ALGORITHM\\_SWEEP](#) = 1, [ALGORITHM\\_GENETIC](#) = 2 }

*Enum to define the algorithms.*

## Functions

- void [variable\\_new](#) ([Variable](#) \*variable)  
*Function to create a new [Variable](#) struct.*
- void [variable\\_free](#) ([Variable](#) \*variable, unsigned int type)  
*Function to free the memory of a [Variable](#) struct.*
- void [variable\\_error](#) ([Variable](#) \*variable, char \*message)  
*Function to print a message error opening an [Variable](#) struct.*
- int [variable\\_open\\_xml](#) ([Variable](#) \*variable, xmlNode \*node, unsigned int algorithm, unsigned int nsteps)  
*Function to open the variable file.*
- int [variable\\_open\\_json](#) ([Variable](#) \*variable, JsonNode \*node, unsigned int algorithm, unsigned int nsteps)  
*Function to open the variable file.*

## Variables

- const char \* [format](#) [[NPRECISIONS](#)]  
*Array of C-strings with variable formats.*
- const double [precision](#) [[NPRECISIONS](#)]  
*Array of variable precisions.*

### 5.27.1 Detailed Description

Header file to define the variable data.

#### Authors

Javier Burguete.

#### Copyright

Copyright 2012-2016, all rights reserved.

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

### 5.27.2 Enumeration Type Documentation

#### 5.27.2.1 Algorithm

enum [Algorithm](#)

Enum to define the algorithms.

#### Enumerator

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

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

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

## 5.27.3 Function Documentation

### 5.27.3.1 `variable_error()`

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

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

#### Parameters

<i>variable</i>	<a href="#">Variable</a> 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", _("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", _("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>	<a href="#">Variable</a> 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>	<a href="#">Variable</a> 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>	<a href="#">Variable</a> 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 303 of file [variable.c](#).

```

00305 {
00306     JsonObject *object;
00307     const char *label;
00308     int error_code;
00309     #if DEBUG_VARIABLE
00310     fprintf (stderr, "variable_open_json: start\n");
00311     #endif
00312     object = json_node_get_object (node);
00313     label = json_object_get_string_member (object, LABEL_NAME);
00314     if (!label)
00315     {
00316         variable_error (variable, _("no name"));
00317         goto exit_on_error;
00318     }
00319     variable->name = g_strdup (label);
00320     if (json_object_get_member (object, LABEL_MINIMUM))
00321     {
00322         variable->rangemin
00323         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00324         if (error_code)
00325         {
00326             variable_error (variable, _("bad minimum"));
00327             goto exit_on_error;
00328         }
00329         variable->rangeminabs
00330         = json_object_get_float_with_default (object,
00331 LABEL_ABSOLUTE_MINIMUM,
00332                                             -G_MAXDOUBLE, &error_code);
00333         if (error_code)
00334         {
00335             variable_error (variable, _("bad absolute minimum"));
00336             goto exit_on_error;
00337         }
00338         if (variable->rangemin < variable->rangeminabs)
00339         {
00340             variable_error (variable, _("minimum range not allowed"));
00341             goto exit_on_error;
00342         }
00343     }
00344     else
00345     {
00346         variable_error (variable, _("no minimum range"));
00347         goto exit_on_error;
00348     }
00349     if (json_object_get_member (object, LABEL_MAXIMUM))
00350     {
00351         variable->rangemax
00352         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00353         if (error_code)
00354         {
00355             variable_error (variable, _("bad maximum"));
00356             goto exit_on_error;
00357         }
00358         variable->rangemaxabs
00359         = json_object_get_float_with_default (object,
00360 LABEL_ABSOLUTE_MAXIMUM,
00361                                             G_MAXDOUBLE, &error_code);
00362         if (error_code)
00363         {
00364             variable_error (variable, _("bad absolute maximum"));
00365             goto exit_on_error;
00366         }
00367         if (variable->rangemax > variable->rangemaxabs)
00368         {
00369             variable_error (variable, _("maximum range not allowed"));
00370             goto exit_on_error;
00371         }
00372         if (variable->rangemax < variable->rangemin)
00373         {
00374             variable_error (variable, _("bad range"));
00375             goto exit_on_error;
00376         }
00377     }
00378     else
00379     {
00380         variable_error (variable, _("no maximum range"));
00381         goto exit_on_error;
00382     }
00383     variable->precision
00384     = json_object_get_uint_with_default (object,
00385 LABEL_PRECISION,
00386                                         DEFAULT_PRECISION, &error_code);
00387     if (error_code || variable->precision >= NPRECISIONS)
00388     {
00389         variable_error (variable, _("bad precision"));
00390         goto exit_on_error;
00391     }

```

```

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

```

Here is the call graph for this function:



### 5.27.3.5 variable\_open\_xml()

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

Function to open the variable file.

#### Parameters

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

#### Returns

1 on success, 0 on error.

Definition at line 136 of file [variable.c](#).

```
00138 {
00139     int error_code;
00140
00141     #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143     #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, _("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, _("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, _("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170         if (variable->rangemin < variable->rangeminabs)
00171         {
00172             variable_error (variable, _("minimum range not allowed"));
00173             goto exit_on_error;
00174         }
00175     }
00176     else
00177     {
00178         variable_error (variable, _("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 *)
```

```

    LABEL_MAXIMUM,
00184                                     &error_code);
00185     if (error_code)
00186     {
00187         variable_error (variable, _("bad maximum"));
00188         goto exit_on_error;
00189     }
00190     variable->rangemaxabs = xml_node_get_float_with_default
00191     (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00192      &error_code);
00193     if (error_code)
00194     {
00195         variable_error (variable, _("bad absolute maximum"));
00196         goto exit_on_error;
00197     }
00198     if (variable->rangemax > variable->rangemaxabs)
00199     {
00200         variable_error (variable, _("maximum range not allowed"));
00201         goto exit_on_error;
00202     }
00203     if (variable->rangemax < variable->rangemin)
00204     {
00205         variable_error (variable, _("bad range"));
00206         goto exit_on_error;
00207     }
00208 }
00209 else
00210 {
00211     variable_error (variable, _("no maximum range"));
00212     goto exit_on_error;
00213 }
00214 variable->precision
00215 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_PRECISION,
00216                                     DEFAULT_PRECISION, &error_code);
00217 if (error_code || variable->precision >= NPRECISIONS)
00218 {
00219     variable_error (variable, _("bad precision"));
00220     goto exit_on_error;
00221 }
00222 if (algorithm == ALGORITHM_SWEEP)
00223 {
00224     if (xmlHasProp (node, (const xmlChar *) LABEL_NSWEEPS))
00225     {
00226         variable->nsweeps
00227         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NSWEEPS,
00228                                     &error_code);
00229         if (error_code || !variable->nsweeps)
00230         {
00231             variable_error (variable, _("bad sweeps"));
00232             goto exit_on_error;
00233         }
00234     }
00235     else
00236     {
00237         variable_error (variable, _("no sweeps number"));
00238         goto exit_on_error;
00239     }
00240 #if DEBUG_VARIABLE
00241     fprintf (stderr, "variable_open_xml: nsweeps=%u\n", variable->nsweeps);
00242 #endif
00243 }
00244 if (algorithm == ALGORITHM_GENETIC)
00245 {
00246     // Obtaining bits representing each variable
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00248     {
00249         variable->nbits
00250         = xml_node_get_uint (node, (const xmlChar *)
LABEL_NBITS,
00251                                     &error_code);
00252         if (error_code || !variable->nbits)
00253         {
00254             variable_error (variable, _("invalid bits number"));
00255             goto exit_on_error;
00256         }
00257     }
00258     else
00259     {
00260         variable_error (variable, _("no bits number"));
00261         goto exit_on_error;
00262     }
00263 }
00264 else if (nsteps)
00265 {
00266     variable->step

```

```

00267         =
00268         xml_node_get_float (node, (const xmlChar *) LABEL_STEP, &error_code);
00269         if (error_code || variable->step < 0.)
00270         {
00271             variable_error (variable, _("bad step size"));
00272             goto exit_on_error;
00273         }
00274     }
00275
00276     #if DEBUG_VARIABLE
00277     fprintf (stderr, "variable_open_xml: end\n");
00278     #endif
00279     return 1;
00280 exit_on_error:
00281     variable_free (variable, INPUT_TYPE_XML);
00282     #if DEBUG_VARIABLE
00283     fprintf (stderr, "variable_open_xml: end\n");
00284     #endif
00285     return 0;
00286 }

```

Here is the call graph for this function:



## 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

```

```
00050 };
00051
00056 typedef struct
00057 {
00058     char *name;
00059     double rangemin;
00060     double rangemax;
00061     double rangeminabs;
00062     double rangemaxabs;
00063     double step;
00064     unsigned int precision;
00065     unsigned int nsweeps;
00066     unsigned int nbits;
00067 } Variable;
00068
00069 extern const char *format[NPRECISIONS];
00070 extern const double precision[NPRECISIONS];
00071
00072 // Public functions
00073 void variable_new (Variable * variable);
00074 void variable_free (Variable * variable, unsigned int type);
00075 void variable_error (Variable * variable, char *message);
00076 int variable_open_xml (Variable * variable, xmlNode * node,
00077                       unsigned int algorithm, unsigned int nsteps);
00078 int variable_open_json (Variable * variable, JsonNode * node,
00079                        unsigned int algorithm, unsigned int nsteps);
00080
00081 #endif
```





# Index

Algorithm  
variable.h, 285

config.h, 29  
INPUT\_TYPE, 32

cores\_number  
utils.c, 240  
utils.h, 258

DirectionMethod  
input.h, 80

ErrorNorm  
input.h, 80

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, 278

gtk\_array\_get\_active  
interface.h, 155  
utils.c, 240  
utils.h, 258

INPUT\_TYPE  
config.h, 32

Input, 16

input.c, 53  
input\_error, 54  
input\_open, 54  
input\_open\_json, 56  
input\_open\_xml, 61

input.h, 78  
DirectionMethod, 80  
ErrorNorm, 80  
input\_error, 81  
input\_open, 81  
input\_open\_json, 82  
input\_open\_xml, 88

input\_error  
input.c, 54  
input.h, 81

input\_open  
input.c, 54  
input.h, 81

input\_open\_json  
input.c, 56  
input.h, 82

input\_open\_xml  
input.c, 61  
input.h, 88

input\_save  
interface.c, 97  
interface.h, 156

input\_save\_direction\_json  
interface.c, 98

input\_save\_direction\_xml  
interface.c, 99

input\_save\_json  
interface.c, 100

input\_save\_xml  
interface.c, 103

interface.c, 95  
input\_save, 97  
input\_save\_direction\_json, 98  
input\_save\_direction\_xml, 99  
input\_save\_json, 100  
input\_save\_xml, 103  
window\_get\_algorithm, 105  
window\_get\_direction, 106  
window\_get\_norm, 107  
window\_new, 107  
window\_read, 116

- window\_save, 118
  - window\_template\_experiment, 120
- interface.h, 153
  - gtk\_array\_get\_active, 155
  - input\_save, 156
  - window\_get\_algorithm, 157
  - window\_get\_direction, 157
  - window\_get\_norm, 158
  - window\_new, 159
  - window\_read, 168
  - window\_save, 170
  - window\_template\_experiment, 172
- json\_object\_get\_float
  - utils.c, 240
  - utils.h, 258
- json\_object\_get\_float\_with\_default
  - utils.c, 241
  - utils.h, 259
- json\_object\_get\_int
  - utils.c, 242
  - utils.h, 260
- json\_object\_get\_uint
  - utils.c, 243
  - utils.h, 261
- json\_object\_get\_uint\_with\_default
  - utils.c, 243
  - utils.h, 261
- json\_object\_set\_float
  - utils.c, 244
  - utils.h, 262
- json\_object\_set\_int
  - utils.c, 245
  - utils.h, 263
- json\_object\_set\_uint
  - utils.c, 245
  - utils.h, 263
- main
  - main.c, 176
- main.c, 175
  - main, 176
- Optimize, 17
  - thread\_direction, 20
- optimize.c, 182
  - optimize\_best, 185
  - optimize\_best\_direction, 186
  - optimize\_direction\_sequential, 186
  - optimize\_direction\_thread, 187
  - optimize\_estimate\_direction\_coordinates, 188
  - optimize\_estimate\_direction\_random, 189
  - optimize\_genetic\_objective, 189
  - optimize\_input, 190
  - optimize\_merge, 191
  - optimize\_norm\_euclidian, 192
  - optimize\_norm\_maximum, 193
  - optimize\_norm\_p, 194
  - optimize\_norm\_taxicab, 194
  - optimize\_parse, 195
  - optimize\_save\_variables, 197
  - optimize\_step\_direction, 198
  - optimize\_thread, 199
- optimize.h, 218
  - optimize\_best, 220
  - optimize\_best\_direction, 221
  - optimize\_direction\_sequential, 222
  - optimize\_direction\_thread, 223
  - optimize\_estimate\_direction\_coordinates, 224
  - optimize\_estimate\_direction\_random, 224
  - optimize\_genetic\_objective, 225
  - optimize\_input, 226
  - optimize\_merge, 227
  - optimize\_norm\_euclidian, 228
  - optimize\_norm\_maximum, 229
  - optimize\_norm\_p, 229
  - optimize\_norm\_taxicab, 230
  - optimize\_parse, 231
  - optimize\_save\_variables, 233
  - optimize\_step\_direction, 233
  - optimize\_thread, 235
- optimize\_best
  - optimize.c, 185
  - optimize.h, 220
- optimize\_best\_direction
  - optimize.c, 186
  - optimize.h, 221
- optimize\_direction\_sequential
  - optimize.c, 186
  - optimize.h, 222
- optimize\_direction\_thread
  - optimize.c, 187
  - optimize.h, 223
- optimize\_estimate\_direction\_coordinates
  - optimize.c, 188
  - optimize.h, 224
- optimize\_estimate\_direction\_random
  - optimize.c, 189
  - optimize.h, 224
- optimize\_genetic\_objective
  - optimize.c, 189
  - optimize.h, 225
- optimize\_input
  - optimize.c, 190
  - optimize.h, 226
- optimize\_merge
  - optimize.c, 191
  - optimize.h, 227
- optimize\_norm\_euclidian
  - optimize.c, 192
  - optimize.h, 228
- optimize\_norm\_maximum
  - optimize.c, 193
  - optimize.h, 229
- optimize\_norm\_p
  - optimize.c, 194
  - optimize.h, 229

- optimize\_norm\_taxicab
  - optimize.c, [194](#)
  - optimize.h, [230](#)
- optimize\_parse
  - optimize.c, [195](#)
  - optimize.h, [231](#)
- optimize\_save\_variables
  - optimize.c, [197](#)
  - optimize.h, [233](#)
- optimize\_step\_direction
  - optimize.c, [198](#)
  - optimize.h, [233](#)
- optimize\_thread
  - optimize.c, [199](#)
  - optimize.h, [235](#)
- Options, [20](#)
- ParallelData, [21](#)
- precision
  - variable.c, [279](#)
- Running, [21](#)
- show\_error
  - utils.c, [245](#)
  - utils.h, [263](#)
- show\_message
  - utils.c, [246](#)
  - utils.h, [264](#)
- template
  - experiment.c, [41](#)
- thread\_direction
  - Optimize, [20](#)
- utils.c, [238](#)
  - cores\_number, [240](#)
  - gtk\_array\_get\_active, [240](#)
  - json\_object\_get\_float, [240](#)
  - json\_object\_get\_float\_with\_default, [241](#)
  - json\_object\_get\_int, [242](#)
  - json\_object\_get\_uint, [243](#)
  - json\_object\_get\_uint\_with\_default, [243](#)
  - json\_object\_set\_float, [244](#)
  - json\_object\_set\_int, [245](#)
  - json\_object\_set\_uint, [245](#)
  - show\_error, [245](#)
  - show\_message, [246](#)
  - xml\_node\_get\_float, [247](#)
  - xml\_node\_get\_float\_with\_default, [247](#)
  - xml\_node\_get\_int, [248](#)
  - xml\_node\_get\_uint, [249](#)
  - xml\_node\_get\_uint\_with\_default, [249](#)
  - xml\_node\_set\_float, [250](#)
  - xml\_node\_set\_int, [251](#)
  - xml\_node\_set\_uint, [251](#)
- utils.h, [256](#)
  - cores\_number, [258](#)
  - gtk\_array\_get\_active, [258](#)
  - json\_object\_get\_float, [258](#)
  - json\_object\_get\_float\_with\_default, [259](#)
  - json\_object\_get\_int, [260](#)
  - json\_object\_get\_uint, [261](#)
  - json\_object\_get\_uint\_with\_default, [261](#)
  - json\_object\_set\_float, [262](#)
  - json\_object\_set\_int, [263](#)
  - json\_object\_set\_uint, [263](#)
  - show\_error, [263](#)
  - show\_message, [264](#)
  - xml\_node\_get\_float, [265](#)
  - xml\_node\_get\_float\_with\_default, [265](#)
  - xml\_node\_get\_int, [266](#)
  - xml\_node\_get\_uint, [267](#)
  - xml\_node\_get\_uint\_with\_default, [267](#)
  - xml\_node\_set\_float, [268](#)
  - xml\_node\_set\_int, [269](#)
  - xml\_node\_set\_uint, [269](#)
- Variable, [22](#)
- variable.c, [271](#)
  - format, [278](#)
  - precision, [279](#)
  - variable\_error, [272](#)
  - variable\_free, [273](#)
  - variable\_new, [273](#)
  - variable\_open\_json, [273](#)
  - variable\_open\_xml, [276](#)
- variable.h, [284](#)
  - Algorithm, [285](#)
  - variable\_error, [286](#)
  - variable\_free, [286](#)
  - variable\_new, [287](#)
  - variable\_open\_json, [287](#)
  - variable\_open\_xml, [289](#)
- variable\_error
  - variable.c, [272](#)
  - variable.h, [286](#)
- variable\_free
  - variable.c, [273](#)
  - variable.h, [286](#)
- variable\_new
  - variable.c, [273](#)
  - variable.h, [287](#)
- variable\_open\_json
  - variable.c, [273](#)
  - variable.h, [287](#)
- variable\_open\_xml
  - variable.c, [276](#)
  - variable.h, [289](#)
- Window, [23](#)
- window\_get\_algorithm
  - interface.c, [105](#)
  - interface.h, [157](#)
- window\_get\_direction
  - interface.c, [106](#)
  - interface.h, [157](#)
- window\_get\_norm

- interface.c, [107](#)
  - interface.h, [158](#)
- window\_new
  - interface.c, [107](#)
  - interface.h, [159](#)
- window\_read
  - interface.c, [116](#)
  - interface.h, [168](#)
- window\_save
  - interface.c, [118](#)
  - interface.h, [170](#)
- window\_template\_experiment
  - interface.c, [120](#)
  - interface.h, [172](#)
- xml\_node\_get\_float
  - utils.c, [247](#)
  - utils.h, [265](#)
- xml\_node\_get\_float\_with\_default
  - utils.c, [247](#)
  - utils.h, [265](#)
- xml\_node\_get\_int
  - utils.c, [248](#)
  - utils.h, [266](#)
- xml\_node\_get\_uint
  - utils.c, [249](#)
  - utils.h, [267](#)
- xml\_node\_get\_uint\_with\_default
  - utils.c, [249](#)
  - utils.h, [267](#)
- xml\_node\_set\_float
  - utils.c, [250](#)
  - utils.h, [268](#)
- xml\_node\_set\_int
  - utils.c, [251](#)
  - utils.h, [269](#)
- xml\_node\_set\_uint
  - utils.c, [251](#)
  - utils.h, [269](#)