

MPCOTool

3.0.3

Generated by Doxygen 1.8.11

Contents

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

5	File Documentation	29
5.1	config.h File Reference	29
5.2	config.h	29
5.3	experiment.c File Reference	30
5.3.1	Detailed Description	31
5.3.2	Function Documentation	31
5.3.2.1	experiment_error(Experiment *experiment, char *message)	31
5.3.2.2	experiment_free(Experiment *experiment, unsigned int type)	32
5.3.2.3	experiment_new(Experiment *experiment)	32
5.3.2.4	experiment_open_json(Experiment *experiment, JsonNode *node, unsigned int ninputs)	33
5.3.2.5	experiment_open_xml(Experiment *experiment, xmlNode *node, unsigned int ninputs)	34
5.3.3	Variable Documentation	36
5.3.3.1	template	36
5.4	experiment.c	36
5.5	experiment.h File Reference	40
5.5.1	Detailed Description	40
5.5.2	Function Documentation	40
5.5.2.1	experiment_error(Experiment *experiment, char *message)	40
5.5.2.2	experiment_free(Experiment *experiment, unsigned int type)	41
5.5.2.3	experiment_new(Experiment *experiment)	41
5.5.2.4	experiment_open_json(Experiment *experiment, JsonNode *node, unsigned int ninputs)	42
5.5.2.5	experiment_open_xml(Experiment *experiment, xmlNode *node, unsigned int ninputs)	43
5.6	experiment.h	45
5.7	input.c File Reference	46
5.7.1	Detailed Description	47
5.7.2	Function Documentation	47
5.7.2.1	input_error(char *message)	47
5.7.2.2	input_open(char *filename)	47

5.7.2.3	input_open_json(JsonParser *parser)	48
5.7.2.4	input_open_xml(xmlDoc *doc)	53
5.8	input.c	59
5.9	input.h File Reference	70
5.9.1	Detailed Description	71
5.9.2	Enumeration Type Documentation	71
5.9.2.1	DirectionMethod	71
5.9.2.2	ErrorNorm	72
5.9.3	Function Documentation	72
5.9.3.1	input_error(char *message)	72
5.9.3.2	input_open(char *filename)	72
5.9.3.3	input_open_json(JsonParser *parser)	73
5.9.3.4	input_open_xml(xmlDoc *doc)	78
5.10	input.h	84
5.11	interface.c File Reference	85
5.11.1	Detailed Description	87
5.11.2	Function Documentation	87
5.11.2.1	input_save(char *filename)	87
5.11.2.2	input_save_direction_json(JsonNode *node)	88
5.11.2.3	input_save_direction_xml(xmlNode *node)	89
5.11.2.4	input_save_json(JsonGenerator *generator)	89
5.11.2.5	input_save_xml(xmlDoc *doc)	92
5.11.2.6	window_get_algorithm()	94
5.11.2.7	window_get_direction()	94
5.11.2.8	window_get_norm()	95
5.11.2.9	window_read(char *filename)	95
5.11.2.10	window_save()	97
5.11.2.11	window_template_experiment(void *data)	99
5.12	interface.c	100
5.13	interface.h File Reference	131

5.13.1 Detailed Description	133
5.13.2 Function Documentation	133
5.13.2.1 gtk_array_get_active(GtkRadioButton *array[], unsigned int n)	133
5.13.2.2 input_save(char *filename)	134
5.13.2.3 window_get_algorithm()	135
5.13.2.4 window_get_direction()	135
5.13.2.5 window_get_norm()	136
5.13.2.6 window_read(char *filename)	136
5.13.2.7 window_save()	138
5.13.2.8 window_template_experiment(void *data)	140
5.14 interface.h	140
5.15 main.c File Reference	143
5.15.1 Detailed Description	144
5.16 main.c	144
5.17 optimize.c File Reference	147
5.17.1 Detailed Description	149
5.17.2 Function Documentation	149
5.17.2.1 optimize_best(unsigned int simulation, double value)	149
5.17.2.2 optimize_best_direction(unsigned int simulation, double value)	150
5.17.2.3 optimize_direction_sequential(unsigned int simulation)	151
5.17.2.4 optimize_direction_thread(ParallelData *data)	152
5.17.2.5 optimize_estimate_direction_coordinates(unsigned int variable, unsigned int estimate)	153
5.17.2.6 optimize_estimate_direction_random(unsigned int variable, unsigned int estimate)	153
5.17.2.7 optimize_genetic_objective(Entity *entity)	154
5.17.2.8 optimize_input(unsigned int simulation, char *input, GMappedFile *template)	154
5.17.2.9 optimize_merge(unsigned int nsaveds, unsigned int *simulation_best, double *error_best)	156
5.17.2.10 optimize_norm_euclidian(unsigned int simulation)	156
5.17.2.11 optimize_norm_maximum(unsigned int simulation)	157
5.17.2.12 optimize_norm_p(unsigned int simulation)	157

5.17.2.13 optimize_norm_taxicab(unsigned int simulation)	158
5.17.2.14 optimize_parse(unsigned int simulation, unsigned int experiment)	159
5.17.2.15 optimize_save_variables(unsigned int simulation, double error)	160
5.17.2.16 optimize_step_direction(unsigned int simulation)	161
5.17.2.17 optimize_thread(ParallelData *data)	162
5.18 optimize.c	162
5.19 optimize.h File Reference	180
5.19.1 Detailed Description	182
5.19.2 Function Documentation	182
5.19.2.1 optimize_best(unsigned int simulation, double value)	182
5.19.2.2 optimize_best_direction(unsigned int simulation, double value)	183
5.19.2.3 optimize_direction_sequential(unsigned int simulation)	184
5.19.2.4 optimize_direction_thread(ParallelData *data)	185
5.19.2.5 optimize_estimate_direction_coordinates(unsigned int variable, unsigned int estimate)	186
5.19.2.6 optimize_estimate_direction_random(unsigned int variable, unsigned int estimate)	186
5.19.2.7 optimize_genetic_objective(Entity *entity)	187
5.19.2.8 optimize_input(unsigned int simulation, char *input, GMappedFile *template)	187
5.19.2.9 optimize_merge(unsigned int nsaveds, unsigned int *simulation_best, double *error_best)	189
5.19.2.10 optimize_norm_euclidian(unsigned int simulation)	189
5.19.2.11 optimize_norm_maximum(unsigned int simulation)	190
5.19.2.12 optimize_norm_p(unsigned int simulation)	190
5.19.2.13 optimize_norm_taxicab(unsigned int simulation)	191
5.19.2.14 optimize_parse(unsigned int simulation, unsigned int experiment)	192
5.19.2.15 optimize_save_variables(unsigned int simulation, double error)	193
5.19.2.16 optimize_step_direction(unsigned int simulation)	194
5.19.2.17 optimize_thread(ParallelData *data)	195
5.20 optimize.h	195
5.21 utils.c File Reference	197
5.21.1 Detailed Description	199

5.21.2	Function Documentation	199
5.21.2.1	cores_number()	199
5.21.2.2	gtk_array_get_active(GtkRadioButton *array[], unsigned int n)	199
5.21.2.3	json_object_get_float(JsonObject *object, const char *prop, int *error_code)	200
5.21.2.4	json_object_get_float_with_default(JsonObject *object, const char *prop, double default_value, int *error_code)	200
5.21.2.5	json_object_get_int(JsonObject *object, const char *prop, int *error_code)	201
5.21.2.6	json_object_get_uint(JsonObject *object, const char *prop, int *error_code)	201
5.21.2.7	json_object_get_uint_with_default(JsonObject *object, const char *prop, unsigned int default_value, int *error_code)	202
5.21.2.8	json_object_set_float(JsonObject *object, const char *prop, double value)	203
5.21.2.9	json_object_set_int(JsonObject *object, const char *prop, int value)	203
5.21.2.10	json_object_set_uint(JsonObject *object, const char *prop, unsigned int value)	203
5.21.2.11	show_error(char *msg)	204
5.21.2.12	show_message(char *title, char *msg, int type)	204
5.21.2.13	xml_node_get_float(xmlNode *node, const xmlChar *prop, int *error_code)	205
5.21.2.14	xml_node_get_float_with_default(xmlNode *node, const xmlChar *prop, double default_value, int *error_code)	206
5.21.2.15	xml_node_get_int(xmlNode *node, const xmlChar *prop, int *error_code)	207
5.21.2.16	xml_node_get_uint(xmlNode *node, const xmlChar *prop, int *error_code)	207
5.21.2.17	xml_node_get_uint_with_default(xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)	208
5.21.2.18	xml_node_set_float(xmlNode *node, const xmlChar *prop, double value)	208
5.21.2.19	xml_node_set_int(xmlNode *node, const xmlChar *prop, int value)	209
5.21.2.20	xml_node_set_uint(xmlNode *node, const xmlChar *prop, unsigned int value)	209
5.22	utils.c	209
5.23	utils.h File Reference	213
5.23.1	Detailed Description	215
5.23.2	Function Documentation	215
5.23.2.1	cores_number()	215
5.23.2.2	gtk_array_get_active(GtkRadioButton *array[], unsigned int n)	215
5.23.2.3	json_object_get_float(JsonObject *object, const char *prop, int *error_code)	216

5.23.2.4	<code>json_object_get_float_with_default(JsonObject *object, const char *prop, double default_value, int *error_code)</code>	216
5.23.2.5	<code>json_object_get_int(JsonObject *object, const char *prop, int *error_code)</code>	217
5.23.2.6	<code>json_object_get_uint(JsonObject *object, const char *prop, int *error_code)</code>	217
5.23.2.7	<code>json_object_get_uint_with_default(JsonObject *object, const char *prop, unsigned int default_value, int *error_code)</code>	218
5.23.2.8	<code>json_object_set_float(JsonObject *object, const char *prop, double value)</code>	219
5.23.2.9	<code>json_object_set_int(JsonObject *object, const char *prop, int value)</code>	219
5.23.2.10	<code>json_object_set_uint(JsonObject *object, const char *prop, unsigned int value)</code>	219
5.23.2.11	<code>show_error(char *msg)</code>	220
5.23.2.12	<code>show_message(char *title, char *msg, int type)</code>	220
5.23.2.13	<code>xml_node_get_float(xmlNode *node, const xmlChar *prop, int *error_code)</code>	221
5.23.2.14	<code>xml_node_get_float_with_default(xmlNode *node, const xmlChar *prop, double default_value, int *error_code)</code>	222
5.23.2.15	<code>xml_node_get_int(xmlNode *node, const xmlChar *prop, int *error_code)</code>	223
5.23.2.16	<code>xml_node_get_uint(xmlNode *node, const xmlChar *prop, int *error_code)</code>	223
5.23.2.17	<code>xml_node_get_uint_with_default(xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)</code>	224
5.23.2.18	<code>xml_node_set_float(xmlNode *node, const xmlChar *prop, double value)</code>	224
5.23.2.19	<code>xml_node_set_int(xmlNode *node, const xmlChar *prop, int value)</code>	225
5.23.2.20	<code>xml_node_set_uint(xmlNode *node, const xmlChar *prop, unsigned int value)</code>	225
5.24	<code>utils.h</code>	225
5.25	<code>variable.c</code> File Reference	227
5.25.1	Detailed Description	227
5.25.2	Function Documentation	228
5.25.2.1	<code>variable_error(Variable *variable, char *message)</code>	228
5.25.2.2	<code>variable_free(Variable *variable, unsigned int type)</code>	228
5.25.2.3	<code>variable_new(Variable *variable)</code>	228
5.25.2.4	<code>variable_open_json(Variable *variable, JsonNode *node, unsigned int algorithm, unsigned int nsteps)</code>	229
5.25.2.5	<code>variable_open_xml(Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)</code>	231
5.25.3	Variable Documentation	233

5.25.3.1	format	233
5.25.3.2	precision	233
5.26	variable.c	234
5.27	variable.h File Reference	238
5.27.1	Detailed Description	239
5.27.2	Enumeration Type Documentation	239
5.27.2.1	Algorithm	239
5.27.3	Function Documentation	239
5.27.3.1	variable_error(Variable *variable, char *message)	239
5.27.3.2	variable_free(Variable *variable, unsigned int type)	240
5.27.3.3	variable_new(Variable *variable)	240
5.27.3.4	variable_open_json(Variable *variable, JsonNode *node, unsigned int algorithm, unsigned int nsteps)	241
5.27.3.5	variable_open_xml(Variable *variable, xmlNode *node, unsigned int algorithm, unsigned int nsteps)	243
5.28	variable.h	245
Index		247

Chapter 1

MPCOTool

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

VERSIONS

- 3.0.3: Stable and recommended version.
- 3.1.3: Developing version to do new features.

AUTHORS

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

TOOLS AND LIBRARIES REQUIRED TO BUILD THE EXECUTABLE

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

OPTIONAL TOOLS AND LIBRARIES

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

FILES

The source code has to have the following files:

- 3.0.3/configure.ac: configure generator.
- 3.0.3/Makefile.in: Makefile generator.
- 3.0.3/config.h.in: config header generator.
- 3.0.3/mpcotool.c: main source code.
- 3.0.3/mpcotool.h: main header code.
- 3.0.3/mpcotool.ico: icon file.
- 3.0.3/interface.h: interface header code.
- 3.0.3/build: script to build all.
- 3.0.3/logo.png: logo figure.
- 3.0.3/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 4.4

Dyson Illumos

FreeBSD 10.3

Linux Mint DE 2

NetBSD 7.0

OpenSUSE Linux 13

Ubuntu Linux 16.04

1. Download the latest `genetic` doing on a terminal:

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

2. Download this repository:

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

3. Link the latest genetic version to genetic:

```
$ cd mpcotool/3.0.3
```

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

4. Build doing on a terminal:

```
$ ./build
```

OpenBSD 5.9

1. Select adequate versions:

```
$ export AUTOCONF_VERSION=2.69 AUTOMAKE_VERSION=1.15
```

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

Microsoft Windows 7

Microsoft Windows 8.1

1. Install `MSYS2` and the required libraries and utilities. You can follow detailed instructions in `install-unix`

2. Then, in a MSYS2 terminal, follow steps 1 to 4 of the previous Debian 8 section.

3. Optional Windows binary package can be built doing in the terminal:

```
$ make windist
```

Fedora Linux 24

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.

Building no-GUI version on servers

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

```
$ ./build_without_gui
```

MAKING MANUALS INSTRUCTIONS

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

```
$ make manuals
```

MAKING TESTS INSTRUCTIONS

In order to build the tests follow the next instructions:

1. Link some tests that needs genetic library doing in a terminal (assuming that you are in the directory `mpcotool/3.0.3`):

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

2. Build all tests doing in the same terminal:

```
$ cd ../../3.0.3
$ 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:

```

00001 <?xml version="1.0"?>
00002 <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">
00003   <experiment name="data_file_1" template1="template_1_1" template2="template_1_2" ... weight="weight_1"/
00004   >
00005   <experiment name="data_file_N" template1="template_N_1" template2="template_N_2" ... weight="weight_N"/
00006   >
00007   <variable name="variable_1" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps
      ="sweeps_number" nbits="bits_number" step="step_size"/>
00008   ...
00009   <variable name="variable_M" minimum="min_value" maximum="max_value" precision="precision_digits" sweeps
      ="sweeps_number" nbits="bits_number" step="step_size"/>
00010 </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:


```

00001 {
00002     "simulator": "simulator_name",
00003     "evaluator": "evaluator_name",
00004     "algorithm": "algorithm_type",
00005     "nsimulations": "simulations_number",
00006     "niterations": "iterations_number",
00007     "tolerance": "tolerance_value",
00008     "nbest": "best_number",
00009     "npopulation": "population_number",
00010     "ngenerations": "generations_number",
00011     "mutation": "mutation_ratio",
00012     "reproduction": "reproduction_ratio",
00013     "adaptation": "adaptation_ratio",
00014     "direction": "direction_search_type",
00015     "nsteps": "steps_number",
00016     "relaxation": "relaxation_parameter",
00017     "nestimates": "estimates_number",
00018     "threshold": "threshold_parameter",
00019     "norm": "norm_type",
00020     "p": "p_parameter",
00021     "seed": "random_seed",
00022     "result_file": "result_file",
00023     "variables_file": "variables_file",
00024     "experiments":
00025     [
00026         {
00027             "name": "data_file_1",
00028             "template1": "template_1_1",
00029             "template2": "template_1_2",
00030             ...
00031             "weight": "weight_1",
00032         },
00033         ...
00034         {
00035             "name": "data_file_N",
00036             "template1": "template_N_1",
00037             "template2": "template_N_2",
00038             ...
00039             "weight": "weight_N",
00040         }
00041     ],
00042     "variables":
00043     [
00044         {
00045             "name": "variable_1",
00046             "minimum": "min_value",
00047             "maximum": "max_value",
00048             "precision": "precision_digits",
00049             "sweeps": "sweeps_number",
00050             "nbits": "bits_number",
00051             "step": "step_size",
00052         },
00053         ...
00054         {
00055             "name": "variable_M",
00056             "minimum": "min_value",
00057             "maximum": "max_value",
00058             "precision": "precision_digits",
00059             "sweeps": "sweeps_number",
00060             "nbits": "bits_number",
00061             "step": "step_size",
00062         }
00063     ]
00064 ]
00065 }

```

SOME EXAMPLES OF INPUT FILES

Example 1

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

```
$ ./pivot input_file output_file
```
- The program to evaluate the objective function is: *compare*
- The syntax is:

```
$ ./compare simulated_file data_file result_file
```

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

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

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

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

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

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

- Then, the number of simulations to run is: $4 \times 5 \times 5 \times 5 = 2500$.
- The input file is:

```
00001 <?xml version="1.0"?>
00002 <optimize simulator="pivot" evaluator="compare" algorithm="sweep">
00003   <experiment name="27-48.txt" template1="template1.js"/>
00004   <experiment name="42.txt" template1="template2.js"/>
00005   <experiment name="52.txt" template1="template3.js"/>
00006   <experiment name="100.txt" template1="template4.js"/>
00007   <variable name="alpha1" minimum="179.70" maximum="180.20" precision="2" nsweeps="5"/>
00008   <variable name="alpha2" minimum="179.30" maximum="179.60" precision="2" nsweeps="5"/>
00009   <variable name="random" minimum="0.00" maximum="0.20" precision="2" nsweeps="5"/>
00010   <variable name="boot-time" minimum="0.0" maximum="3.0" precision="1" nsweeps="5"/>
00011 </optimize>
```

- A template file as *template1.js*:

```
00001 {
00002   "towers" :
00003   [
00004     {
00005       "length"      : 50.11,
00006       "velocity"    : 0.02738,
00007       "@variable1@" : @value1@,
00008       "@variable2@" : @value2@,
00009       "@variable3@" : @value3@,
00010       "@variable4@" : @value4@
00011     },
00012     {
00013       "length"      : 50.11,
00014       "velocity"    : 0.02824,
00015       "@variable1@" : @value1@,
00016       "@variable2@" : @value2@,
00017       "@variable3@" : @value3@,
00018       "@variable4@" : @value4@
00019     },
00020     {
00021       "length"      : 50.11,
00022       "velocity"    : 0.03008,
00023       "@variable1@" : @value1@,
00024       "@variable2@" : @value2@,
00025       "@variable3@" : @value3@,
00026       "@variable4@" : @value4@
00027     },
00028   ]
}
```

```

00029     "length"      : 50.11,
00030     "velocity"     : 0.03753,
00031     "@variable1@"  : @value1@,
00032     "@variable2@"  : @value2@,
00033     "@variable3@"  : @value3@,
00034     "@variable4@"  : @value4@
00035   }
00036 ],
00037 "cycle-time"      : 71.0,
00038 "plot-time"       : 1.0,
00039 "comp-time-step"  : 0.1,
00040 "active-percent"  : 27.48
00041 }

```

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

```

00001 {
00002   "towers" :
00003   [
00004     {
00005       "length"      : 50.11,
00006       "velocity"     : 0.02738,
00007       "alpha1"      : 179.95,
00008       "alpha2"      : 179.45,
00009       "random"       : 0.10,
00010       "boot-time"    : 1.5
00011     },
00012     {
00013       "length"      : 50.11,
00014       "velocity"     : 0.02824,
00015       "alpha1"      : 179.95,
00016       "alpha2"      : 179.45,
00017       "random"       : 0.10,
00018       "boot-time"    : 1.5
00019     },
00020     {
00021       "length"      : 50.11,
00022       "velocity"     : 0.03008,
00023       "alpha1"      : 179.95,
00024       "alpha2"      : 179.45,
00025       "random"       : 0.10,
00026       "boot-time"    : 1.5
00027     },
00028     {
00029       "length"      : 50.11,
00030       "velocity"     : 0.03753,
00031       "alpha1"      : 179.95,
00032       "alpha2"      : 179.45,
00033       "random"       : 0.10,
00034       "boot-time"    : 1.5
00035     }
00036   ],
00037   "cycle-time"      : 71.0,
00038   "plot-time"       : 1.0,
00039   "comp-time-step"  : 0.1,
00040   "active-percent"  : 27.48
00041 }

```


Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

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

Chapter 3

File Index

3.1 File List

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

config.h	Configuration header file	29
experiment.c	Source file to define the experiment data	30
experiment.h	Header file to define the experiment data	40
generate.c	??
input.c	Source file to define the input functions	46
input.h	Header file to define the input functions	70
interface.c	Source file to define the graphical interface functions	85
interface.h	Header file to define the graphical interface functions	131
main.c	Main source file	143
optimize.c	Source file to define the optimization functions	147
optimize.h	Header file to define the optimization functions	180
utils.c	Source file to define some useful functions	197
utils.h	Header file to define some useful functions	213
variable.c	Source file to define the variable data	227
variable.h	Header file to define the variable data	238

Chapter 4

Data Structure Documentation

4.1 Experiment Struct Reference

Struct to define the experiment data.

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

Data Fields

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

4.1.1 Detailed Description

Struct to define the experiment data.

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

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

- [experiment.h](#)

4.2 Input Struct Reference

Struct to define the optimization input file.

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

Collaboration diagram for Input:

Data Fields

- [Experiment](#) * [experiment](#)
Array or experiments.
- [Variable](#) * [variable](#)
Array of variables.
- char * [result](#)
Name of the result file.
- char * [variables](#)
Name of the variables file.
- char * [simulator](#)
Name of the simulator program.
- char * [evaluator](#)
Name of the program to evaluate the objective function.
- char * [directory](#)
Working directory.
- char * [name](#)
Input data file name.
- double [tolerance](#)
Algorithm tolerance.
- double [mutation_ratio](#)
Mutation probability.
- double [reproduction_ratio](#)
Reproduction probability.
- double [adaptation_ratio](#)
Adaptation probability.
- double [relaxation](#)
Relaxation parameter.
- double [p](#)
Exponent of the P error norm.
- double [threshold](#)
Threshold to finish the optimization.
- unsigned long int [seed](#)
Seed of the pseudo-random numbers generator.
- unsigned int [nvariables](#)
Variables number.
- unsigned int [nexperiments](#)
Experiments number.
- unsigned int [nsimulations](#)
Simulations number per experiment.
- unsigned int [algorithm](#)
Algorithm type.
- unsigned int [nsteps](#)
Number of steps to do the direction search method.
- unsigned int [direction](#)
Method to estimate the direction search.
- unsigned int [nestimates](#)
Number of simulations to estimate the direction search.
- unsigned int [niterations](#)
Number of algorithm iterations.
- unsigned int [nbest](#)

- unsigned int [norm](#)
Number of best simulations.
- unsigned int [type](#)
Error norm type.
- unsigned int [type](#)
Type of input file.

4.2.1 Detailed Description

Struct to define the optimization input file.

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

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

- [input.h](#)

4.3 Optimize Struct Reference

Struct to define the optimization ation data.

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

Data Fields

- GMappedFile ** [file](#) [[MAX_NINPUTS](#)]
Matrix of input template files.
- char ** [experiment](#)
Array of experimental data file names.
- char ** [label](#)
Array of variable names.
- gsl_rng * [rng](#)
GSL random number generator.
- GeneticVariable * [genetic_variable](#)
Array of variables for the genetic algorithm.
- FILE * [file_result](#)
Result file.
- FILE * [file_variables](#)
Variables file.
- char * [result](#)
Name of the result file.
- char * [variables](#)
Name of the variables file.
- char * [simulator](#)
Name of the simulator program.
- char * [evaluator](#)
Name of the program to evaluate the objective function.
- double * [value](#)
Array of variable values.

- double * [rangemin](#)
Array of minimum variable values.
- double * [rangemax](#)
Array of maximum variable values.
- double * [rangeminabs](#)
Array of absolute minimum variable values.
- double * [rangemaxabs](#)
Array of absolute maximum variable values.
- double * [error_best](#)
Array of the best minimum errors.
- double * [weight](#)
Array of the experiment weights.
- double * [step](#)
Array of direction search method step sizes.
- double * [direction](#)
Vector of direction search estimation.
- double * [value_old](#)
Array of the best variable values on the previous step.
- double * [error_old](#)
Array of the best minimum errors on the previous step.
- unsigned int * [precision](#)
Array of variable precisions.
- unsigned int * [nsweeps](#)
Array of sweeps of the sweep algorithm.
- unsigned int * [nbits](#)
Array of bits number of the genetic algorithm.
- unsigned int * [thread](#)
Array of simulation numbers to calculate on the thread.
- unsigned int * [thread_direction](#)
- unsigned int * [simulation_best](#)
Array of best simulation numbers.
- double [tolerance](#)
Algorithm tolerance.
- double [mutation_ratio](#)
Mutation probability.
- double [reproduction_ratio](#)
Reproduction probability.
- double [adaptation_ratio](#)
Adaptation probability.
- double [relaxation](#)
Relaxation parameter.
- double [calculation_time](#)
Calculation time.
- double [p](#)
Exponent of the P error norm.
- double [threshold](#)
Threshold to finish the optimization.
- unsigned long int [seed](#)
Seed of the pseudo-random numbers generator.
- unsigned int [nvariables](#)
Variables number.

- unsigned int [nexperiments](#)
Experiments number.
- unsigned int [ninputs](#)
Number of input files to the simulator.
- unsigned int [nsimulations](#)
Simulations number per experiment.
- unsigned int [nsteps](#)
Number of steps for the direction search method.
- unsigned int [nestimates](#)
Number of simulations to estimate the direction.
- unsigned int [algorithm](#)
Algorithm type.
- unsigned int [nstart](#)
Beginning simulation number of the task.
- unsigned int [nend](#)
Ending simulation number of the task.
- unsigned int [nstart_direction](#)
Beginning simulation number of the task for the direction search method.
- unsigned int [nend_direction](#)
Ending simulation number of the task for the direction search method.
- unsigned int [niterations](#)
Number of algorithm iterations.
- unsigned int [nbest](#)
Number of best simulations.
- unsigned int [nsaveds](#)
Number of saved simulations.
- unsigned int [stop](#)
To stop the simulations.
- int [mpi_rank](#)
Number of MPI task.

4.3.1 Detailed Description

Struct to define the optimization ation data.

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

4.3.2 Field Documentation

4.3.2.1 unsigned int* Optimize::thread_direction

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

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

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

- [optimize.h](#)

4.4 Options Struct Reference

Struct to define the options dialog.

```
#include <interface.h>
```

Data Fields

- `GtkDialog * dialog`
Main GtkDialog.
- `GtkGrid * grid`
Main GtkGrid.
- `GtkLabel * label_seed`
Pseudo-random numbers generator seed GtkLabel.
- `GtkSpinButton * spin_seed`
Pseudo-random numbers generator seed GtkSpinButton.
- `GtkLabel * label_threads`
Threads number GtkLabel.
- `GtkSpinButton * spin_threads`
Threads number GtkSpinButton.
- `GtkLabel * label_direction`
Direction threads number GtkLabel.
- `GtkSpinButton * spin_direction`
Direction threads number GtkSpinButton.

4.4.1 Detailed Description

Struct to define the options dialog.

Definition at line 48 of file [interface.h](#).

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

- [interface.h](#)

4.5 ParallelData Struct Reference

Struct to pass to the GThreads parallelized function.

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

Data Fields

- `unsigned int thread`
Thread number.

4.5.1 Detailed Description

Struct to pass to the GThreads parallelized function.

Definition at line 122 of file [optimize.h](#).

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

- [optimize.h](#)

4.6 Running Struct Reference

Struct to define the running dialog.

```
#include <interface.h>
```

Data Fields

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

4.6.1 Detailed Description

Struct to define the running dialog.

Definition at line 67 of file [interface.h](#).

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

- [interface.h](#)

4.7 Variable Struct Reference

Struct to define the variable data.

```
#include <variable.h>
```

Data Fields

- char * [name](#)
Variable name.
- double [rangemin](#)
Minimum variable value.
- double [rangemax](#)
Maximum variable value.
- double [rangeminabs](#)
Absolute minimum variable value.
- double [rangemaxabs](#)
Absolute maximum variable value.
- double [step](#)
Direction search method step size.
- unsigned int [precision](#)
Variable precision.
- unsigned int [nsweeps](#)
Sweeps of the sweep algorithm.
- unsigned int [nbits](#)
Bits number of the genetic algorithm.

4.7.1 Detailed Description

Struct to define the variable data.

Definition at line 56 of file [variable.h](#).

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

- [variable.h](#)

4.8 Window Struct Reference

Struct to define the main window.

```
#include <interface.h>
```

Collaboration diagram for Window:

Data Fields

- GtkWidget * [window](#)
Main GtkWidget.
- GtkWidget * [grid](#)
Main GtkWidget.
- GtkWidget * [bar_buttons](#)
GtkToolbar to store the main buttons.
- GtkWidget * [button_open](#)
Open GtkWidget.
- GtkWidget * [button_save](#)
Save GtkWidget.
- GtkWidget * [button_run](#)
Run GtkWidget.
- GtkWidget * [button_options](#)
Options GtkWidget.
- GtkWidget * [button_help](#)
Help GtkWidget.
- GtkWidget * [button_about](#)
Help GtkWidget.
- GtkWidget * [button_exit](#)
Exit GtkWidget.
- GtkWidget * [grid_files](#)
Files GtkWidget.
- GtkWidget * [label_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [button_simulator](#)
Simulator program GtkWidget.
- GtkWidget * [check_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [button_evaluator](#)
Evaluator program GtkWidget.
- GtkWidget * [label_result](#)
Result file GtkWidget.
- GtkWidget * [entry_result](#)
Result file GtkWidget.
- GtkWidget * [label_variables](#)
Variables file GtkWidget.
- GtkWidget * [entry_variables](#)
Variables file GtkWidget.
- GtkWidget * [frame_norm](#)
GtkFrame to set the error norm.
- GtkWidget * [grid_norm](#)
GtkWidget to set the error norm.
- GtkWidget * [button_norm](#) [NNORMS]
Array of GtkWidget to set the error norm.
- GtkWidget * [label_p](#)
GtkWidget to set the p parameter.
- GtkWidget * [spin_p](#)
GtkSpinButton to set the p parameter.
- GtkWidget * [scrolled_p](#)

- GtkScrolledWindow to set the p parameter.*

 - GtkFrame * [frame_algorithm](#)
GtkFrame to set the algorithm.
 - GtkGrid * [grid_algorithm](#)
GtkGrid to set the algorithm.
 - GtkRadioButton * [button_algorithm](#) [NALGORITHMS]
Array of GtkButtons to set the algorithm.
 - GtkLabel * [label_simulations](#)
GtkLabel to set the simulations number.
 - GtkSpinButton * [spin_simulations](#)
GtkSpinButton to set the simulations number.
 - GtkLabel * [label_iterations](#)
GtkLabel to set the iterations number.
 - GtkSpinButton * [spin_iterations](#)
GtkSpinButton to set the iterations number.
 - GtkLabel * [label_tolerance](#)
GtkLabel to set the tolerance.
 - GtkSpinButton * [spin_tolerance](#)
GtkSpinButton to set the tolerance.
 - GtkLabel * [label_best](#)
GtkLabel to set the best number.
 - GtkSpinButton * [spin_best](#)
GtkSpinButton to set the best number.
 - GtkLabel * [label_population](#)
GtkLabel to set the population number.
 - GtkSpinButton * [spin_population](#)
GtkSpinButton to set the population number.
 - GtkLabel * [label_generations](#)
GtkLabel to set the generations number.
 - GtkSpinButton * [spin_generations](#)
GtkSpinButton to set the generations number.
 - GtkLabel * [label_mutation](#)
GtkLabel to set the mutation ratio.
 - GtkSpinButton * [spin_mutation](#)
GtkSpinButton to set the mutation ratio.
 - GtkLabel * [label_reproduction](#)
GtkLabel to set the reproduction ratio.
 - GtkSpinButton * [spin_reproduction](#)
GtkSpinButton to set the reproduction ratio.
 - GtkLabel * [label_adaptation](#)
GtkLabel to set the adaptation ratio.
 - GtkSpinButton * [spin_adaptation](#)
GtkSpinButton to set the adaptation ratio.
 - GtkCheckButton * [check_direction](#)
GtkCheckButton to check running the direction search method.
 - GtkGrid * [grid_direction](#)
GtkGrid to pack the direction search method widgets.
 - GtkRadioButton * [button_direction](#) [NDIRECTIONS]
GtkRadioButtons array to set the direction estimate method.
 - GtkLabel * [label_steps](#)
GtkLabel to set the steps number.

- GtkSpinButton * [spin_steps](#)
GtkSpinButton to set the steps number.
- GtkLabel * [label_estimates](#)
GtkLabel to set the estimates number.
- GtkSpinButton * [spin_estimates](#)
GtkSpinButton to set the estimates number.
- GtkLabel * [label_relaxation](#)
GtkLabel to set the relaxation parameter.
- GtkSpinButton * [spin_relaxation](#)
GtkSpinButton to set the relaxation parameter.
- GtkLabel * [label_threshold](#)
GtkLabel to set the threshold.
- GtkSpinButton * [spin_threshold](#)
GtkSpinButton to set the threshold.
- GtkScrolledWindow * [scrolled_threshold](#)
GtkScrolledWindow to set the threshold.
- GtkFrame * [frame_variable](#)
Variable GtkFrame.
- GtkGrid * [grid_variable](#)
Variable GtkGrid.
- GtkComboBoxText * [combo_variable](#)
GtkComboBoxEntry to select a variable.
- GtkButton * [button_add_variable](#)
GtkButton to add a variable.
- GtkButton * [button_remove_variable](#)
GtkButton to remove a variable.
- GtkLabel * [label_variable](#)
Variable GtkLabel.
- GtkEntry * [entry_variable](#)
GtkEntry to set the variable name.
- GtkLabel * [label_min](#)
Minimum GtkLabel.
- GtkSpinButton * [spin_min](#)
Minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_min](#)
Minimum GtkScrolledWindow.
- GtkLabel * [label_max](#)
Maximum GtkLabel.
- GtkSpinButton * [spin_max](#)
Maximum GtkSpinButton.
- GtkScrolledWindow * [scrolled_max](#)
Maximum GtkScrolledWindow.
- GtkCheckButton * [check_minabs](#)
Absolute minimum GtkCheckButton.
- GtkSpinButton * [spin_minabs](#)
Absolute minimum GtkSpinButton.
- GtkScrolledWindow * [scrolled_minabs](#)
Absolute minimum GtkScrolledWindow.
- GtkCheckButton * [check_maxabs](#)
Absolute maximum GtkCheckButton.
- GtkSpinButton * [spin_maxabs](#)

- Absolute maximum GtkSpinButton.*
- GtkScrolledWindow * [scrolled_maxabs](#)
 - Absolute maximum GtkScrolledWindow.*
- GtkLabel * [label_precision](#)
 - Precision GtkLabel.*
- GtkSpinButton * [spin_precision](#)
 - Precision digits GtkSpinButton.*
- GtkLabel * [label_sweeps](#)
 - Sweeps number GtkLabel.*
- GtkSpinButton * [spin_sweeps](#)
 - Sweeps number GtkSpinButton.*
- GtkLabel * [label_bits](#)
 - Bits number GtkLabel.*
- GtkSpinButton * [spin_bits](#)
 - Bits number GtkSpinButton.*
- GtkLabel * [label_step](#)
 - GtkLabel to set the step.*
- GtkSpinButton * [spin_step](#)
 - GtkSpinButton to set the step.*
- GtkScrolledWindow * [scrolled_step](#)
 - step GtkScrolledWindow.*
- GtkFrame * [frame_experiment](#)
 - Experiment GtkFrame.*
- GtkGrid * [grid_experiment](#)
 - Experiment GtkGrid.*
- GtkComboBoxText * [combo_experiment](#)
 - Experiment GtkComboBoxEntry.*
- GtkButton * [button_add_experiment](#)
 - GtkButton to add a experiment.*
- GtkButton * [button_remove_experiment](#)
 - GtkButton to remove a experiment.*
- GtkLabel * [label_experiment](#)
 - Experiment GtkLabel.*
- GtkFileChooserButton * [button_experiment](#)
 - GtkFileChooserButton to set the experimental data file.*
- GtkLabel * [label_weight](#)
 - Weight GtkLabel.*
- GtkSpinButton * [spin_weight](#)
 - Weight GtkSpinButton.*
- GtkCheckButton * [check_template](#) [MAX_NINPUTS]
 - Array of GtkCheckButtons to set the input templates.*
- GtkFileChooserButton * [button_template](#) [MAX_NINPUTS]
 - Array of GtkFileChooserButtons to set the input templates.*
- GdkPixbuf * [logo](#)
 - Logo GdkPixbuf.*
- [Experiment](#) * [experiment](#)
 - Array of experiments data.*
- [Variable](#) * [variable](#)
 - Array of variables data.*
- char * [application_directory](#)
 - Application directory.*

- gulong [id_experiment](#)
Identifier of the combo_experiment signal.
- gulong [id_experiment_name](#)
Identifier of the button_experiment signal.
- gulong [id_variable](#)
Identifier of the combo_variable signal.
- gulong [id_variable_label](#)
Identifier of the entry_variable signal.
- gulong [id_template](#) [MAX_NINPUTS]
Array of identifiers of the check_template signal.
- gulong [id_input](#) [MAX_NINPUTS]
Array of identifiers of the button_template signal.
- unsigned int [nexperiments](#)
Number of experiments.
- unsigned int [nvariables](#)
Number of variables.

4.8.1 Detailed Description

Struct to define the main window.

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

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

- [interface.h](#)

Chapter 5

File Documentation

5.1 config.h File Reference

Configuration header file.

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

5.2 config.h

```
00001 /* config.h.  Generated from config.h.in by configure.  */
00002 /*
00003 MPCOTool:
00004 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00005 calibrations or optimizations of empirical parameters.
00006
00007 AUTHORS: Javier Burguete and Borja Latorre.
00008
00009 Copyright 2012-2016, AUTHORS.
00010
00011 Redistribution and use in source and binary forms, with or without modification,
00012 are permitted provided that the following conditions are met:
00013
00014     1. Redistributions of source code must retain the above copyright notice,
00015        this list of conditions and the following disclaimer.
00016
00017     2. Redistributions in binary form must reproduce the above copyright notice,
00018        this list of conditions and the following disclaimer in the
00019        documentation and/or other materials provided with the distribution.
00020
00021 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00022 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00023 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00024 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00025 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00027 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00028 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00029 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00030 OF SUCH DAMAGE.
00031 */
00032
00033 #ifndef CONFIG__H
00034 #define CONFIG__H 1
00035
00036 // Array sizes
00037
00038 #define MAX_NINPUTS 8
00039 #define NALGORITHMS 3
00040 #define NDIRECTIONS 2
00041 #define NNORMS 4
00042 #define NPRECISIONS 15
00043
```

```

00051 // Default choices
00052
00053 #define DEFAULT_PRECISION (NPRECISIONS - 1)
00054 #define DEFAULT_RANDOM_SEED 7007
00055 #define DEFAULT_RELAXATION 1.
00056
00057 // Interface labels
00058
00059 #define LOCALE_DIR "locales"
00060 #define PROGRAM_INTERFACE "mpcotool"
00061
00062 // Labels
00063
00064 #define LABEL_ABSOLUTE_MINIMUM "absolute_minimum"
00065 #define LABEL_ABSOLUTE_MAXIMUM "absolute_maximum"
00066
00067 #define LABEL_ADAPTATION "adaptation"
00068 #define LABEL_ALGORITHM "algorithm"
00069 #define LABEL_OPTIMIZE "optimize"
00070 #define LABEL_COORDINATES "coordinates"
00071 #define LABEL_DIRECTION "direction"
00072 #define LABEL_EUCLIDIAN "euclidian"
00073 #define LABEL_EVALUATOR "evaluator"
00074 #define LABEL_EXPERIMENT "experiment"
00075 #define LABEL_EXPERIMENTS "experiments"
00076 #define LABEL_GENETIC "genetic"
00077 #define LABEL_MINIMUM "minimum"
00078 #define LABEL_MAXIMUM "maximum"
00079 #define LABEL_MONTE_CARLO "Monte-Carlo"
00080 #define LABEL_MUTATION "mutation"
00081 #define LABEL_NAME "name"
00082 #define LABEL_NBEST "nbest"
00083 #define LABEL_NBITS "nbits"
00084 #define LABEL_NESTIMATES "nestimates"
00085 #define LABEL_NGENERATIONS "ngenerations"
00086 #define LABEL_NITERATIONS "niterations"
00087 #define LABEL_NORM "norm"
00088 #define LABEL_NPOPULATION "npopulation"
00089 #define LABEL_NSIMULATIONS "nsimulations"
00090 #define LABEL_NSTEPS "nsteps"
00091 #define LABEL_NSWEEPS "nsweeps"
00092 #define LABEL_P "p"
00093 #define LABEL_PRECISION "precision"
00094 #define LABEL_RANDOM "random"
00095 #define LABEL_RELAXATION "relaxation"
00096 #define LABEL_REPRODUCTION "reproduction"
00097 #define LABEL_RESULT_FILE "result_file"
00098 #define LABEL_SIMULATOR "simulator"
00099 #define LABEL_SEED "seed"
00100 #define LABEL_STEP "step"
00101 #define LABEL_SWEEP "sweep"
00102 #define LABEL_TAXICAB "taxicab"
00103 #define LABEL_TEMPLATE1 "template1"
00104 #define LABEL_TEMPLATE2 "template2"
00105 #define LABEL_TEMPLATE3 "template3"
00106 #define LABEL_TEMPLATE4 "template4"
00107 #define LABEL_TEMPLATE5 "template5"
00108 #define LABEL_TEMPLATE6 "template6"
00109 #define LABEL_TEMPLATE7 "template7"
00110 #define LABEL_TEMPLATE8 "template8"
00111 #define LABEL_THRESHOLD "threshold"
00112 #define LABEL_TOLERANCE "tolerance"
00113 #define LABEL_VARIABLE "variable"
00114 #define LABEL_VARIABLES "variables"
00115 #define LABEL_VARIABLES_FILE "variables_file"
00116 #define LABEL_WEIGHT "weight"
00117
00118 // Enumerations
00119
00120
00125 enum INPUT_TYPE
00126 {
00127     INPUT_TYPE_XML = 0,
00128     INPUT_TYPE_JSON = 1
00129 };
00130
00131 #endif

```

5.3 experiment.c File Reference

Source file to define the experiment data.


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

Include dependency graph for experiment.c:

Macros

- `#define _GNU_SOURCE`
- `#define DEBUG_EXPERIMENT 0`
Macro to debug experiment functions.

Functions

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

Variables

- const char * `template` [`MAX_NINPUTS`]
Array of xmlChar strings with template labels.

5.3.1 Detailed Description

Source file to define the experiment data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file `experiment.c`.

5.3.2 Function Documentation

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	Error message.

Definition at line 121 of file [experiment.c](#).

```

00122 {
00123     char buffer[64];
00124     if (!experiment->name)
00125         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00128                 message);
00129     error_message = g_strdup (buffer);
00130 }
```

5.3.2.2 void experiment_free ([Experiment](#) * *experiment*, unsigned int *type*)

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

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	Type of input file.

Definition at line 88 of file [experiment.c](#).

```

00089 {
00090     unsigned int i;
00091     #if DEBUG_EXPERIMENT
00092     fprintf (stderr, "experiment_free: start\n");
00093     #endif
00094     if (type == INPUT_TYPE_XML)
00095     {
00096         for (i = 0; i < experiment->ninputs; ++i)
00097             xmlFree (experiment->template[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->template[i]);
00104         g_free (experiment->name);
00105     }
00106     experiment->ninputs = 0;
00107     #if DEBUG_EXPERIMENT
00108     fprintf (stderr, "experiment_free: end\n");
00109     #endif
00110 }
```

5.3.2.3 void experiment_new ([Experiment](#) * *experiment*)

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

Parameters

<i>experiment</i>	Experiment struct.
-------------------	------------------------------------

Definition at line 64 of file [experiment.c](#).

```

00065 {
00066     unsigned int i;
00067     #if DEBUG_EXPERIMENT
00068         fprintf (stderr, "experiment_new: start\n");
00069     #endif
00070     experiment->name = NULL;
00071     experiment->ninputs = 0;
00072     for (i = 0; i < MAX_NINPUTS; ++i)
00073         experiment->template[i] = NULL;
00074     #if DEBUG_EXPERIMENT
00075         fprintf (stderr, "input_new: end\n");
00076     #endif
00077 }

```

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	JSON node.
<i>ninputs</i>	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 252 of file [experiment.c](#).

```

00254 {
00255     char buffer[64];
00256     JsonObject *object;
00257     const char *name;
00258     int error_code;
00259     unsigned int i;
00260
00261     #if DEBUG_EXPERIMENT
00262         fprintf (stderr, "experiment_open_json: start\n");
00263     #endif
00264
00265     // Resetting experiment data
00266     experiment_new (experiment);
00267
00268     // Getting JSON object
00269     object = json_node_get_object (node);
00270
00271     // Reading the experimental data
00272     name = json_object_get_string_member (object, LABEL_NAME);
00273     if (!name)
00274     {
00275         experiment_error (experiment, gettext ("no data file name"));
00276         goto exit_on_error;
00277     }
00278     experiment->name = g_strdup (name);
00279     #if DEBUG_EXPERIMENT
00280         fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00281     #endif
00282     experiment->weight
00283     = json_object_get_float_with_default (object,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291 }

```

```

00289     }
00290     #if DEBUG_EXPERIMENT
00291     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00292     #endif
00293     name = json_object_get_string_member (object, template[0]);
00294     if (name)
00295     {
00296     #if DEBUG_EXPERIMENT
00297     fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00298             name, template[0]);
00299     #endif
00300     ++experiment->ninputs;
00301     }
00302     else
00303     {
00304     experiment_error (experiment, gettext ("no template"));
00305     goto exit_on_error;
00306     }
00307     experiment->template[0] = g_strdup (name);
00308     for (i = 1; i < MAX_NINPUTS; ++i)
00309     {
00310     #if DEBUG_EXPERIMENT
00311     fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00312     #endif
00313     if (json_object_get_member (object, template[i]))
00314     {
00315     if (ninputs && ninputs <= i)
00316     {
00317     experiment_error (experiment, gettext ("bad templates number"));
00318     goto exit_on_error;
00319     }
00320     name = json_object_get_string_member (object, template[i]);
00321     #if DEBUG_EXPERIMENT
00322     fprintf (stderr,
00323             "experiment_open_json: experiment=%s template%u=%s\n",
00324             experiment->nexperiments, name, template[i]);
00325     #endif
00326     experiment->template[i] = g_strdup (name);
00327     ++experiment->ninputs;
00328     }
00329     else if (ninputs && ninputs > i)
00330     {
00331     snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00332     experiment_error (experiment, buffer);
00333     goto exit_on_error;
00334     }
00335     else
00336     break;
00337     }
00338     #if DEBUG_EXPERIMENT
00339     fprintf (stderr, "experiment_open_json: end\n");
00340     #endif
00341     return 1;
00342 }
00343
00344 exit_on_error:
00345 experiment_free (experiment, INPUT_TYPE_JSON);
00346 #if DEBUG_EXPERIMENT
00347 fprintf (stderr, "experiment_open_json: end\n");
00348 #endif
00349 return 0;
00350 }

```

Here is the call graph for this function:

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	XML node.
<i>ninputs</i>	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 145 of file [experiment.c](#).

```

00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153     fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, gettext ("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167     fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     = xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
00171                                     &error_code);
00172     if (error_code)
00173     {
00174         experiment_error (experiment, gettext ("bad weight"));
00175         goto exit_on_error;
00176     }
00177     #if DEBUG_EXPERIMENT
00178     fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00179     #endif
00180     experiment->template[0]
00181     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00182     if (experiment->template[0])
00183     {
00184         #if DEBUG_EXPERIMENT
00185         fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00186                 experiment->name, template[0]);
00187         #endif
00188         ++experiment->ninputs;
00189     }
00190     else
00191     {
00192         experiment_error (experiment, gettext ("no template"));
00193         goto exit_on_error;
00194     }
00195     for (i = 1; i < MAX_NINPUTS; ++i)
00196     {
00197         #if DEBUG_EXPERIMENT
00198         fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00199         #endif
00200         if (xmlHasProp (node, (const xmlChar *) template[i]))
00201         {
00202             if (ninputs && ninputs <= i)
00203             {
00204                 experiment_error (experiment, gettext ("bad templates number"));
00205                 goto exit_on_error;
00206             }
00207             experiment->template[i]
00208             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00209             #if DEBUG_EXPERIMENT
00210             fprintf (stderr, "experiment_open_xml: experiment=%s template%u=%s\n",
00211                     experiment->name, experiment->nexperiments, experiment->template[i]);
00212             #endif
00213             ++experiment->ninputs;
00214         }
00215         else if (ninputs && ninputs > i)
00216         {
00217             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00218             experiment_error (experiment, buffer);
00219             goto exit_on_error;
00220         }
00221     }
00222     else
00223         break;

```

```

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

```

Here is the call graph for this function:

5.3.3 Variable Documentation

5.3.3.1 `const char* template[MAX_NINPUTS]`

Initial value:

```

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

```

Array of xmlChar strings with template labels.

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

5.4 experiment.c

```

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

```

```

00039 #include "config.h"
00040 #include <stdio.h>
00041 #include <libxml/parser.h>
00042 #include <libintl.h>
00043 #include <glib.h>
00044 #include <json-glib/json-glib.h>
00045 #include "utils.h"
00046 #include "experiment.h"
00047
00048 #define DEBUG_EXPERIMENT 0
00049
00050 const char *template[MAX_NINPUTS] = {
00051     LABEL_TEMPLATE1, LABEL_TEMPLATE2,
00052     LABEL_TEMPLATE3, LABEL_TEMPLATE4,
00053     LABEL_TEMPLATE5, LABEL_TEMPLATE6,
00054     LABEL_TEMPLATE7, LABEL_TEMPLATE8
00055 };
00056
00057 void
00058 experiment_new (Experiment * experiment)
00059 {
00060     unsigned int i;
00061     #if DEBUG_EXPERIMENT
00062         fprintf (stderr, "experiment_new: start\n");
00063     #endif
00064     experiment->name = NULL;
00065     experiment->ninputs = 0;
00066     for (i = 0; i < MAX_NINPUTS; ++i)
00067         experiment->template[i] = NULL;
00068     #if DEBUG_EXPERIMENT
00069         fprintf (stderr, "input_new: end\n");
00070     #endif
00071 }
00072
00073 void
00074 experiment_free (Experiment * experiment, unsigned int type)
00075 {
00076     unsigned int i;
00077     #if DEBUG_EXPERIMENT
00078         fprintf (stderr, "experiment_free: start\n");
00079     #endif
00080     if (type == INPUT_TYPE_XML)
00081     {
00082         for (i = 0; i < experiment->ninputs; ++i)
00083             xmlFree (experiment->template[i]);
00084         xmlFree (experiment->name);
00085     }
00086     else
00087     {
00088         for (i = 0; i < experiment->ninputs; ++i)
00089             g_free (experiment->template[i]);
00090         g_free (experiment->name);
00091     }
00092     experiment->ninputs = 0;
00093     #if DEBUG_EXPERIMENT
00094         fprintf (stderr, "experiment_free: end\n");
00095     #endif
00096 }
00097
00098 void
00099 experiment_error (Experiment * experiment, char *message)
00100 {
00101     char buffer[64];
00102     if (!experiment->name)
00103         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00104     else
00105         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00106                 message);
00107     error_message = g_strdup (buffer);
00108 }
00109
00110 int
00111 experiment_open_xml (Experiment * experiment, xmlNode * node,
00112                     unsigned int ninputs)
00113 {
00114     char buffer[64];
00115     int error_code;
00116     unsigned int i;
00117
00118     #if DEBUG_EXPERIMENT
00119         fprintf (stderr, "experiment_open_xml: start\n");
00120     #endif
00121
00122     // Resetting experiment data
00123     experiment_new (experiment);
00124 }

```

```

00159 // Reading the experimental data
00160 experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161 if (!experiment->name)
00162 {
00163     experiment_error (experiment, gettext ("no data file name"));
00164     goto exit_on_error;
00165 }
00166 #if DEBUG_EXPERIMENT
00167 fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168 #endif
00169 experiment->weight
00170     = xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
00171                                     &error_code);
00172 if (error_code)
00173 {
00174     experiment_error (experiment, gettext ("bad weight"));
00175     goto exit_on_error;
00176 }
00177 #if DEBUG_EXPERIMENT
00178 fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00179 #endif
00180 experiment->template[0]
00181     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00182 if (experiment->template[0])
00183 {
00184     #if DEBUG_EXPERIMENT
00185     fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00186             experiment->name, template[0]);
00187     #endif
00188     ++experiment->ninputs;
00189 }
00190 else
00191 {
00192     experiment_error (experiment, gettext ("no template"));
00193     goto exit_on_error;
00194 }
00195 for (i = 1; i < MAX_NINPUTS; ++i)
00196 {
00197     #if DEBUG_EXPERIMENT
00198     fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00199     #endif
00200     if (xmlHasProp (node, (const xmlChar *) template[i]))
00201     {
00202         if (ninputs && ninputs <= i)
00203         {
00204             experiment_error (experiment, gettext ("bad templates number"));
00205             goto exit_on_error;
00206         }
00207         experiment->template[i]
00208             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00209         #if DEBUG_EXPERIMENT
00210         fprintf (stderr, "experiment_open_xml: experiment=%s template%u=%s\n",
00211                 experiment->name, experiment->nexperiments,
00212                 experiment->template[i]);
00213         #endif
00214         ++experiment->ninputs;
00215     }
00216     else if (ninputs && ninputs > i)
00217     {
00218         snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00219         experiment_error (experiment, buffer);
00220         goto exit_on_error;
00221     }
00222     else
00223     break;
00224 }
00225 #if DEBUG_EXPERIMENT
00227 fprintf (stderr, "experiment_open_xml: end\n");
00228 #endif
00229 return 1;
00230
00231 exit_on_error:
00232     experiment_free (experiment, INPUT_TYPE_XML);
00233     #if DEBUG_EXPERIMENT
00234     fprintf (stderr, "experiment_open_xml: end\n");
00235     #endif
00236     return 0;
00237 }
00238
00251 int
00252 experiment_open_json (Experiment * experiment, XmlNode * node,
00253                     unsigned int ninputs)
00254 {
00255     char buffer[64];
00256     JsonObject *object;

```



```

00257     const char *name;
00258     int error_code;
00259     unsigned int i;
00260
00261     #if DEBUG_EXPERIMENT
00262     fprintf (stderr, "experiment_open_json: start\n");
00263     #endif
00264
00265     // Resetting experiment data
00266     experiment_new (experiment);
00267
00268     // Getting JSON object
00269     object = json_node_get_object (node);
00270
00271     // Reading the experimental data
00272     name = json_object_get_string_member (object, LABEL_NAME);
00273     if (!name)
00274     {
00275         experiment_error (experiment, gettext ("no data file name"));
00276         goto exit_on_error;
00277     }
00278     experiment->name = g_strdup (name);
00279     #if DEBUG_EXPERIMENT
00280     fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00281     #endif
00282     experiment->weight
00283     = json_object_get_float_with_default (object,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291     #if DEBUG_EXPERIMENT
00292     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00293     #endif
00294     name = json_object_get_string_member (object, template[0]);
00295     if (name)
00296     {
00297         #if DEBUG_EXPERIMENT
00298         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00299         name, template[0]);
00300         #endif
00301         ++experiment->ninputs;
00302     }
00303     else
00304     {
00305         experiment_error (experiment, gettext ("no template"));
00306         goto exit_on_error;
00307     }
00308     experiment->template[0] = g_strdup (name);
00309     for (i = 1; i < MAX_NINPUTS; ++i)
00310     {
00311         #if DEBUG_EXPERIMENT
00312         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00313         #endif
00314         if (json_object_get_member (object, template[i]))
00315         {
00316             if (ninputs && ninputs <= i)
00317             {
00318                 experiment_error (experiment, gettext ("bad templates number"));
00319                 goto exit_on_error;
00320             }
00321             name = json_object_get_string_member (object, template[i]);
00322             #if DEBUG_EXPERIMENT
00323             fprintf (stderr,
00324             "experiment_open_json: experiment=%s template%u=%s\n",
00325             experiment->nexperiments, name, template[i]);
00326             #endif
00327             experiment->template[i] = g_strdup (name);
00328             ++experiment->ninputs;
00329         }
00330         else if (ninputs && ninputs > i)
00331         {
00332             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00333             experiment_error (experiment, buffer);
00334             goto exit_on_error;
00335         }
00336         else
00337             break;
00338     }
00339     #if DEBUG_EXPERIMENT
00340     fprintf (stderr, "experiment_open_json: end\n");
00341     #endif
00342     return 1;

```

```

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

```

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, cJSON *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 void experiment_error ([Experiment](#) * *experiment*, char * *message*)

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

Parameters

<i>experiment</i>	Experiment struct.
<i>message</i>	Error message.

Definition at line 121 of file [experiment.c](#).

```

00122 {
00123     char buffer[64];
00124     if (!experiment->name)
00125         snprintf (buffer, 64, "%s: %s", gettext ("Experiment"), message);
00126     else
00127         snprintf (buffer, 64, "%s %s: %s", gettext ("Experiment"), experiment->name,
00128                 message);
00129     error_message = g_strdup (buffer);
00130 }
```

5.5.2.2 void experiment_free ([Experiment](#) * *experiment*, unsigned int *type*)

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

Parameters

<i>experiment</i>	Experiment struct.
<i>type</i>	Type of input file.

Definition at line 88 of file [experiment.c](#).

```

00089 {
00090     unsigned int i;
00091     #if DEBUG_EXPERIMENT
00092     fprintf (stderr, "experiment_free: start\n");
00093     #endif
00094     if (type == INPUT_TYPE_XML)
00095     {
00096         for (i = 0; i < experiment->ninputs; ++i)
00097             xmlFree (experiment->template[i]);
00098         xmlFree (experiment->name);
00099     }
00100     else
00101     {
00102         for (i = 0; i < experiment->ninputs; ++i)
00103             g_free (experiment->template[i]);
00104         g_free (experiment->name);
00105     }
00106     experiment->ninputs = 0;
00107     #if DEBUG_EXPERIMENT
00108     fprintf (stderr, "experiment_free: end\n");
00109     #endif
00110 }
```

5.5.2.3 void experiment_new ([Experiment](#) * *experiment*)

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

Parameters

<i>experiment</i>	Experiment struct.
-------------------	------------------------------------

Definition at line 64 of file [experiment.c](#).

```

00065 {
00066     unsigned int i;
00067     #if DEBUG_EXPERIMENT
00068         fprintf (stderr, "experiment_new: start\n");
00069     #endif
00070     experiment->name = NULL;
00071     experiment->ninputs = 0;
00072     for (i = 0; i < MAX_NINPUTS; ++i)
00073         experiment->template[i] = NULL;
00074     #if DEBUG_EXPERIMENT
00075         fprintf (stderr, "input_new: end\n");
00076     #endif
00077 }

```

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	JSON node.
<i>ninputs</i>	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 252 of file [experiment.c](#).

```

00254 {
00255     char buffer[64];
00256     JsonObject *object;
00257     const char *name;
00258     int error_code;
00259     unsigned int i;
00260
00261     #if DEBUG_EXPERIMENT
00262         fprintf (stderr, "experiment_open_json: start\n");
00263     #endif
00264
00265     // Resetting experiment data
00266     experiment_new (experiment);
00267
00268     // Getting JSON object
00269     object = json_node_get_object (node);
00270
00271     // Reading the experimental data
00272     name = json_object_get_string_member (object, LABEL_NAME);
00273     if (!name)
00274     {
00275         experiment_error (experiment, gettext ("no data file name"));
00276         goto exit_on_error;
00277     }
00278     experiment->name = g_strdup (name);
00279     #if DEBUG_EXPERIMENT
00280         fprintf (stderr, "experiment_open_json: name=%s\n", experiment->name);
00281     #endif
00282     experiment->weight
00283     = json_object_get_float_with_default (object,
00284     LABEL_WEIGHT, 1.,
00285     &error_code);
00286     if (error_code)
00287     {
00288         experiment_error (experiment, gettext ("bad weight"));
00289         goto exit_on_error;
00290     }
00291 }

```

```

00289     }
00290 #if DEBUG_EXPERIMENT
00291     fprintf (stderr, "experiment_open_json: weight=%lg\n", experiment->weight);
00292 #endif
00293     name = json_object_get_string_member (object, template[0]);
00294     if (name)
00295     {
00296 #if DEBUG_EXPERIMENT
00297         fprintf (stderr, "experiment_open_json: experiment=%s template1=%s\n",
00298             name, template[0]);
00299 #endif
00300         ++experiment->ninputs;
00301     }
00302     else
00303     {
00304         experiment_error (experiment, gettext ("no template"));
00305         goto exit_on_error;
00306     }
00307     experiment->template[0] = g_strdup (name);
00308     for (i = 1; i < MAX_NINPUTS; ++i)
00309     {
00310 #if DEBUG_EXPERIMENT
00311         fprintf (stderr, "experiment_open_json: template%u\n", i + 1);
00312 #endif
00313         if (json_object_get_member (object, template[i]))
00314         {
00315             if (ninputs && ninputs <= i)
00316             {
00317                 experiment_error (experiment, gettext ("bad templates number"));
00318                 goto exit_on_error;
00319             }
00320             name = json_object_get_string_member (object, template[i]);
00321 #if DEBUG_EXPERIMENT
00322             fprintf (stderr,
00323                 "experiment_open_json: experiment=%s template%u=%s\n",
00324                 experiment->nexperiments, name, template[i]);
00325 #endif
00326             experiment->template[i] = g_strdup (name);
00327             ++experiment->ninputs;
00328         }
00329         else if (ninputs && ninputs > i)
00330         {
00331             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00332             experiment_error (experiment, buffer);
00333             goto exit_on_error;
00334         }
00335         else
00336             break;
00337     }
00338 #if DEBUG_EXPERIMENT
00339     fprintf (stderr, "experiment_open_json: end\n");
00340 #endif
00341     return 1;
00342 }
00343 exit_on_error:
00344     experiment_free (experiment, INPUT_TYPE_JSON);
00345 #if DEBUG_EXPERIMENT
00346     fprintf (stderr, "experiment_open_json: end\n");
00347 #endif
00348     return 0;
00349 }
00350 }

```

Here is the call graph for this function:

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

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

Parameters

<i>experiment</i>	Experiment struct.
<i>node</i>	XML node.
<i>ninputs</i>	Number of the simulator input files.

Returns

1 on success, 0 on error.

Definition at line 145 of file [experiment.c](#).

```

00147 {
00148     char buffer[64];
00149     int error_code;
00150     unsigned int i;
00151
00152     #if DEBUG_EXPERIMENT
00153         fprintf (stderr, "experiment_open_xml: start\n");
00154     #endif
00155
00156     // Resetting experiment data
00157     experiment_new (experiment);
00158
00159     // Reading the experimental data
00160     experiment->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00161     if (!experiment->name)
00162     {
00163         experiment_error (experiment, gettext ("no data file name"));
00164         goto exit_on_error;
00165     }
00166     #if DEBUG_EXPERIMENT
00167         fprintf (stderr, "experiment_open_xml: name=%s\n", experiment->name);
00168     #endif
00169     experiment->weight
00170     = xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_WEIGHT, 1.,
00171                                     &error_code);
00172     if (error_code)
00173     {
00174         experiment_error (experiment, gettext ("bad weight"));
00175         goto exit_on_error;
00176     }
00177     #if DEBUG_EXPERIMENT
00178         fprintf (stderr, "experiment_open_xml: weight=%lg\n", experiment->weight);
00179     #endif
00180     experiment->template[0]
00181     = (char *) xmlGetProp (node, (const xmlChar *) template[0]);
00182     if (experiment->template[0])
00183     {
00184         #if DEBUG_EXPERIMENT
00185             fprintf (stderr, "experiment_open_xml: experiment=%s template1=%s\n",
00186                     experiment->name, template[0]);
00187         #endif
00188         ++experiment->ninputs;
00189     }
00190     else
00191     {
00192         experiment_error (experiment, gettext ("no template"));
00193         goto exit_on_error;
00194     }
00195     for (i = 1; i < MAX_NINPUTS; ++i)
00196     {
00197         #if DEBUG_EXPERIMENT
00198             fprintf (stderr, "experiment_open_xml: template%u\n", i + 1);
00199         #endif
00200         if (xmlHasProp (node, (const xmlChar *) template[i]))
00201         {
00202             if (ninputs && ninputs <= i)
00203             {
00204                 experiment_error (experiment, gettext ("bad templates number"));
00205                 goto exit_on_error;
00206             }
00207             experiment->template[i]
00208             = (char *) xmlGetProp (node, (const xmlChar *) template[i]);
00209             #if DEBUG_EXPERIMENT
00210                 fprintf (stderr, "experiment_open_xml: experiment=%s template%u=%s\n",
00211                         experiment->name, experiment->nexperiments,
00212                         experiment->template[i]);
00213             #endif
00214             ++experiment->ninputs;
00215         }
00216         else if (ninputs && ninputs > i)
00217         {
00218             snprintf (buffer, 64, "%s%u", gettext ("no template"), i + 1);
00219             experiment_error (experiment, buffer);
00220             goto exit_on_error;
00221         }
00222         else
00223             break;

```

```

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

```

Here is the call graph for this function:

5.6 experiment.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef EXPERIMENT__H
00033 #define EXPERIMENT__H 1
00034
00035 typedef struct
00036 {
00037     char *name;
00038     char *template[MAX_NINPUTS];
00039     double weight;
00040     unsigned int ninputs;
00041 } Experiment;
00042
00043 extern const char *template[MAX_NINPUTS];
00044
00045 // Public functions
00046 void experiment_new (Experiment * experiment);
00047 void experiment_free (Experiment * experiment, unsigned int type);
00048 void experiment_error (Experiment * experiment, char *message);
00049 int experiment_open_xml (Experiment * experiment, xmlNode * node,
00050                         unsigned int ninputs);
00051 int experiment_open_json (Experiment * experiment, JsonNode * node,
00052                          unsigned int ninputs);
00053
00054 #endif

```

5.7 input.c File Reference

Source file to define the input functions.

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

Include dependency graph for input.c:

Macros

- `#define _GNU_SOURCE`
- `#define DEBUG_INPUT 0`
Macro to debug input functions.

Functions

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

Variables

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

5.7.1 Detailed Description

Source file to define the input functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.7.2 Function Documentation

5.7.2.1 void input_error (char * *message*)

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

Parameters

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

Definition at line [124](#) of file [input.c](#).

```
00125 {  
00126     char buffer[64];  
00127     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);  
00128     error\_message = g_strdup (buffer);  
00129 }
```

5.7.2.2 int input_open (char * *filename*)

Function to open the input file.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line [947](#) of file [input.c](#).

```
00948 {
```

```

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

```

Here is the call graph for this function:

5.7.2.3 int input_open_json (JsonParser * parser)

Function to open the input file in JSON format.

Parameters

<i>parser</i>	JsonParser struct.
---------------	--------------------

Returns

1_on_success, 0_on_error.

Definition at line 557 of file [input.c](#).

```

00558 {
00559     JsonNode *node, *child;

```

```

00560     JsonObject *object;
00561     JsonArray *array;
00562     const char *buffer;
00563     int error_code;
00564     unsigned int i, n;
00565
00566     #if DEBUG_INPUT
00567     fprintf (stderr, "input_open_json: start\n");
00568     #endif
00569
00570     // Resetting input data
00571     input->type = INPUT_TYPE_JSON;
00572
00573     // Getting the root node
00574     #if DEBUG_INPUT
00575     fprintf (stderr, "input_open_json: getting the root node\n");
00576     #endif
00577     node = json_parser_get_root (parser);
00578     object = json_node_get_object (node);
00579
00580     // Getting result and variables file names
00581     if (!input->result)
00582     {
00583         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00584         if (!buffer)
00585             buffer = result_name;
00586         input->result = g_strdup (buffer);
00587     }
00588     else
00589         input->result = g_strdup (result_name);
00590     if (!input->variables)
00591     {
00592         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00593         if (!buffer)
00594             buffer = variables_name;
00595         input->variables = g_strdup (buffer);
00596     }
00597     else
00598         input->variables = g_strdup (variables_name);
00599
00600     // Opening simulator program name
00601     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00602     if (!buffer)
00603     {
00604         input_error (gettext ("Bad simulator program"));
00605         goto exit_on_error;
00606     }
00607     input->simulator = g_strdup (buffer);
00608
00609     // Opening evaluator program name
00610     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00611     if (buffer)
00612         input->evaluator = g_strdup (buffer);
00613
00614     // Obtaining pseudo-random numbers generator seed
00615     input->seed
00616     = json_object_get_uint_with_default (object,
00617     LABEL_SEED,
00618     DEFAULT_RANDOM_SEED, &error_code);
00619     if (error_code)
00620     {
00621         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00622         goto exit_on_error;
00623     }
00624
00625     // Opening algorithm
00626     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00627     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00628     {
00629         input->algorithm = ALGORITHM_MONTE_CARLO;
00630
00631         // Obtaining simulations number
00632         input->nsimulations
00633         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00634     );
00635     if (error_code)
00636     {
00637         input_error (gettext ("Bad simulations number"));
00638         goto exit_on_error;
00639     }
00640     else if (!strcmp (buffer, LABEL_SWEEP))
00641         input->algorithm = ALGORITHM_SWEEP;
00642     else if (!strcmp (buffer, LABEL_GENETIC))
00643     {
00644         input->algorithm = ALGORITHM_GENETIC;

```

```

00645     // Obtaining population
00646     if (json_object_get_member (object, LABEL_NPOPULATION))
00647     {
00648         input->nsimulations
00649         = json_object_get_uint (object,
00650 LABEL_NPOPULATION, &error_code);
00651         if (error_code || input->nsimulations < 3)
00652         {
00653             input_error (gettext ("Invalid population number"));
00654             goto exit_on_error;
00655         }
00656     else
00657     {
00658         input_error (gettext ("No population number"));
00659         goto exit_on_error;
00660     }
00661
00662     // Obtaining generations
00663     if (json_object_get_member (object, LABEL_NGENERATIONS))
00664     {
00665         input->niterations
00666         = json_object_get_uint (object,
00667 LABEL_NGENERATIONS, &error_code);
00668         if (error_code || !input->niterations)
00669         {
00670             input_error (gettext ("Invalid generations number"));
00671             goto exit_on_error;
00672         }
00673     else
00674     {
00675         input_error (gettext ("No generations number"));
00676         goto exit_on_error;
00677     }
00678
00679     // Obtaining mutation probability
00680     if (json_object_get_member (object, LABEL_MUTATION))
00681     {
00682         input->mutation_ratio
00683         = json_object_get_float (object, LABEL_MUTATION, &error_code
00684 );
00685         if (error_code || input->mutation_ratio < 0.
00686             || input->mutation_ratio >= 1.)
00687         {
00688             input_error (gettext ("Invalid mutation probability"));
00689             goto exit_on_error;
00690         }
00691     else
00692     {
00693         input_error (gettext ("No mutation probability"));
00694         goto exit_on_error;
00695     }
00696
00697     // Obtaining reproduction probability
00698     if (json_object_get_member (object, LABEL_REPRODUCTION))
00699     {
00700         input->reproduction_ratio
00701         = json_object_get_float (object,
00702 LABEL_REPRODUCTION, &error_code);
00703         if (error_code || input->reproduction_ratio < 0.
00704             || input->reproduction_ratio >= 1.0)
00705         {
00706             input_error (gettext ("Invalid reproduction probability"));
00707             goto exit_on_error;
00708         }
00709     else
00710     {
00711         input_error (gettext ("No reproduction probability"));
00712         goto exit_on_error;
00713     }
00714
00715     // Obtaining adaptation probability
00716     if (json_object_get_member (object, LABEL_ADAPTATION))
00717     {
00718         input->adaptation_ratio
00719         = json_object_get_float (object,
00720 LABEL_ADAPTATION, &error_code);
00721         if (error_code || input->adaptation_ratio < 0.
00722             || input->adaptation_ratio >= 1.)
00723         {
00724             input_error (gettext ("Invalid adaptation probability"));
00725             goto exit_on_error;
00726         }
00727     }

```

```

00727     else
00728     {
00729         input_error (gettext ("No adaptation probability"));
00730         goto exit_on_error;
00731     }
00732
00733     // Checking survivals
00734     i = input->mutation_ratio * input->nsimulations;
00735     i += input->reproduction_ratio * input->
nsimulations;
00736     i += input->adaptation_ratio * input->
nsimulations;
00737     if (i > input->nsimulations - 2)
00738     {
00739         input_error
00740         (gettext
00741          ("No enough survival entities to reproduce the population"));
00742         goto exit_on_error;
00743     }
00744
00745     else
00746     {
00747         input_error (gettext ("Unknown algorithm"));
00748         goto exit_on_error;
00749     }
00750
00751     if (input->algorithm == ALGORITHM_MONTE_CARLO
00752         || input->algorithm == ALGORITHM_SWEEP)
00753     {
00754
00755         // Obtaining iterations number
00756         input->niterations
00757         = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00758         if (error_code == 1)
00759             input->niterations = 1;
00760         else if (error_code)
00761         {
00762             input_error (gettext ("Bad iterations number"));
00763             goto exit_on_error;
00764         }
00765
00766         // Obtaining best number
00767         input->nbest
00768         = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
00769                                             &error_code);
00770         if (error_code || !input->nbest)
00771         {
00772             input_error (gettext ("Invalid best number"));
00773             goto exit_on_error;
00774         }
00775
00776         // Obtaining tolerance
00777         input->tolerance
00778         = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
00779                                             &error_code);
00780         if (error_code || input->tolerance < 0.)
00781         {
00782             input_error (gettext ("Invalid tolerance"));
00783             goto exit_on_error;
00784         }
00785
00786         // Getting direction search method parameters
00787         if (json_object_get_member (object, LABEL_NSTEPS))
00788         {
00789             input->nsteps
00790             = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00791             if (error_code || !input->nsteps)
00792             {
00793                 input_error (gettext ("Invalid steps number"));
00794                 goto exit_on_error;
00795             }
00796             buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00797             if (!strcmp (buffer, LABEL_COORDINATES))
00798                 input->direction = DIRECTION_METHOD_COORDINATES;
00799             else if (!strcmp (buffer, LABEL_RANDOM))
00800             {
00801                 input->direction = DIRECTION_METHOD_RANDOM;
00802                 input->nestimates
00803                 = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00804                 if (error_code || !input->nestimates)
00805                 {
00806                     input_error (gettext ("Invalid estimates number"));
00807                     goto exit_on_error;

```

```

00808         }
00809     }
00810     else
00811     {
00812         input_error
00813         (gettext ("Unknown method to estimate the direction search"));
00814         goto exit_on_error;
00815     }
00816     input->relaxation
00817     = json_object_get_float_with_default (object,
00818     LABEL_RELAXATION,
00819     DEFAULT_RELAXATION,
00820     &error_code);
00821     if (error_code || input->relaxation < 0. || input->
00822     relaxation > 2.)
00823     {
00824         input_error (gettext ("Invalid relaxation parameter"));
00825         goto exit_on_error;
00826     }
00827     else
00828     {
00829         input->nsteps = 0;
00830         // Obtaining the threshold
00831         input->threshold
00832         = json_object_get_float_with_default (object,
00833     LABEL_THRESHOLD, 0.,
00834     &error_code);
00835     if (error_code)
00836     {
00837         input_error (gettext ("Invalid threshold"));
00838         goto exit_on_error;
00839     }
00840     // Reading the experimental data
00841     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00842     n = json_array_get_length (array);
00843     input->experiment = (Experiment *) g_malloc (n * sizeof (
00844     Experiment));
00845     for (i = 0; i < n; ++i)
00846     {
00847         #if DEBUG_INPUT
00848         fprintf (stderr, "input_open_json: nexperiments=%u\n",
00849         input->nexperiments);
00850         #endif
00851         child = json_array_get_element (array, i);
00852         if (!input->nexperiments)
00853         {
00854             if (!experiment_open_json (input->experiment, child, 0))
00855             {
00856                 goto exit_on_error;
00857             }
00858             else
00859             {
00860                 if (!experiment_open_json (input->experiment +
00861         input->nexperiments,
00862         child, input->experiment->
00863         ninputs))
00864             {
00865                 goto exit_on_error;
00866             }
00867             ++input->nexperiments;
00868             #if DEBUG_INPUT
00869             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00870             input->nexperiments);
00871             #endif
00872             if (!input->nexperiments)
00873             {
00874                 input_error (gettext ("No optimization experiments"));
00875                 goto exit_on_error;
00876             }
00877             // Reading the variables data
00878             array = json_object_get_array_member (object, LABEL_VARIABLES);
00879             n = json_array_get_length (array);
00880             input->variable = (Variable *) g_malloc (n * sizeof (
00881             Variable));
00882             for (i = 0; i < n; ++i)
00883             {
00884                 #if DEBUG_INPUT
00885                 fprintf (stderr, "input_open_json: nvariables=%u\n", input->
00886         nvariables);
00887                 #endif
00888                 child = json_array_get_element (array, i);
00889                 if (!variable_open_json (input->variable +
00890         input->nvariables, child,
00891         input->algorithm, input->
00892         nsteps))

```

```

00885         goto exit_on_error;
00886     ++input->nvariables;
00887 }
00888 if (!input->nvariables)
00889 {
00890     input_error (gettext ("No optimization variables"));
00891     goto exit_on_error;
00892 }
00893
00894 // Obtaining the error norm
00895 if (json_object_get_member (object, LABEL_NORM))
00896 {
00897     buffer = json_object_get_string_member (object, LABEL_NORM);
00898     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00899         input->norm = ERROR_NORM_EUCLIDIAN;
00900     else if (!strcmp (buffer, LABEL_MAXIMUM))
00901         input->norm = ERROR_NORM_MAXIMUM;
00902     else if (!strcmp (buffer, LABEL_P))
00903     {
00904         input->norm = ERROR_NORM_P;
00905         input->p = json_object_get_float (object,
00906 LABEL_P, &error_code);
00907         if (!error_code)
00908         {
00909             input_error (gettext ("Bad P parameter"));
00910             goto exit_on_error;
00911         }
00912     }
00913     else if (!strcmp (buffer, LABEL_TAXICAB))
00914         input->norm = ERROR_NORM_TAXICAB;
00915     else
00916     {
00917         input_error (gettext ("Unknown error norm"));
00918         goto exit_on_error;
00919     }
00920 }
00921 else
00922     input->norm = ERROR_NORM_EUCLIDIAN;
00923
00924 // Closing the JSON document
00925 g_object_unref (parser);
00926
00927 #if DEBUG_INPUT
00928 fprintf (stderr, "input_open_json: end\n");
00929 #endif
00930 return 1;
00931
00932 exit_on_error:
00933 g_object_unref (parser);
00934 #if DEBUG_INPUT
00935 fprintf (stderr, "input_open_json: end\n");
00936 #endif
00937 return 0;
00938 }

```

Here is the call graph for this function:

5.7.2.4 int input_open_xml (xmlDoc * doc)

Function to open the input file in XML format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 139 of file [input.c](#).

```

00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157         fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (gettext ("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173     }
00174     if (!input->variables)
00175     {
00176         input->variables =
00177             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178         if (!input->variables)
00179             input->variables =
00180                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (gettext ("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                DEFAULT_RANDOM_SEED, &error_code);
00199     if (error_code)
00200     {
00201         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213         = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                                &error_code);
00214         if (error_code)
00215         {
00216             input_error (gettext ("Bad simulations number"));
00217             goto exit_on_error;
00218         }
00219     }
00220     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00221     {
00222         input->algorithm = ALGORITHM_SWEEP;
00223     }

```



```

00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226         input->algorithm = ALGORITHM_GENETIC;
00227
00228         // Obtaining population
00229         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231             input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NPOPULATION,
00233                                 &error_code);
00234             if (error_code || input->nsimulations < 3)
00235             {
00236                 input_error (gettext ("Invalid population number"));
00237                 goto exit_on_error;
00238             }
00239         }
00240         else
00241         {
00242             input_error (gettext ("No population number"));
00243             goto exit_on_error;
00244         }
00245
00246         // Obtaining generations
00247         if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248         {
00249             input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NGENERATIONS,
00251                                 &error_code);
00252             if (error_code || !input->niterations)
00253             {
00254                 input_error (gettext ("Invalid generations number"));
00255                 goto exit_on_error;
00256             }
00257         }
00258         else
00259         {
00260             input_error (gettext ("No generations number"));
00261             goto exit_on_error;
00262         }
00263
00264         // Obtaining mutation probability
00265         if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266         {
00267             input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *)
LABEL_MUTATION,
00269                                 &error_code);
00270             if (error_code || input->mutation_ratio < 0.
|| input->mutation_ratio >= 1.)
00271             {
00272                 input_error (gettext ("Invalid mutation probability"));
00273                 goto exit_on_error;
00274             }
00275         }
00276         else
00277         {
00278             input_error (gettext ("No mutation probability"));
00279             goto exit_on_error;
00280         }
00281
00282         // Obtaining reproduction probability
00283         if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00284         {
00285             input->reproduction_ratio
00286             = xml_node_get_float (node, (const xmlChar *)
LABEL_REPRODUCTION,
00287                                 &error_code);
00288             if (error_code || input->reproduction_ratio < 0.
|| input->reproduction_ratio >= 1.0)
00289             {
00290                 input_error (gettext ("Invalid reproduction probability"));
00291                 goto exit_on_error;
00292             }
00293         }
00294         else
00295         {
00296             input_error (gettext ("No reproduction probability"));
00297             goto exit_on_error;
00298         }
00299
00300         // Obtaining adaptation probability
00301         if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00302         {
00303             input->adaptation_ratio
00304             = xml_node_get_float (node, (const xmlChar *)

```

```

    LABEL_ADAPTATION,
00307                                     &error_code);
00308         if (error_code || input->adaptation_ratio < 0.
00309             || input->adaptation_ratio >= 1.)
00310         {
00311             input_error (gettext ("Invalid adaptation probability"));
00312             goto exit_on_error;
00313         }
00314     }
00315     else
00316     {
00317         input_error (gettext ("No adaptation probability"));
00318         goto exit_on_error;
00319     }
00320
00321     // Checking survivals
00322     i = input->mutation_ratio * input->nsimulations;
00323     i += input->reproduction_ratio * input->
00324     nsimulations;
00325     if (i > input->nsimulations - 2)
00326     {
00327         input_error
00328         (gettext
00329          ("No enough survival entities to reproduce the population"));
00330         goto exit_on_error;
00331     }
00332 }
00333 else
00334 {
00335     input_error (gettext ("Unknown algorithm"));
00336     goto exit_on_error;
00337 }
00338 xmlFree (buffer);
00339 buffer = NULL;
00340
00341 if (input->algorithm == ALGORITHM_MONTE_CARLO
00342     || input->algorithm == ALGORITHM_SWEEP)
00343 {
00344     // Obtaining iterations number
00345     input->niterations
00346     = xml_node_get_uint (node, (const xmlChar *)
00347     LABEL_NITERATIONS,
00348                         &error_code);
00349     if (error_code == 1)
00350         input->niterations = 1;
00351     else if (error_code)
00352     {
00353         input_error (gettext ("Bad iterations number"));
00354         goto exit_on_error;
00355     }
00356
00357     // Obtaining best number
00358     input->nbest
00359     = xml_node_get_uint_with_default (node, (const xmlChar *)
00360     LABEL_NBEST,
00361                                     1, &error_code);
00362     if (error_code || !input->nbest)
00363     {
00364         input_error (gettext ("Invalid best number"));
00365         goto exit_on_error;
00366     }
00367
00368     // Obtaining tolerance
00369     input->tolerance
00370     = xml_node_get_float_with_default (node,
00371                                     (const xmlChar *) LABEL_TOLERANCE,
00372                                     0., &error_code);
00373     if (error_code || input->tolerance < 0.)
00374     {
00375         input_error (gettext ("Invalid tolerance"));
00376         goto exit_on_error;
00377     }
00378
00379     // Getting direction search method parameters
00380     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00381     {
00382         input->nsteps =
00383         xml_node_get_uint (node, (const xmlChar *)
00384         LABEL_NSTEPS,
00385                         &error_code);
00386         if (error_code || !input->nsteps)
00387         {
00388             input_error (gettext ("Invalid steps number"));
00389             goto exit_on_error;
00390         }
00391     }

```

```

00388     }
00389     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00390     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00391         input->direction = DIRECTION_METHOD_COORDINATES;
00392     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00393     {
00394         input->direction = DIRECTION_METHOD_RANDOM;
00395         input->nestimates
00396             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00397                                 &error_code);
00398         if (error_code || !input->nestimates)
00399         {
00400             input_error (gettext ("Invalid estimates number"));
00401             goto exit_on_error;
00402         }
00403     }
00404     else
00405     {
00406         input_error
00407             (gettext ("Unknown method to estimate the direction search"));
00408         goto exit_on_error;
00409     }
00410     xmlFree (buffer);
00411     buffer = NULL;
00412     input->relaxation
00413         = xml_node_get_float_with_default (node,
00414                                           (const xmlChar *)
LABEL_RELAXATION,
00415                                           DEFAULT_RELAXATION, &error_code);
00416     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00417     {
00418         input_error (gettext ("Invalid relaxation parameter"));
00419         goto exit_on_error;
00420     }
00421     }
00422     }
00423     else
00424         input->nsteps = 0;
00425     }
00426     // Obtaining the threshold
00427     input->threshold =
00428         xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00429                                         0., &error_code);
00430     if (error_code)
00431     {
00432         input_error (gettext ("Invalid threshold"));
00433         goto exit_on_error;
00434     }
00435     // Reading the experimental data
00436     for (child = node->children; child; child = child->next)
00437     {
00438         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00439             break;
00440         #if DEBUG_INPUT
00441         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00442                 input->nexperiments);
00443         #endif
00444         input->experiment = (Experiment *)
00445             g_realloc (input->experiment,
00446                       (1 + input->nexperiments) * sizeof (
Experiment));
00447         if (!input->nexperiments)
00448         {
00449             if (!experiment_open_xml (input->experiment, child, 0))
00450                 goto exit_on_error;
00451         }
00452         else
00453         {
00454             if (!experiment_open_xml (input->experiment +
input->nexperiments,
00455                                     child, input->experiment->
ninputs))
00456                 goto exit_on_error;
00457         }
00458         ++input->nexperiments;
00459         #if DEBUG_INPUT
00460         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00461                 input->nexperiments);
00462         #endif
00463     }
00464     if (!input->nexperiments)
00465     {
00466         input_error (gettext ("No optimization experiments"));
00467         goto exit_on_error;
00468     }

```

```

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

```

Here is the call graph for this function:

5.8 input.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <string.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <glib/gstdio.h>
00040 #include <json-glib/json-glib.h>
00041 #include "utils.h"
00042 #include "experiment.h"
00043 #include "variable.h"
00044 #include "input.h"
00045
00046 #define DEBUG_INPUT 0
00047
00048 Input input[1];
00049
00050 const char *result_name = "result";
00051 const char *variables_name = "variables";
00052
00053 void
00054 input_new ()
00055 {
00056     #if DEBUG_INPUT
00057         fprintf (stderr, "input_new: start\n");
00058     #endif
00059     input->nvariables = input->nexperiments = input->nsteps = 0;
00060     input->simulator = input->evaluator = input->directory = input->
00061         name = NULL;
00062     input->experiment = NULL;
00063     input->variable = NULL;
00064     #if DEBUG_INPUT
00065         fprintf (stderr, "input_new: end\n");
00066     #endif
00067 }
00068
00069 void
00070 input_free ()
00071 {
00072     unsigned int i;
00073     #if DEBUG_INPUT
00074         fprintf (stderr, "input_free: start\n");
00075     #endif
00076     g_free (input->name);
00077     g_free (input->directory);
00078     for (i = 0; i < input->nexperiments; ++i)
00079         experiment_free (input->experiment + i, input->type);
00080     for (i = 0; i < input->nvariables; ++i)
00081         variable_free (input->variable + i, input->type);
00082     g_free (input->experiment);
00083     g_free (input->variable);
00084     if (input->type == INPUT_TYPE_XML)

```

```

00098     {
00099         xmlFree (input->evaluator);
00100         xmlFree (input->simulator);
00101         xmlFree (input->result);
00102         xmlFree (input->variables);
00103     }
00104     else
00105     {
00106         g_free (input->evaluator);
00107         g_free (input->simulator);
00108         g_free (input->result);
00109         g_free (input->variables);
00110     }
00111     input->nexperiments = input->nvariables = input->nsteps = 0;
00112     #if DEBUG_INPUT
00113     fprintf (stderr, "input_free: end\n");
00114     #endif
00115 }
00116
00123 void
00124 input_error (char *message)
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
00130
00138 int
00139 input_open_xml (xmlDoc * doc)
00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148     fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157     fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (gettext ("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *) result_name);
00173     }
00174     if (!input->variables)
00175     {
00176         input->variables =
00177             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178         if (!input->variables)
00179             input->variables =
00180                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (gettext ("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed

```

```

00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
00199     LABEL_SEED,
00200     DEFAULT_RANDOM_SEED, &error_code);
00201     if (error_code)
00202     {
00203         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00204         goto exit_on_error;
00205     }
00206     // Opening algorithm
00207     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00208     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00209     {
00210         input->algorithm = ALGORITHM_MONTE_CARLO;
00211     }
00212     // Obtaining simulations number
00213     input->nsimulations
00214     = xml_node_get_int (node, (const xmlChar *)
00215     LABEL_NSIMULATIONS,
00216     &error_code);
00217     if (error_code)
00218     {
00219         input_error (gettext ("Bad simulations number"));
00220         goto exit_on_error;
00221     }
00222     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00223         input->algorithm = ALGORITHM_SWEEP;
00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226         input->algorithm = ALGORITHM_GENETIC;
00227     }
00228     // Obtaining population
00229     if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230     {
00231         input->nsimulations
00232         = xml_node_get_uint (node, (const xmlChar *)
00233         LABEL_NPOPULATION,
00234         &error_code);
00235         if (error_code || input->nsimulations < 3)
00236         {
00237             input_error (gettext ("Invalid population number"));
00238             goto exit_on_error;
00239         }
00240     }
00241     else
00242     {
00243         input_error (gettext ("No population number"));
00244         goto exit_on_error;
00245     }
00246     // Obtaining generations
00247     if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248     {
00249         input->niterations
00250         = xml_node_get_uint (node, (const xmlChar *)
00251         LABEL_NGENERATIONS,
00252         &error_code);
00253         if (error_code || !input->niterations)
00254         {
00255             input_error (gettext ("Invalid generations number"));
00256             goto exit_on_error;
00257         }
00258     }
00259     else
00260     {
00261         input_error (gettext ("No generations number"));
00262         goto exit_on_error;
00263     }
00264     // Obtaining mutation probability
00265     if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266     {
00267         input->mutation_ratio
00268         = xml_node_get_float (node, (const xmlChar *)
00269         LABEL_MUTATION,
00270         &error_code);
00271         if (error_code || input->mutation_ratio < 0.
00272             || input->mutation_ratio >= 1.)
00273         {
00274             input_error (gettext ("Invalid mutation probability"));
00275             goto exit_on_error;
00276         }
00277     }
00278     else
00279     {
00280         input_error (gettext ("No mutation probability"));

```

```

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 *)
00288 LABEL_REPRODUCTION,
00289                             &error_code);
00289         if (error_code || input->reproduction_ratio < 0.
00290             || input->reproduction_ratio >= 1.0)
00291         {
00292             input_error (gettext ("Invalid reproduction probability"));
00293             goto exit_on_error;
00294         }
00295     }
00296     else
00297     {
00298         input_error (gettext ("No reproduction probability"));
00299         goto exit_on_error;
00300     }
00301
00302     // Obtaining adaptation probability
00303     if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00304     {
00305         input->adaptation_ratio
00306         = xml_node_get_float (node, (const xmlChar *)
00307 LABEL_ADAPTATION,
00308                             &error_code);
00308         if (error_code || input->adaptation_ratio < 0.
00309             || input->adaptation_ratio >= 1.)
00310         {
00311             input_error (gettext ("Invalid adaptation probability"));
00312             goto exit_on_error;
00313         }
00314     }
00315     else
00316     {
00317         input_error (gettext ("No adaptation probability"));
00318         goto exit_on_error;
00319     }
00320
00321     // Checking survivals
00322     i = input->mutation_ratio * input->nsimulations;
00323     i += input->reproduction_ratio * input->nsimulations;
00324     i += input->adaptation_ratio * input->nsimulations;
00325     if (i > input->nsimulations - 2)
00326     {
00327         input_error
00328         (gettext
00329          ("No enough survival entities to reproduce the population"));
00330         goto exit_on_error;
00331     }
00332 }
00333 else
00334 {
00335     input_error (gettext ("Unknown algorithm"));
00336     goto exit_on_error;
00337 }
00338 xmlFree (buffer);
00339 buffer = NULL;
00340
00341 if (input->algorithm == ALGORITHM_MONTE_CARLO
00342     || input->algorithm == ALGORITHM_SWEEP)
00343 {
00344
00345     // Obtaining iterations number
00346     input->niterations
00347     = xml_node_get_uint (node, (const xmlChar *)
00348 LABEL_NITERATIONS,
00349                         &error_code);
00349     if (error_code == 1)
00350         input->niterations = 1;
00351     else if (error_code)
00352     {
00353         input_error (gettext ("Bad iterations number"));
00354         goto exit_on_error;
00355     }
00356
00357     // Obtaining best number
00358     input->nbest
00359     = xml_node_get_uint_with_default (node, (const xmlChar *)
00360 LABEL_NBEST,
00361                                     1, &error_code);
00361     if (error_code || !input->nbest)
00362     {

```



```

00363         input_error (gettext ("Invalid best number"));
00364         goto exit_on_error;
00365     }
00366
00367     // Obtaining tolerance
00368     input->tolerance
00369     = xml_node_get_float_with_default (node,
00370         (const xmlChar *) LABEL_TOLERANCE,
00371         0., &error_code);
00372     if (error_code || input->tolerance < 0.)
00373     {
00374         input_error (gettext ("Invalid tolerance"));
00375         goto exit_on_error;
00376     }
00377
00378     // Getting direction search method parameters
00379     if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00380     {
00381         input->nsteps =
00382             xml_node_get_uint (node, (const xmlChar *)
00383                 LABEL_NSTEPS,
00384                 &error_code);
00385         if (error_code || !input->nsteps)
00386         {
00387             input_error (gettext ("Invalid steps number"));
00388             goto exit_on_error;
00389         }
00390         buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00391         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00392             input->direction = DIRECTION_METHOD_COORDINATES;
00393         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00394         {
00395             input->direction = DIRECTION_METHOD_RANDOM;
00396             input->nestimates
00397             = xml_node_get_uint (node, (const xmlChar *)
00398                 LABEL_NESTIMATES,
00399                 &error_code);
00400             if (error_code || !input->nestimates)
00401             {
00402                 input_error (gettext ("Invalid estimates number"));
00403                 goto exit_on_error;
00404             }
00405             else
00406             {
00407                 input_error
00408                 (gettext ("Unknown method to estimate the direction search"));
00409                 goto exit_on_error;
00410             }
00411             xmlFree (buffer);
00412             buffer = NULL;
00413             input->relaxation
00414             = xml_node_get_float_with_default (node,
00415                 (const xmlChar *)
00416                 LABEL_RELAXATION,
00417                 DEFAULT_RELAXATION, &error_code);
00418             if (error_code || input->relaxation < 0. || input->
00419                 relaxation > 2.)
00420             {
00421                 input_error (gettext ("Invalid relaxation parameter"));
00422                 goto exit_on_error;
00423             }
00424             else
00425             {
00426                 input->nsteps = 0;
00427             }
00428             // Obtaining the threshold
00429             input->threshold =
00430                 xml_node_get_float_with_default (node, (const xmlChar *)
00431                     LABEL_THRESHOLD,
00432                     0., &error_code);
00433             if (error_code)
00434             {
00435                 input_error (gettext ("Invalid threshold"));
00436                 goto exit_on_error;
00437             }
00438             // Reading the experimental data
00439             for (child = node->children; child; child = child->next)
00440             {
00441                 if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00442                     break;
00443             }
00444             #if DEBUG_INPUT
00445             fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00446                 input->nexperiments);
00447             #endif
00448             input->experiment = (Experiment *)

```

```

00446         g_realloc (input->experiment,
00447                     (1 + input->nexperiments) * sizeof (Experiment));
00448     if (!input->nexperiments)
00449     {
00450         if (!experiment_open_xml (input->experiment, child, 0))
00451             goto exit_on_error;
00452     }
00453     else
00454     {
00455         if (!experiment_open_xml (input->experiment + input->
00456     nexperiments,
00457                                     child, input->experiment->ninputs))
00458             goto exit_on_error;
00459         ++input->nexperiments;
00460 #if DEBUG_INPUT
00461         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00462                 input->nexperiments);
00463 #endif
00464     }
00465     if (!input->nexperiments)
00466     {
00467         input_error (gettext ("No optimization experiments"));
00468         goto exit_on_error;
00469     }
00470     buffer = NULL;
00471
00472     // Reading the variables data
00473     for (; child; child = child->next)
00474     {
00475 #if DEBUG_INPUT
00476         fprintf (stderr, "input_open_xml: nvariables=%u\n", input->nvariables);
00477 #endif
00478         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_VARIABLE))
00479         {
00480             snprintf (buffer2, 64, "%s %u: %s",
00481                     gettext ("Variable"),
00482                     input->nvariables + 1, gettext ("bad XML node"));
00483             input_error (buffer2);
00484             goto exit_on_error;
00485         }
00486         input->variable = (Variable *)
00487             g_realloc (input->variable,
00488                       (1 + input->nvariables) * sizeof (Variable));
00489         if (!variable_open_xml (input->variable + input->
00490     nvariables, child,
00491                                     input->algorithm, input->nsteps))
00492             goto exit_on_error;
00493         ++input->nvariables;
00494     }
00495     if (!input->nvariables)
00496     {
00497         input_error (gettext ("No optimization variables"));
00498         goto exit_on_error;
00499     }
00500     buffer = NULL;
00501
00502     // Obtaining the error norm
00503     if (xmlHasProp (node, (const xmlChar *) LABEL_NORM))
00504     {
00505         buffer = xmlGetProp (node, (const xmlChar *) LABEL_NORM);
00506         if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_EUCLIDIAN))
00507             input->norm = ERROR_NORM_EUCLIDIAN;
00508         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MAXIMUM))
00509             input->norm = ERROR_NORM_MAXIMUM;
00510         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_P))
00511         {
00512             input->norm = ERROR_NORM_P;
00513             input->p
00514             = xml_node_get_float (node, (const xmlChar *)
00515     LABEL_P, &error_code);
00516             if (!error_code)
00517             {
00518                 input_error (gettext ("Bad P parameter"));
00519                 goto exit_on_error;
00520             }
00521         }
00522         else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_TAXICAB))
00523             input->norm = ERROR_NORM_TAXICAB;
00524         else
00525         {
00526             input_error (gettext ("Unknown error norm"));
00527             goto exit_on_error;
00528         }
00529         xmlFree (buffer);
00530     }
00531     else

```

```

00530     input->norm = ERROR_NORM_EUCLIDIAN;
00531
00532     // Closing the XML document
00533     xmlFreeDoc (doc);
00534
00535     #if DEBUG_INPUT
00536     fprintf (stderr, "input_open_xml: end\n");
00537     #endif
00538     return 1;
00539
00540 exit_on_error:
00541     xmlFree (buffer);
00542     xmlFreeDoc (doc);
00543     #if DEBUG_INPUT
00544     fprintf (stderr, "input_open_xml: end\n");
00545     #endif
00546     return 0;
00547 }
00548
00549 int
00550 input_open_json (JsonParser * parser)
00551 {
00552     JsonNode *node, *child;
00553     JsonObject *object;
00554     JsonArray *array;
00555     const char *buffer;
00556     int error_code;
00557     unsigned int i, n;
00558
00559     #if DEBUG_INPUT
00560     fprintf (stderr, "input_open_json: start\n");
00561     #endif
00562
00563     // Resetting input data
00564     input->type = INPUT_TYPE_JSON;
00565
00566     // Getting the root node
00567     #if DEBUG_INPUT
00568     fprintf (stderr, "input_open_json: getting the root node\n");
00569     #endif
00570     node = json_parser_get_root (parser);
00571     object = json_node_get_object (node);
00572
00573     // Getting result and variables file names
00574     if (!input->result)
00575     {
00576         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00577         if (!buffer)
00578             buffer = result_name;
00579         input->result = g_strdup (buffer);
00580     }
00581     else
00582         input->result = g_strdup (result_name);
00583     if (!input->variables)
00584     {
00585         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00586         if (!buffer)
00587             buffer = variables_name;
00588         input->variables = g_strdup (buffer);
00589     }
00590     else
00591         input->variables = g_strdup (variables_name);
00592
00593     // Opening simulator program name
00594     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00595     if (!buffer)
00596     {
00597         input_error (gettext ("Bad simulator program"));
00598         goto exit_on_error;
00599     }
00600     input->simulator = g_strdup (buffer);
00601
00602     // Opening evaluator program name
00603     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00604     if (!buffer)
00605         input->evaluator = g_strdup (buffer);
00606
00607     // Obtaining pseudo-random numbers generator seed
00608     input->seed
00609     = json_object_get_uint_with_default (object,
00610     LABEL_SEED,
00611     DEFAULT_RANDOM_SEED, &error_code);
00612     if (error_code)
00613     {
00614         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00615         goto exit_on_error;
00616     }
00617 }

```

```

00623
00624 // Opening algorithm
00625 buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00626 if (!strcmp (buffer, LABEL_MONTE_CARLO))
00627 {
00628     input->algorithm = ALGORITHM_MONTE_CARLO;
00629
00630     // Obtaining simulations number
00631     input->nsimulations
00632     = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00633 );
00634     if (error_code)
00635     {
00636         input_error (gettext ("Bad simulations number"));
00637         goto exit_on_error;
00638     }
00639     else if (!strcmp (buffer, LABEL_SWEEP))
00640         input->algorithm = ALGORITHM_SWEEP;
00641     else if (!strcmp (buffer, LABEL_GENETIC))
00642     {
00643         input->algorithm = ALGORITHM_GENETIC;
00644
00645         // Obtaining population
00646         if (json_object_get_member (object, LABEL_NPOPULATION))
00647         {
00648             input->nsimulations
00649             = json_object_get_uint (object,
00650 LABEL_NPOPULATION, &error_code);
00651             if (error_code || input->nsimulations < 3)
00652             {
00653                 input_error (gettext ("Invalid population number"));
00654                 goto exit_on_error;
00655             }
00656             else
00657             {
00658                 input_error (gettext ("No population number"));
00659                 goto exit_on_error;
00660             }
00661
00662             // Obtaining generations
00663             if (json_object_get_member (object, LABEL_NGENERATIONS))
00664             {
00665                 input->niterations
00666                 = json_object_get_uint (object,
00667 LABEL_NGENERATIONS, &error_code);
00668                 if (error_code || !input->niterations)
00669                 {
00670                     input_error (gettext ("Invalid generations number"));
00671                     goto exit_on_error;
00672                 }
00673             }
00674             else
00675             {
00676                 input_error (gettext ("No generations number"));
00677                 goto exit_on_error;
00678             }
00679
00680             // Obtaining mutation probability
00681             if (json_object_get_member (object, LABEL_MUTATION))
00682             {
00683                 input->mutation_ratio
00684                 = json_object_get_float (object, LABEL_MUTATION, &error_code
00685 );
00686                 if (error_code || input->mutation_ratio < 0.
00687 || input->mutation_ratio >= 1.)
00688                 {
00689                     input_error (gettext ("Invalid mutation probability"));
00690                     goto exit_on_error;
00691                 }
00692             }
00693             else
00694             {
00695                 input_error (gettext ("No mutation probability"));
00696                 goto exit_on_error;
00697             }
00698
00699             // Obtaining reproduction probability
00700             if (json_object_get_member (object, LABEL_REPRODUCTION))
00701             {
00702                 input->reproduction_ratio
00703                 = json_object_get_float (object,
00704 LABEL_REPRODUCTION, &error_code);
00705                 if (error_code || input->reproduction_ratio < 0.
00706 || input->reproduction_ratio >= 1.0)
00707                 {

```

```

00705         input_error (gettext ("Invalid reproduction probability"));
00706         goto exit_on_error;
00707     }
00708 }
00709 else
00710 {
00711     input_error (gettext ("No reproduction probability"));
00712     goto exit_on_error;
00713 }
00714
00715 // Obtaining adaptation probability
00716 if (json_object_get_member (object, LABEL_ADAPTATION))
00717 {
00718     input->adaptation_ratio
00719     = json_object_get_float (object,
00720 LABEL_ADAPTATION, &error_code);
00721     if (error_code || input->adaptation_ratio < 0.
00722         || input->adaptation_ratio >= 1.)
00723     {
00724         input_error (gettext ("Invalid adaptation probability"));
00725         goto exit_on_error;
00726     }
00727 }
00728 else
00729 {
00730     input_error (gettext ("No adaptation probability"));
00731     goto exit_on_error;
00732 }
00733
00734 // Checking survivals
00735 i = input->mutation_ratio * input->nsimulations;
00736 i += input->reproduction_ratio * input->nsimulations;
00737 i += input->adaptation_ratio * input->nsimulations;
00738 if (i > input->nsimulations - 2)
00739 {
00740     input_error
00741     (gettext
00742      ("No enough survival entities to reproduce the population"));
00743     goto exit_on_error;
00744 }
00745 else
00746 {
00747     input_error (gettext ("Unknown algorithm"));
00748     goto exit_on_error;
00749 }
00750
00751 if (input->algorithm == ALGORITHM_MONTE_CARLO
00752     || input->algorithm == ALGORITHM_SWEEP)
00753 {
00754
00755     // Obtaining iterations number
00756     input->niterations
00757     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
00758 );
00759     if (error_code == 1)
00760         input->niterations = 1;
00761     else if (error_code)
00762     {
00763         input_error (gettext ("Bad iterations number"));
00764         goto exit_on_error;
00765     }
00766
00767     // Obtaining best number
00768     input->nbest
00769     = json_object_get_uint_with_default (object,
00770 LABEL_NBEST, 1,
00771                                         &error_code);
00772     if (error_code || !input->nbest)
00773     {
00774         input_error (gettext ("Invalid best number"));
00775         goto exit_on_error;
00776     }
00777
00778     // Obtaining tolerance
00779     input->tolerance
00780     = json_object_get_float_with_default (object,
00781 LABEL_TOLERANCE, 0.,
00782                                         &error_code);
00783     if (error_code || input->tolerance < 0.)
00784     {
00785         input_error (gettext ("Invalid tolerance"));
00786         goto exit_on_error;
00787     }
00788
00789     // Getting direction search method parameters
00790     if (json_object_get_member (object, LABEL_NSTEPS))

```

```

00788     {
00789         input->nsteps
00790         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00791         if (error_code || !input->nsteps)
00792         {
00793             input_error (gettext ("Invalid steps number"));
00794             goto exit_on_error;
00795         }
00796         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00797         if (!strcmp (buffer, LABEL_COORDINATES))
00798             input->direction = DIRECTION_METHOD_COORDINATES;
00799         else if (!strcmp (buffer, LABEL_RANDOM))
00800         {
00801             input->direction = DIRECTION_METHOD_RANDOM;
00802             input->nestimates
00803             = json_object_get_uint (object,
00804 LABEL_NESTIMATES, &error_code);
00805             if (error_code || !input->nestimates)
00806             {
00807                 input_error (gettext ("Invalid estimates number"));
00808                 goto exit_on_error;
00809             }
00810         }
00811         else
00812         {
00813             input_error
00814             (gettext ("Unknown method to estimate the direction search"));
00815             goto exit_on_error;
00816         }
00817         input->relaxation
00818         = json_object_get_float_with_default (object,
00819 LABEL_RELAXATION,
00820                                             DEFAULT_RELAXATION,
00821                                             &error_code);
00822         if (error_code || input->relaxation < 0. || input->
00823 relaxation > 2.)
00824         {
00825             input_error (gettext ("Invalid relaxation parameter"));
00826             goto exit_on_error;
00827         }
00828         else
00829             input->nsteps = 0;
00830         // Obtaining the threshold
00831         input->threshold
00832         = json_object_get_float_with_default (object,
00833 LABEL_THRESHOLD, 0.,
00834                                             &error_code);
00835         if (error_code)
00836         {
00837             input_error (gettext ("Invalid threshold"));
00838             goto exit_on_error;
00839         }
00840         // Reading the experimental data
00841         array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00842         n = json_array_get_length (array);
00843         input->experiment = (Experiment *) g_malloc (n * sizeof (
00844 Experiment));
00845         for (i = 0; i < n; ++i)
00846         {
00847             #if DEBUG_INPUT
00848             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00849                     input->nexperiments);
00850             #endif
00851             child = json_array_get_element (array, i);
00852             if (!input->nexperiments)
00853             {
00854                 if (!experiment_open_json (input->experiment, child, 0))
00855                     goto exit_on_error;
00856             }
00857             else
00858             {
00859                 if (!experiment_open_json (input->experiment + input->
00860 nexperiments,
00861                                         child, input->experiment->ninputs))
00862                     goto exit_on_error;
00863             }
00864             ++input->nexperiments;
00865             #if DEBUG_INPUT
00866             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00867                     input->nexperiments);
00868             #endif
00869         }
00870         if (!input->nexperiments)
00871         {

```

```

00869     input_error (gettext ("No optimization experiments"));
00870     goto exit_on_error;
00871 }
00872
00873 // Reading the variables data
00874 array = json_object_get_array_member (object, LABEL_VARIABLES);
00875 n = json_array_get_length (array);
00876 input->variable = (Variable *) g_malloc (n * sizeof (Variable));
00877 for (i = 0; i < n; ++i)
00878 {
00879 #if DEBUG_INPUT
00880     fprintf (stderr, "input_open_json: nvariables=%u\n", input->nvariables);
00881 #endif
00882     child = json_array_get_element (array, i);
00883     if (!variable_open_json (input->variable + input->
nvariables, child,
                                input->algorithm, input->nsteps))
00884         goto exit_on_error;
00885     ++input->nvariables;
00886 }
00887 if (!input->nvariables)
00888 {
00889     input_error (gettext ("No optimization variables"));
00890     goto exit_on_error;
00891 }
00892
00893 // Obtaining the error norm
00894 if (json_object_get_member (object, LABEL_NORM))
00895 {
00896     buffer = json_object_get_string_member (object, LABEL_NORM);
00897     if (!strcmp (buffer, LABEL_EUCLIDIAN))
00898         input->norm = ERROR_NORM_EUCLIDIAN;
00899     else if (!strcmp (buffer, LABEL_MAXIMUM))
00900         input->norm = ERROR_NORM_MAXIMUM;
00901     else if (!strcmp (buffer, LABEL_P))
00902     {
00903         input->norm = ERROR_NORM_P;
00904         input->p = json_object_get_float (object,
LABEL_P, &error_code);
00905         if (!error_code)
00906         {
00907             input_error (gettext ("Bad P parameter"));
00908             goto exit_on_error;
00909         }
00910     }
00911     else if (!strcmp (buffer, LABEL_TAXICAB))
00912         input->norm = ERROR_NORM_TAXICAB;
00913     else
00914     {
00915         input_error (gettext ("Unknown error norm"));
00916         goto exit_on_error;
00917     }
00918 }
00919 else
00920     input->norm = ERROR_NORM_EUCLIDIAN;
00921
00922 // Closing the JSON document
00923 g_object_unref (parser);
00924
00925 #if DEBUG_INPUT
00926     fprintf (stderr, "input_open_json: end\n");
00927 #endif
00928 return 1;
00929
00930 exit_on_error:
00931 g_object_unref (parser);
00932 #if DEBUG_INPUT
00933     fprintf (stderr, "input_open_json: end\n");
00934 #endif
00935 return 0;
00936 }
00937
00938 int
00939 input_open (char *filename)
00940 {
00941     xmlDoc *doc;
00942     JsonParser *parser;
00943
00944 #if DEBUG_INPUT
00945     fprintf (stderr, "input_open: start\n");
00946 #endif
00947 // Resetting input data
00948 input_new ();
00949
00950 // Opening input file
00951 #if DEBUG_INPUT

```

```

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

```

5.9 input.h File Reference

Header file to define the input functions.

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

Data Structures

- struct [Input](#)

Struct to define the optimization input file.

Enumerations

- enum [DirectionMethod](#) { [DIRECTION_METHOD_COORDINATES](#) = 0, [DIRECTION_METHOD_RANDOM](#) = 1 }

Enum to define the methods to estimate the direction search.

- enum [ErrorNorm](#) { [ERROR_NORM_EUCLIDIAN](#) = 0, [ERROR_NORM_MAXIMUM](#) = 1, [ERROR_NORM_P](#) = 2, [ERROR_NORM_TAXICAB](#) = 3 }

Enum to define the error norm.

Functions

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

Variables

- [Input](#) [input](#) [1]
Global [Input](#) struct to set the input data.
- const char * [result_name](#)
Name of the result file.
- const char * [variables_name](#)
Name of the variables file.

5.9.1 Detailed Description

Header file to define the input functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.9.2 Enumeration Type Documentation

5.9.2.1 enum DirectionMethod

Enum to define the methods to estimate the direction search.

Enumerator

[DIRECTION_METHOD_COORDINATES](#) Coordinates descent method.
[DIRECTION_METHOD_RANDOM](#) Random method.

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

```
00046 {  
00047     DIRECTION\_METHOD\_COORDINATES = 0,  
00048     DIRECTION\_METHOD\_RANDOM = 1,  
00049 };
```

5.9.2.2 enum ErrorNorm

Enum to define the error norm.

Enumerator

ERROR_NORM_EUCLIDIAN Euclidian norm: $\sqrt{\sum_i (w_i x_i)^2}$.
ERROR_NORM_MAXIMUM Maximum norm: $\max_i |w_i x_i|$.
ERROR_NORM_P P-norm $\sqrt[p]{\sum_i |w_i x_i|^p}$.
ERROR_NORM_TAXICAB Taxicab norm $\sum_i |w_i x_i|$.

Definition at line 55 of file [input.h](#).

```
00056 {
00057     ERROR_NORM_EUCLIDIAN = 0,
00059     ERROR_NORM_MAXIMUM = 1,
00061     ERROR_NORM_P = 2,
00063     ERROR_NORM_TAXICAB = 3
00065 };
```

5.9.3 Function Documentation

5.9.3.1 void input_error (char * message)

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

Parameters

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

Definition at line 124 of file [input.c](#).

```
00125 {
00126     char buffer[64];
00127     snprintf (buffer, 64, "%s: %s\n", gettext ("Input"), message);
00128     error_message = g_strdup (buffer);
00129 }
```

5.9.3.2 int input_open (char * filename)

Function to open the input file.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 947 of file [input.c](#).

```

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

```

Here is the call graph for this function:

5.9.3.3 int input_open_json (JsonParser * parser)

Function to open the input file in JSON format.

Parameters

<i>parser</i>	JsonParser struct.
---------------	--------------------

Returns

1_on_success, 0_on_error.

Definition at line 557 of file [input.c](#).

```

00558 {
00559     JsonNode *node, *child;
00560     JsonObject *object;
00561     JsonArray *array;
00562     const char *buffer;
00563     int error_code;
00564     unsigned int i, n;
00565
00566     #if DEBUG_INPUT
00567         fprintf (stderr, "input_open_json: start\n");
00568     #endif
00569
00570     // Resetting input data
00571     input->type = INPUT_TYPE_JSON;
00572
00573     // Getting the root node
00574     #if DEBUG_INPUT
00575         fprintf (stderr, "input_open_json: getting the root node\n");
00576     #endif
00577     node = json_parser_get_root (parser);
00578     object = json_node_get_object (node);
00579
00580     // Getting result and variables file names
00581     if (!input->result)
00582     {
00583         buffer = json_object_get_string_member (object, LABEL_RESULT_FILE);
00584         if (!buffer)
00585             buffer = result_name;
00586         input->result = g_strdup (buffer);
00587     }
00588     else
00589         input->result = g_strdup (result_name);
00590     if (!input->variables)
00591     {
00592         buffer = json_object_get_string_member (object, LABEL_VARIABLES_FILE);
00593         if (!buffer)
00594             buffer = variables_name;
00595         input->variables = g_strdup (buffer);
00596     }
00597     else
00598         input->variables = g_strdup (variables_name);
00599
00600     // Opening simulator program name
00601     buffer = json_object_get_string_member (object, LABEL_SIMULATOR);
00602     if (!buffer)
00603     {
00604         input_error (gettext ("Bad simulator program"));
00605         goto exit_on_error;
00606     }
00607     input->simulator = g_strdup (buffer);
00608
00609     // Opening evaluator program name
00610     buffer = json_object_get_string_member (object, LABEL_EVALUATOR);
00611     if (buffer)
00612         input->evaluator = g_strdup (buffer);
00613
00614     // Obtaining pseudo-random numbers generator seed
00615     input->seed
00616     = json_object_get_uint_with_default (object,
00617     LABEL_SEED,
00618     DEFAULT_RANDOM_SEED, &error_code);
00619     if (error_code)
00620     {
00621         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00622         goto exit_on_error;
00623     }
00624
00625     // Opening algorithm
00626     buffer = json_object_get_string_member (object, LABEL_ALGORITHM);
00627     if (!strcmp (buffer, LABEL_MONTE_CARLO))
00628     {
00629         input->algorithm = ALGORITHM_MONTE_CARLO;
00630
00631         // Obtaining simulations number
00632         input->nsimulations
00633         = json_object_get_int (object, LABEL_NSIMULATIONS, &error_code
00634 );
00635         if (error_code)
00636         {
00637             input_error (gettext ("Bad simulations number"));
00638             goto exit_on_error;
00639         }
00640     }
00641     else if (!strcmp (buffer, LABEL_SWEEP))
00642         input->algorithm = ALGORITHM_SWEEP;
00643     else if (!strcmp (buffer, LABEL_GENETIC))
00644         {

```

```

00643     input->algorithm = ALGORITHM_GENETIC;
00644
00645     // Obtaining population
00646     if (json_object_get_member (object, LABEL_NPOPULATION))
00647     {
00648         input->nsimulations
00649         = json_object_get_uint (object,
00650 LABEL_NPOPULATION, &error_code);
00651         if (error_code || input->nsimulations < 3)
00652         {
00653             input_error (gettext ("Invalid population number"));
00654             goto exit_on_error;
00655         }
00656     }
00657     else
00658     {
00659         input_error (gettext ("No population number"));
00660         goto exit_on_error;
00661     }
00662     // Obtaining generations
00663     if (json_object_get_member (object, LABEL_NGENERATIONS))
00664     {
00665         input->niterations
00666         = json_object_get_uint (object,
00667 LABEL_NGENERATIONS, &error_code);
00668         if (error_code || !input->niterations)
00669         {
00670             input_error (gettext ("Invalid generations number"));
00671             goto exit_on_error;
00672         }
00673     }
00674     else
00675     {
00676         input_error (gettext ("No generations number"));
00677         goto exit_on_error;
00678     }
00679     // Obtaining mutation probability
00680     if (json_object_get_member (object, LABEL_MUTATION))
00681     {
00682         input->mutation_ratio
00683         = json_object_get_float (object, LABEL_MUTATION, &error_code
00684 );
00685         if (error_code || input->mutation_ratio < 0.
00686             || input->mutation_ratio >= 1.)
00687         {
00688             input_error (gettext ("Invalid mutation probability"));
00689             goto exit_on_error;
00690         }
00691     }
00692     else
00693     {
00694         input_error (gettext ("No mutation probability"));
00695         goto exit_on_error;
00696     }
00697     // Obtaining reproduction probability
00698     if (json_object_get_member (object, LABEL_REPRODUCTION))
00699     {
00700         input->reproduction_ratio
00701         = json_object_get_float (object,
00702 LABEL_REPRODUCTION, &error_code);
00703         if (error_code || input->reproduction_ratio < 0.
00704             || input->reproduction_ratio >= 1.0)
00705         {
00706             input_error (gettext ("Invalid reproduction probability"));
00707             goto exit_on_error;
00708         }
00709     }
00710     else
00711     {
00712         input_error (gettext ("No reproduction probability"));
00713         goto exit_on_error;
00714     }
00715     // Obtaining adaptation probability
00716     if (json_object_get_member (object, LABEL_ADAPTATION))
00717     {
00718         input->adaptation_ratio
00719         = json_object_get_float (object,
00720 LABEL_ADAPTATION, &error_code);
00721         if (error_code || input->adaptation_ratio < 0.
00722             || input->adaptation_ratio >= 1.)
00723         {
00724             input_error (gettext ("Invalid adaptation probability"));
00725             goto exit_on_error;

```

```

00725         }
00726     }
00727     else
00728     {
00729         input_error (gettext ("No adaptation probability"));
00730         goto exit_on_error;
00731     }
00732
00733     // Checking survivals
00734     i = input->mutation_ratio * input->nsimulations;
00735     i += input->reproduction_ratio * input->
nsimulations;
00736     i += input->adaptation_ratio * input->
nsimulations;
00737     if (i > input->nsimulations - 2)
00738     {
00739         input_error
00740         (gettext
00741         ("No enough survival entities to reproduce the population"));
00742         goto exit_on_error;
00743     }
00744 }
00745 else
00746 {
00747     input_error (gettext ("Unknown algorithm"));
00748     goto exit_on_error;
00749 }
00750
00751 if (input->algorithm == ALGORITHM_MONTE_CARLO
|| input->algorithm == ALGORITHM_SWEEP)
00752 {
00753     // Obtaining iterations number
00754     input->niterations
00755     = json_object_get_uint (object, LABEL_NITERATIONS, &error_code
);
00756     if (error_code == 1)
00757         input->niterations = 1;
00758     else if (error_code)
00759     {
00760         input_error (gettext ("Bad iterations number"));
00761         goto exit_on_error;
00762     }
00763
00764     // Obtaining best number
00765     input->nbest
00766     = json_object_get_uint_with_default (object,
LABEL_NBEST, 1,
00767                                         &error_code);
00768     if (error_code || !input->nbest)
00769     {
00770         input_error (gettext ("Invalid best number"));
00771         goto exit_on_error;
00772     }
00773
00774     // Obtaining tolerance
00775     input->tolerance
00776     = json_object_get_float_with_default (object,
LABEL_TOLERANCE, 0.,
00777                                         &error_code);
00778     if (error_code || input->tolerance < 0.)
00779     {
00780         input_error (gettext ("Invalid tolerance"));
00781         goto exit_on_error;
00782     }
00783
00784     // Getting direction search method parameters
00785     if (json_object_get_member (object, LABEL_NSTEPS))
00786     {
00787         input->nsteps
00788         = json_object_get_uint (object, LABEL_NSTEPS, &error_code);
00789         if (error_code || !input->nsteps)
00790         {
00791             input_error (gettext ("Invalid steps number"));
00792             goto exit_on_error;
00793         }
00794         buffer = json_object_get_string_member (object, LABEL_DIRECTION);
00795         if (!strcmp (buffer, LABEL_COORDINATES))
00796             input->direction = DIRECTION_METHOD_COORDINATES;
00797         else if (!strcmp (buffer, LABEL_RANDOM))
00798         {
00799             input->direction = DIRECTION_METHOD_RANDOM;
00800             input->nestimates
00801             = json_object_get_uint (object,
LABEL_NESTIMATES, &error_code);
00802             if (error_code || !input->nestimates)
00803             {

```

```

00806             input_error (gettext ("Invalid estimates number"));
00807             goto exit_on_error;
00808         }
00809     }
00810     else
00811     {
00812         input_error
00813         (gettext ("Unknown method to estimate the direction search"));
00814         goto exit_on_error;
00815     }
00816     input->relaxation
00817     = json_object_get_float_with_default (object,
00818     LABEL_RELAXATION,
00819     DEFAULT_RELAXATION,
00820     &error_code);
00821     if (error_code || input->relaxation < 0. || input->
00822     relaxation > 2.)
00823     {
00824         input_error (gettext ("Invalid relaxation parameter"));
00825         goto exit_on_error;
00826     }
00827     else
00828     {
00829         input->nsteps = 0;
00830     }
00831     // Obtaining the threshold
00832     input->threshold
00833     = json_object_get_float_with_default (object,
00834     LABEL_THRESHOLD, 0.,
00835     &error_code);
00836     if (error_code)
00837     {
00838         input_error (gettext ("Invalid threshold"));
00839         goto exit_on_error;
00840     }
00841     // Reading the experimental data
00842     array = json_object_get_array_member (object, LABEL_EXPERIMENTS);
00843     n = json_array_get_length (array);
00844     input->experiment = (Experiment *) g_malloc (n * sizeof (
00845     Experiment));
00846     for (i = 0; i < n; ++i)
00847     {
00848         #if DEBUG_INPUT
00849             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00850             input->nexperiments);
00851         #endif
00852         child = json_array_get_element (array, i);
00853         if (!input->nexperiments)
00854         {
00855             if (!experiment_open_json (input->experiment, child, 0))
00856                 goto exit_on_error;
00857         }
00858         else
00859         {
00860             if (!experiment_open_json (input->experiment +
00861             input->nexperiments,
00862             child, input->experiment->
00863             ninputs))
00864                 goto exit_on_error;
00865         }
00866         ++input->nexperiments;
00867         #if DEBUG_INPUT
00868             fprintf (stderr, "input_open_json: nexperiments=%u\n",
00869             input->nexperiments);
00870         #endif
00871     }
00872     if (!input->nexperiments)
00873     {
00874         input_error (gettext ("No optimization experiments"));
00875         goto exit_on_error;
00876     }
00877     // Reading the variables data
00878     array = json_object_get_array_member (object, LABEL_VARIABLES);
00879     n = json_array_get_length (array);
00880     input->variable = (Variable *) g_malloc (n * sizeof (
00881     Variable));
00882     for (i = 0; i < n; ++i)
00883     {
00884         #if DEBUG_INPUT
00885             fprintf (stderr, "input_open_json: nvariables=%u\n", input->
00886             nvariables);
00887         #endif
00888         child = json_array_get_element (array, i);
00889         if (!variable_open_json (input->variable +
00890         input->nvariables, child,

```

```

00884                                     input->algorithm, input->
nsteps))
00885         goto exit_on_error;
00886         ++input->nvariables;
00887     }
00888     if (!input->nvariables)
00889     {
00890         input_error (gettext ("No optimization variables"));
00891         goto exit_on_error;
00892     }
00893
00894     // Obtaining the error norm
00895     if (json_object_get_member (object, LABEL_NORM))
00896     {
00897         buffer = json_object_get_string_member (object, LABEL_NORM);
00898         if (!strcmp (buffer, LABEL_EUCLIDIAN))
00899             input->norm = ERROR_NORM_EUCLIDIAN;
00900         else if (!strcmp (buffer, LABEL_MAXIMUM))
00901             input->norm = ERROR_NORM_MAXIMUM;
00902         else if (!strcmp (buffer, LABEL_P))
00903         {
00904             input->norm = ERROR_NORM_P;
00905             input->p = json_object_get_float (object,
LABEL_P, &error_code);
00906             if (!error_code)
00907             {
00908                 input_error (gettext ("Bad P parameter"));
00909                 goto exit_on_error;
00910             }
00911             else if (!strcmp (buffer, LABEL_TAXICAB))
00912                 input->norm = ERROR_NORM_TAXICAB;
00913             else
00914             {
00915                 input_error (gettext ("Unknown error norm"));
00916                 goto exit_on_error;
00917             }
00918         }
00919     }
00920     else
00921         input->norm = ERROR_NORM_EUCLIDIAN;
00922
00923     // Closing the JSON document
00924     g_object_unref (parser);
00925
00926     #if DEBUG_INPUT
00927     fprintf (stderr, "input_open_json: end\n");
00928     #endif
00929     return 1;
00930
00931 exit_on_error:
00932     g_object_unref (parser);
00933     #if DEBUG_INPUT
00934     fprintf (stderr, "input_open_json: end\n");
00935     #endif
00936     return 0;
00937 }

```

Here is the call graph for this function:

5.9.3.4 int input_open_xml (xmlDoc * doc)

Function to open the input file in XML format.

Parameters

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

Returns

1_on_success, 0_on_error.

Definition at line 139 of file [input.c](#).


```

00140 {
00141     char buffer2[64];
00142     xmlNode *node, *child;
00143     xmlChar *buffer;
00144     int error_code;
00145     unsigned int i;
00146
00147     #if DEBUG_INPUT
00148         fprintf (stderr, "input_open_xml: start\n");
00149     #endif
00150
00151     // Resetting input data
00152     buffer = NULL;
00153     input->type = INPUT_TYPE_XML;
00154
00155     // Getting the root node
00156     #if DEBUG_INPUT
00157         fprintf (stderr, "input_open_xml: getting the root node\n");
00158     #endif
00159     node = xmlDocGetRootElement (doc);
00160     if (xmlStrcmp (node->name, (const xmlChar *) LABEL_OPTIMIZE))
00161     {
00162         input_error (gettext ("Bad root XML node"));
00163         goto exit_on_error;
00164     }
00165
00166     // Getting result and variables file names
00167     if (!input->result)
00168     {
00169         input->result =
00170             (char *) xmlGetProp (node, (const xmlChar *) LABEL_RESULT_FILE);
00171         if (!input->result)
00172             input->result = (char *) xmlStrdup ((const xmlChar *)
result_name);
00173     }
00174     if (!input->variables)
00175     {
00176         input->variables =
00177             (char *) xmlGetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE);
00178         if (!input->variables)
00179             input->variables =
00180                 (char *) xmlStrdup ((const xmlChar *) variables_name);
00181     }
00182
00183     // Opening simulator program name
00184     input->simulator =
00185         (char *) xmlGetProp (node, (const xmlChar *) LABEL_SIMULATOR);
00186     if (!input->simulator)
00187     {
00188         input_error (gettext ("Bad simulator program"));
00189         goto exit_on_error;
00190     }
00191
00192     // Opening evaluator program name
00193     input->evaluator =
00194         (char *) xmlGetProp (node, (const xmlChar *) LABEL_EVALUATOR);
00195
00196     // Obtaining pseudo-random numbers generator seed
00197     input->seed
00198     = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_SEED,
                                DEFAULT_RANDOM_SEED, &error_code);
00199     if (error_code)
00200     {
00201         input_error (gettext ("Bad pseudo-random numbers generator seed"));
00202         goto exit_on_error;
00203     }
00204
00205     // Opening algorithm
00206     buffer = xmlGetProp (node, (const xmlChar *) LABEL_ALGORITHM);
00207     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_MONTE_CARLO))
00208     {
00209         input->algorithm = ALGORITHM_MONTE_CARLO;
00210
00211         // Obtaining simulations number
00212         input->nsimulations
00213         = xml_node_get_int (node, (const xmlChar *)
LABEL_NSIMULATIONS,
                                &error_code);
00214         if (error_code)
00215         {
00216             input_error (gettext ("Bad simulations number"));
00217             goto exit_on_error;
00218         }
00219     }
00220     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_SWEEP))
00221     {
00222         input->algorithm = ALGORITHM_SWEEP;
00223     }

```

```

00224     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_GENETIC))
00225     {
00226         input->algorithm = ALGORITHM_GENETIC;
00227
00228         // Obtaining population
00229         if (xmlHasProp (node, (const xmlChar *) LABEL_NPOPULATION))
00230         {
00231             input->nsimulations
00232             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NPOPULATION,
00233                                 &error_code);
00234             if (error_code || input->nsimulations < 3)
00235             {
00236                 input_error (gettext ("Invalid population number"));
00237                 goto exit_on_error;
00238             }
00239         }
00240         else
00241         {
00242             input_error (gettext ("No population number"));
00243             goto exit_on_error;
00244         }
00245
00246         // Obtaining generations
00247         if (xmlHasProp (node, (const xmlChar *) LABEL_NGENERATIONS))
00248         {
00249             input->niterations
00250             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NGENERATIONS,
00251                                 &error_code);
00252             if (error_code || !input->niterations)
00253             {
00254                 input_error (gettext ("Invalid generations number"));
00255                 goto exit_on_error;
00256             }
00257         }
00258         else
00259         {
00260             input_error (gettext ("No generations number"));
00261             goto exit_on_error;
00262         }
00263
00264         // Obtaining mutation probability
00265         if (xmlHasProp (node, (const xmlChar *) LABEL_MUTATION))
00266         {
00267             input->mutation_ratio
00268             = xml_node_get_float (node, (const xmlChar *)
LABEL_MUTATION,
00269                                 &error_code);
00270             if (error_code || input->mutation_ratio < 0.
|| input->mutation_ratio >= 1.)
00271             {
00272                 input_error (gettext ("Invalid mutation probability"));
00273                 goto exit_on_error;
00274             }
00275         }
00276         else
00277         {
00278             input_error (gettext ("No mutation probability"));
00279             goto exit_on_error;
00280         }
00281
00282         // Obtaining reproduction probability
00283         if (xmlHasProp (node, (const xmlChar *) LABEL_REPRODUCTION))
00284         {
00285             input->reproduction_ratio
00286             = xml_node_get_float (node, (const xmlChar *)
LABEL_REPRODUCTION,
00287                                 &error_code);
00288             if (error_code || input->reproduction_ratio < 0.
|| input->reproduction_ratio >= 1.0)
00289             {
00290                 input_error (gettext ("Invalid reproduction probability"));
00291                 goto exit_on_error;
00292             }
00293         }
00294         else
00295         {
00296             input_error (gettext ("No reproduction probability"));
00297             goto exit_on_error;
00298         }
00299
00300         // Obtaining adaptation probability
00301         if (xmlHasProp (node, (const xmlChar *) LABEL_ADAPTATION))
00302         {
00303             input->adaptation_ratio
00304             = xml_node_get_float (node, (const xmlChar *)

```

```

    LABEL_ADAPTATION,
00307                                     &error_code);
00308         if (error_code || input->adaptation_ratio < 0.
00309             || input->adaptation_ratio >= 1.)
00310         {
00311             input_error (gettext ("Invalid adaptation probability"));
00312             goto exit_on_error;
00313         }
00314     }
00315     else
00316     {
00317         input_error (gettext ("No adaptation probability"));
00318         goto exit_on_error;
00319     }
00320
00321     // Checking survivals
00322     i = input->mutation_ratio * input->nsimulations;
00323     i += input->reproduction_ratio * input->
nsimulations;
00324     i += input->adaptation_ratio * input->
nsimulations;
00325     if (i > input->nsimulations - 2)
00326     {
00327         input_error
00328         (gettext
00329          ("No enough survival entities to reproduce the population"));
00330         goto exit_on_error;
00331     }
00332 }
00333 else
00334 {
00335     input_error (gettext ("Unknown algorithm"));
00336     goto exit_on_error;
00337 }
00338 xmlFree (buffer);
00339 buffer = NULL;
00340
00341 if (input->algorithm == ALGORITHM_MONTE_CARLO
00342     || input->algorithm == ALGORITHM_SWEEP)
00343 {
00344     // Obtaining iterations number
00345     input->niterations
00346     = xml_node_get_uint (node, (const xmlChar *)
LABEL_NITERATIONS,
00347                         &error_code);
00348     if (error_code == 1)
00349         input->niterations = 1;
00350     else if (error_code)
00351     {
00352         input_error (gettext ("Bad iterations number"));
00353         goto exit_on_error;
00354     }
00355 }
00356
00357 // Obtaining best number
00358 input->nbest
00359 = xml_node_get_uint_with_default (node, (const xmlChar *)
LABEL_NBEST,
00360                                   1, &error_code);
00361 if (error_code || !input->nbest)
00362 {
00363     input_error (gettext ("Invalid best number"));
00364     goto exit_on_error;
00365 }
00366
00367 // Obtaining tolerance
00368 input->tolerance
00369 = xml_node_get_float_with_default (node,
00370                                   (const xmlChar *) LABEL_TOLERANCE,
00371                                   0., &error_code);
00372 if (error_code || input->tolerance < 0.)
00373 {
00374     input_error (gettext ("Invalid tolerance"));
00375     goto exit_on_error;
00376 }
00377
00378 // Getting direction search method parameters
00379 if (xmlHasProp (node, (const xmlChar *) LABEL_NSTEPS))
00380 {
00381     input->nsteps =
00382     xml_node_get_uint (node, (const xmlChar *)
LABEL_NSTEPS,
00383                       &error_code);
00384     if (error_code || !input->nsteps)
00385     {
00386         input_error (gettext ("Invalid steps number"));
00387         goto exit_on_error;

```

```

00388     }
00389     buffer = xmlGetProp (node, (const xmlChar *) LABEL_DIRECTION);
00390     if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_COORDINATES))
00391         input->direction = DIRECTION_METHOD_COORDINATES;
00392     else if (!xmlStrcmp (buffer, (const xmlChar *) LABEL_RANDOM))
00393     {
00394         input->direction = DIRECTION_METHOD_RANDOM;
00395         input->nestimates
00396             = xml_node_get_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
00397                                 &error_code);
00398         if (error_code || !input->nestimates)
00399         {
00400             input_error (gettext ("Invalid estimates number"));
00401             goto exit_on_error;
00402         }
00403     }
00404     else
00405     {
00406         input_error
00407             (gettext ("Unknown method to estimate the direction search"));
00408         goto exit_on_error;
00409     }
00410     xmlFree (buffer);
00411     buffer = NULL;
00412     input->relaxation
00413         = xml_node_get_float_with_default (node,
00414                                           (const xmlChar *)
LABEL_RELAXATION,
00415                                           DEFAULT_RELAXATION, &error_code);
00416     if (error_code || input->relaxation < 0. || input->
relaxation > 2.)
00417     {
00418         input_error (gettext ("Invalid relaxation parameter"));
00419         goto exit_on_error;
00420     }
00421     }
00422     }
00423     else
00424         input->nsteps = 0;
00425     }
00426     // Obtaining the threshold
00427     input->threshold =
00428         xml_node_get_float_with_default (node, (const xmlChar *)
LABEL_THRESHOLD,
00429                                         0., &error_code);
00430     if (error_code)
00431     {
00432         input_error (gettext ("Invalid threshold"));
00433         goto exit_on_error;
00434     }
00435     // Reading the experimental data
00436     for (child = node->children; child; child = child->next)
00437     {
00438         if (xmlStrcmp (child->name, (const xmlChar *) LABEL_EXPERIMENT))
00439             break;
00440         #if DEBUG_INPUT
00441         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00442                 input->nexperiments);
00443         #endif
00444         input->experiment = (Experiment *)
00445             g_realloc (input->experiment,
00446                       (1 + input->nexperiments) * sizeof (
Experiment));
00447         if (!input->nexperiments)
00448         {
00449             if (!experiment_open_xml (input->experiment, child, 0))
00450                 goto exit_on_error;
00451         }
00452         else
00453         {
00454             if (!experiment_open_xml (input->experiment +
input->nexperiments,
00455                                     child, input->experiment->
ninputs))
00456                 goto exit_on_error;
00457         }
00458         ++input->nexperiments;
00459         #if DEBUG_INPUT
00460         fprintf (stderr, "input_open_xml: nexperiments=%u\n",
00461                 input->nexperiments);
00462         #endif
00463     }
00464     if (!input->nexperiments)
00465     {
00466         input_error (gettext ("No optimization experiments"));
00467         goto exit_on_error;
00468     }

```

```

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

```

Here is the call graph for this function:

5.10 input.h

```

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

```

```

00111 void input_new ();
00112 void input_free ();
00113 void input_error (char *message);
00114 int input_open_xml (xmlDoc * doc);
00115 int input_open_json (JsonParser * parser);
00116 int input_open (char *filename);
00117
00118 #endif

```

5.11 interface.c File Reference

Source file to define the graphical interface functions.

```

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

```

Include dependency graph for interface.c:

Macros

- #define **_GNU_SOURCE**
- #define **DEBUG_INTERFACE** 0
Macro to debug interface functions.
- #define **INPUT_FILE** "test-ga.xml"
Macro to define the initial input file.

Functions

- void **input_save_direction_xml** (xmlNode *node)
Function to save the direction search method data in a XML node.
- void **input_save_direction_json** (JsonNode *node)
Function to save the direction search method data in a JSON node.
- void **input_save_xml** (xmlDoc *doc)
Function to save the input file in XML format.
- void **input_save_json** (JsonGenerator *generator)

- Function to save the input file in JSON format.*

 - void [input_save](#) (char *filename)

Function to save the input file.
- void [options_new](#) ()

Function to open the options dialog.
- void [running_new](#) ()

Function to open the running dialog.
- unsigned int [window_get_algorithm](#) ()

Function to get the stochastic algorithm number.
- unsigned int [window_get_direction](#) ()

Function to get the direction search method number.
- unsigned int [window_get_norm](#) ()

Function to get the norm method number.
- void [window_save_direction](#) ()

Function to save the direction search method data in the input file.
- int [window_save](#) ()

Function to save the input file.
- void [window_run](#) ()

Function to run a optimization.
- void [window_help](#) ()

Function to show a help dialog.
- void [window_about](#) ()

Function to show an about dialog.
- void [window_update_direction](#) ()

Function to update direction search method widgets view in the main window.
- void [window_update](#) ()

Function to update the main window view.
- void [window_set_algorithm](#) ()

Function to avoid memory errors changing the algorithm.
- void [window_set_experiment](#) ()

Function to set the experiment data in the main window.
- void [window_remove_experiment](#) ()

Function to remove an experiment in the main window.
- void [window_add_experiment](#) ()

Function to add an experiment in the main window.
- void [window_name_experiment](#) ()

Function to set the experiment name in the main window.
- void [window_weight_experiment](#) ()

Function to update the experiment weight in the main window.
- void [window_inputs_experiment](#) ()

Function to update the experiment input templates number in the main window.
- void [window_template_experiment](#) (void *data)

Function to update the experiment i-th input template in the main window.
- void [window_set_variable](#) ()

Function to set the variable data in the main window.
- void [window_remove_variable](#) ()

Function to remove a variable in the main window.
- void [window_add_variable](#) ()

Function to add a variable in the main window.
- void [window_label_variable](#) ()

Function to set the variable label in the main window.

- void [window_precision_variable](#) ()
Function to update the variable precision in the main window.
- void [window_rangemin_variable](#) ()
Function to update the variable rangemin in the main window.
- void [window_rangemax_variable](#) ()
Function to update the variable rangemax in the main window.
- void [window_rangeminabs_variable](#) ()
Function to update the variable rangeminabs in the main window.
- void [window_rangemaxabs_variable](#) ()
Function to update the variable rangemaxabs in the main window.
- void [window_step_variable](#) ()
Function to update the variable step in the main window.
- void [window_update_variable](#) ()
Function to update the variable data in the main window.
- int [window_read](#) (char *filename)
Function to read the input data of a file.
- void [window_open](#) ()
Function to open the input data.
- void [window_new](#) ()
Function to open the main window.

Variables

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

5.11.1 Detailed Description

Source file to define the graphical interface functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.11.2 Function Documentation

5.11.2.1 void input_save (char * filename)

Function to save the input file.

Parameters

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

Definition at line 577 of file [interface.c](#).

```

00578 {
00579     xmlDoc *doc;
00580     JsonGenerator *generator;
00581
00582     #if DEBUG_INTERFACE
00583         fprintf (stderr, "input_save: start\n");
00584     #endif
00585
00586     // Getting the input file directory
00587     input->name = g_path_get_basename (filename);
00588     input->directory = g_path_get_dirname (filename);
00589
00590     if (input->type == INPUT_TYPE_XML)
00591     {
00592         // Opening the input file
00593         doc = xmlNewDoc ((const xmlChar *) "1.0");
00594         input_save_xml (doc);
00595
00596         // Saving the XML file
00597         xmlSaveFormatFile (filename, doc, 1);
00598
00599         // Freeing memory
00600         xmlFreeDoc (doc);
00601     }
00602     else
00603     {
00604         // Opening the input file
00605         generator = json_generator_new ();
00606         json_generator_set_pretty (generator, TRUE);
00607         input_save_json (generator);
00608
00609         // Saving the JSON file
00610         json_generator_to_file (generator, filename, NULL);
00611
00612         // Freeing memory
00613         g_object_unref (generator);
00614     }
00615
00616     #if DEBUG_INTERFACE
00617         fprintf (stderr, "input_save: end\n");
00618     #endif
00619 }

```

Here is the call graph for this function:

5.11.2.2 void input_save_direction_json (JsonNode * node)

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

Parameters

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

Definition at line 209 of file [interface.c](#).

```

00210 {
00211     JsonObject *object;
00212     #if DEBUG_INTERFACE
00213         fprintf (stderr, "input_save_direction_json: start\n");
00214     #endif
00215     object = json_node_get_object (node);
00216     if (input->nsteps)

```

```

00217     {
00218         json_object_set_uint (object, LABEL_NSTEPS,
00219 input->nsteps);
00219         if (input->relaxation != DEFAULT_RELAXATION)
00220             json_object_set_float (object, LABEL_RELAXATION,
00221 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);
00229             json_object_set_uint (object, LABEL_NESTIMATES,
00230 input->nestimates);
00230         }
00231     }
00232     #if DEBUG_INTERFACE
00233     fprintf (stderr, "input_save_direction_json: end\n");
00234     #endif
00235 }

```

Here is the call graph for this function:

5.11.2.3 void input_save_direction_xml (xmlNode * node)

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

Parameters

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

Definition at line 173 of file [interface.c](#).

```

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

```

Here is the call graph for this function:

5.11.2.4 void input_save_json (JsonGenerator * generator)

Function to save the input file in JSON format.

Parameters

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

Definition at line 414 of file [interface.c](#).

```

00415 {
00416     unsigned int i, j;
00417     char *buffer;
00418     JsonNode *node, *child;
00419     JsonObject *object, *object2;
00420     JsonArray *array;
00421     GFile *file, *file2;
00422
00423     #if DEBUG_INTERFACE
00424     fprintf (stderr, "input_save_json: start\n");
00425     #endif
00426
00427     // Setting root JSON node
00428     node = json_node_new (JSON_NODE_OBJECT);
00429     object = json_node_get_object (node);
00430     json_generator_set_root (generator, node);
00431
00432     // Adding properties to the root JSON node
00433     if (strcmp (input->result, result_name))
00434         json_object_set_string_member (object, LABEL_RESULT_FILE,
input->result);
00435     if (strcmp (input->variables, variables_name))
00436         json_object_set_string_member (object, LABEL_VARIABLES_FILE,
input->variables);
00437
00438     file = g_file_new_for_path (input->directory);
00439     file2 = g_file_new_for_path (input->simulator);
00440     buffer = g_file_get_relative_path (file, file2);
00441     g_object_unref (file2);
00442     json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00443     g_free (buffer);
00444     if (input->evaluator)
00445     {
00446         file2 = g_file_new_for_path (input->evaluator);
00447         buffer = g_file_get_relative_path (file, file2);
00448         g_object_unref (file2);
00449         if (strlen (buffer))
00450             json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00451         g_free (buffer);
00452     }
00453     if (input->seed != DEFAULT_RANDOM_SEED)
00454         json_object_set_uint (object, LABEL_SEED,
input->seed);
00455
00456     // Setting the algorithm
00457     buffer = (char *) g_slice_alloc (64);
00458     switch (input->algorithm)
00459     {
00460     case ALGORITHM_MONTE_CARLO:
00461         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_MONTE_CARLO);
00462         snprintf (buffer, 64, "%u", input->nsimulations);
00463         json_object_set_string_member (object, LABEL_NSIMULATIONS, buffer);
00464         snprintf (buffer, 64, "%u", input->niterations);
00465         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00466         snprintf (buffer, 64, "%.3lg", input->tolerance);
00467         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00468         snprintf (buffer, 64, "%u", input->nbest);
00469         json_object_set_string_member (object, LABEL_NBEST, buffer);
00470         input_save_direction_json (node);
00471         break;
00472     case ALGORITHM_SWEEP:
00473         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_SWEEP);
00474         snprintf (buffer, 64, "%u", input->niterations);
00475         json_object_set_string_member (object, LABEL_NITERATIONS, buffer);
00476         snprintf (buffer, 64, "%.3lg", input->tolerance);
00477         json_object_set_string_member (object, LABEL_TOLERANCE, buffer);
00478         snprintf (buffer, 64, "%u", input->nbest);
00479         json_object_set_string_member (object, LABEL_NBEST, buffer);
00480         input_save_direction_json (node);
00481         break;
00482     default:
00483         json_object_set_string_member (object, LABEL_ALGORITHM,
LABEL_GENETIC);
00484         snprintf (buffer, 64, "%u", input->nsimulations);
00485         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 g_slice_free1 (64, buffer);
00498 if (input->threshold != 0.)
00499     json_object_set_float (object, LABEL_THRESHOLD,
00500 input->threshold);
00500
00501 // Setting the experimental data
00502 array = json_array_new ();
00503 for (i = 0; i < input->nexperiments; ++i)
00504 {
00505     child = json_node_new (JSON_NODE_OBJECT);
00506     object = json_node_get_object (child);
00507     json_object_set_string_member (object2, LABEL_NAME,
00508 input->experiment[i].name);
00509     if (input->experiment[i].weight != 1.)
00510         json_object_set_float (object2, LABEL_WEIGHT,
00511 input->experiment[i].weight);
00512     for (j = 0; j < input->experiment->ninputs; ++j)
00513         json_object_set_string_member (object2, template[j],
00514 input->experiment[i].
00515 template[j]);
00516     json_array_add_element (array, child);
00517 }
00518 json_object_set_array_member (object, LABEL_EXPERIMENTS, array);
00519
00520 // Setting the variables data
00521 array = json_array_new ();
00522 for (i = 0; i < input->nvariables; ++i)
00523 {
00524     child = json_node_new (JSON_NODE_OBJECT);
00525     object = json_node_get_object (child);
00526     json_object_set_string_member (object2, LABEL_NAME,
00527 input->variable[i].name);
00528     json_object_set_float (object2, LABEL_MINIMUM,
00529 input->variable[i].rangemin);
00530     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00531         json_object_set_float (object2,
00532 LABEL_ABSOLUTE_MINIMUM,
00533 input->variable[i].rangeminabs);
00534     json_object_set_float (object2, LABEL_MAXIMUM,
00535 input->variable[i].rangemax);
00536     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00537         json_object_set_float (object2,
00538 LABEL_ABSOLUTE_MAXIMUM,
00539 input->variable[i].rangemaxabs);
00540     if (input->variable[i].precision !=
00541 DEFAULT_PRECISION)
00542         json_object_set_uint (object2, LABEL_PRECISION,
00543 input->variable[i].precision);
00544     if (input->algorithm == ALGORITHM_SWEEP)
00545         json_object_set_uint (object2, LABEL_NSWEEPS,
00546 input->variable[i].nsweeps);
00547     else if (input->algorithm == ALGORITHM_GENETIC)
00548         json_object_set_uint (object2, LABEL_NBITS,
00549 input->variable[i].nbits);
00550     if (input->nsteps)
00551         json_object_set_float (object, LABEL_STEP,
00552 input->variable[i].step);
00553     json_array_add_element (array, child);
00554 }
00555 json_object_set_array_member (object, LABEL_VARIABLES, array);
00556
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 }
00572 #if DEBUG_INTERFACE

```

```

00566     fprintf (stderr, "input_save_json: end\n");
00567 #endif
00568 }

```

Here is the call graph for this function:

5.11.2.5 void input_save_xml (xmlDoc * doc)

Function to save the input file in XML format.

Parameters

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

Definition at line 244 of file [interface.c](#).

```

00245 {
00246     unsigned int i, j;
00247     char *buffer;
00248     xmlNode *node, *child;
00249     GFile *file, *file2;
00250
00251 #if DEBUG_INTERFACE
00252     fprintf (stderr, "input_save_xml: start\n");
00253 #endif
00254
00255     // Setting root XML node
00256     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00257     xmlDocSetRootElement (doc, node);
00258
00259     // Adding properties to the root XML node
00260     if (xmlStrcmp
00261         ((const xmlChar *) input->result, (const xmlChar *) result_name))
00262         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
00263                     (xmlChar *) input->result);
00264     if (xmlStrcmp
00265         ((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
00266         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
00267                     (xmlChar *) input->variables);
00268     file = g_file_new_for_path (input->directory);
00269     file2 = g_file_new_for_path (input->simulator);
00270     buffer = g_file_get_relative_path (file, file2);
00271     g_object_unref (file2);
00272     xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273     g_free (buffer);
00274     if (input->evaluator)
00275     {
00276         file2 = g_file_new_for_path (input->evaluator);
00277         buffer = g_file_get_relative_path (file, file2);
00278         g_object_unref (file2);
00279         if (xmlStrlen ((xmlChar *) buffer))
00280             xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00281                         (xmlChar *) buffer);
00282         g_free (buffer);
00283     }
00284     if (input->seed != DEFAULT_RANDOM_SEED)
00285         xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);
00286
00287     // Setting the algorithm
00288     buffer = (char *) g_slice_alloc (64);
00289     switch (input->algorithm)
00290     {
00291     case ALGORITHM_MONTE_CARLO:
00292         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                     (const xmlChar *) LABEL_MONTE_CARLO);
00294         snprintf (buffer, 64, "%u", input->nsimulations);
00295         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                     (xmlChar *) buffer);
00297         snprintf (buffer, 64, "%u", input->niterations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                     (xmlChar *) buffer);

```

```

00300     snprintf (buffer, 64, "%.3lg", input->tolerance);
00301     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302     snprintf (buffer, 64, "%u", input->nbest);
00303     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304     input_save_direction_xml (node);
00305     break;
00306 case ALGORITHM_SWEEP:
00307     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                 (const xmlChar *) LABEL_SWEEP);
00309     snprintf (buffer, 64, "%u", input->niterations);
00310     xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                 (xmlChar *) buffer);
00312     snprintf (buffer, 64, "%.3lg", input->tolerance);
00313     xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00314     snprintf (buffer, 64, "%u", input->nbest);
00315     xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00316     input_save_direction_xml (node);
00317     break;
00318 default:
00319     xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00320                 (const xmlChar *) LABEL_GENETIC);
00321     snprintf (buffer, 64, "%u", input->nsimulations);
00322     xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00323                 (xmlChar *) buffer);
00324     snprintf (buffer, 64, "%u", input->niterations);
00325     xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00326                 (xmlChar *) buffer);
00327     snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00328     xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00329     snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00330     xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00331                 (xmlChar *) buffer);
00332     snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00333     xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00334     break;
00335 }
00336 g_slice_free1 (64, buffer);
00337 if (input->threshold != 0.)
00338     xml_node_set_float (node, (const xmlChar *)
00339 LABEL_THRESHOLD,
00340                        input->threshold);
00341 // Setting the experimental data
00342 for (i = 0; i < input->nexperiments; ++i)
00343 {
00344     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00345     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00346                 (xmlChar *) input->experiment[i].name);
00347     if (input->experiment[i].weight != 1.)
00348         xml_node_set_float (child, (const xmlChar *)
00349 LABEL_WEIGHT,
00350                            input->experiment[i].weight);
00351     for (j = 0; j < input->experiment->ninputs; ++j)
00352         xmlSetProp (child, (const xmlChar *) template[j],
00353                     (xmlChar *) input->experiment[i].template[j]);
00354 }
00355 // Setting the variables data
00356 for (i = 0; i < input->nvariables; ++i)
00357 {
00358     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00359     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00360                 (xmlChar *) input->variable[i].name);
00361     xml_node_set_float (child, (const xmlChar *)
00362 LABEL_MINIMUM,
00363                        input->variable[i].rangemin);
00364     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365         xml_node_set_float (child, (const xmlChar *)
00366 LABEL_ABSOLUTE_MINIMUM,
00367                            input->variable[i].rangeminabs);
00368     xml_node_set_float (child, (const xmlChar *)
00369 LABEL_MAXIMUM,
00370                        input->variable[i].rangemax);
00371     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00372         xml_node_set_float (child, (const xmlChar *)
00373 LABEL_ABSOLUTE_MAXIMUM,
00374                            input->variable[i].rangemaxabs);
00375     if (input->variable[i].precision !=
00376         DEFAULT_PRECISION)
00377         xml_node_set_uint (child, (const xmlChar *)
00378 LABEL_PRECISION,
00379                        input->variable[i].precision);
00380     if (input->algorithm == ALGORITHM_SWEEP)
00381         xml_node_set_uint (child, (const xmlChar *)
00382 LABEL_NSWEEPS,
00383                        input->variable[i].nsweeps);
00384     else if (input->algorithm == ALGORITHM_GENETIC)

```

```

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

```

Here is the call graph for this function:

5.11.2.6 unsigned int window_get_algorithm ()

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 728 of file [interface.c](#).

```

00729 {
00730     unsigned int i;
00731     #if DEBUG_INTERFACE
00732         fprintf (stderr, "window_get_algorithm: start\n");
00733     #endif
00734     i = gtk_array_get_active (window->button_algorithm,
NALGORITHMS);
00735     #if DEBUG_INTERFACE
00736         fprintf (stderr, "window_get_algorithm: %u\n", i);
00737         fprintf (stderr, "window_get_algorithm: end\n");
00738     #endif
00739     return i;
00740 }

```

Here is the call graph for this function:

5.11.2.7 unsigned int window_get_direction ()

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 748 of file [interface.c](#).

```

00749 {
00750     unsigned int i;
00751     #if DEBUG_INTERFACE
00752     fprintf (stderr, "window_get_direction: start\n");
00753     #endif
00754     i = gtk_array_get_active (window->button_direction,
00755                             NDIRECTIONS);
00756     #if DEBUG_INTERFACE
00757     fprintf (stderr, "window_get_direction: %u\n", i);
00758     fprintf (stderr, "window_get_direction: end\n");
00759     #endif
00760     return i;
00761 }
```

Here is the call graph for this function:

5.11.2.8 unsigned int window_get_norm ()

Function to get the norm method number.

Returns

Norm method number.

Definition at line 768 of file [interface.c](#).

```

00769 {
00770     unsigned int i;
00771     #if DEBUG_INTERFACE
00772     fprintf (stderr, "window_get_norm: start\n");
00773     #endif
00774     i = gtk_array_get_active (window->button_norm,
00775                             NNORMS);
00776     #if DEBUG_INTERFACE
00777     fprintf (stderr, "window_get_norm: %u\n", i);
00778     fprintf (stderr, "window_get_norm: end\n");
00779     #endif
00780     return i;
00781 }
```

Here is the call graph for this function:

5.11.2.9 int window_read (char * filename)

Function to read the input data of a file.

Parameters

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

Returns

1 on succes, 0 on error.

Definition at line 1876 of file [interface.c](#).

```

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

```

```

01947     gtk_spin_button_set_value (window->spin_reproduction,
01948                               input->reproduction_ratio);
01949     gtk_spin_button_set_value (window->spin_adaptation,
01950                               input->adaptation_ratio);
01951 }
01952 gtk_toggle_button_set_active
01953 (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01954 gtk_spin_button_set_value (window->spin_p, input->p);
01955 gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01956 g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01957 g_signal_handler_block (window->button_experiment,
                        window->id_experiment_name);
01958
01959 gtk_combo_box_text_remove_all (window->combo_experiment);
01960 for (i = 0; i < input->nexperiments; ++i)
01961     gtk_combo_box_text_append_text (window->combo_experiment,
                                input->experiment[i].name);
01962
01963 g_signal_handler_unblock
01964 (window->button_experiment, window->
id_experiment_name);
01965 g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01966 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967 g_signal_handler_block (window->combo_variable, window->
id_variable);
01968 g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01969 gtk_combo_box_text_remove_all (window->combo_variable);
01970 for (i = 0; i < input->nvariables; ++i)
01971     gtk_combo_box_text_append_text (window->combo_variable,
                                input->variable[i].name);
01972
01973 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01974 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01975 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976 window_set_variable ();
01977 window_update ();
01978
01979 #if DEBUG_INTERFACE
01980 fprintf (stderr, "window_read: end\n");
01981 #endif
01982 return 1;
01983 }

```

Here is the call graph for this function:

5.11.2.10 int window_save ()

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 821 of file [interface.c](#).

```

00822 {
00823     GtkFileChooserDialog *dlg;
00824     GtkFileFilter *filter1, *filter2;
00825     char *buffer;
00826
00827 #if DEBUG_INTERFACE
00828     fprintf (stderr, "window_save: start\n");
00829 #endif
00830
00831     // Opening the saving dialog
00832     dlg = (GtkFileChooserDialog *)
00833         gtk_file_chooser_dialog_new (gettext ("Save file"),
00834                                     window->window,
00835                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00836                                     gettext ("_Cancel"),
00837                                     GTK_RESPONSE_CANCEL,
00838                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);

```

```

00839 gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00840 buffer = g_build_filename (input->directory, input->name, NULL);
00841 gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00842 g_free (buffer);
00843
00844 // Adding XML filter
00845 filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00846 gtk_file_filter_set_name (filter1, "XML");
00847 gtk_file_filter_add_pattern (filter1, "*.xml");
00848 gtk_file_filter_add_pattern (filter1, "*.XML");
00849 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00850
00851 // Adding JSON filter
00852 filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00853 gtk_file_filter_set_name (filter2, "JSON");
00854 gtk_file_filter_add_pattern (filter2, "*.json");
00855 gtk_file_filter_add_pattern (filter2, "*.JSON");
00856 gtk_file_filter_add_pattern (filter2, "*.js");
00857 gtk_file_filter_add_pattern (filter2, "*.JS");
00858 gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00859
00860 if (input->type == INPUT_TYPE_XML)
00861     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00862 else
00863     gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865 // If OK response then saving
00866 if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00867 {
00868     // Setting input file type
00869     filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00870     buffer = (char *) gtk_file_filter_get_name (filter1);
00871     if (!strcmp (buffer, "XML"))
00872         input->type = INPUT_TYPE_XML;
00873     else
00874         input->type = INPUT_TYPE_JSON;
00875
00876     // Adding properties to the root XML node
00877     input->simulator = gtk_file_chooser_get_filename
00878         (GTK_FILE_CHOOSER (window->button_simulator));
00879     if (gtk_toggle_button_get_active
00880         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00881         input->evaluator = gtk_file_chooser_get_filename
00882             (GTK_FILE_CHOOSER (window->button_evaluator));
00883     else
00884         input->evaluator = NULL;
00885     if (input->type == INPUT_TYPE_XML)
00886     {
00887         input->result
00888             = (char *) xmlStrdup ((const xmlChar *)
00889                 gtk_entry_get_text (window->entry_result));
00890         input->variables
00891             = (char *) xmlStrdup ((const xmlChar *)
00892                 gtk_entry_get_text (window->
00893                     entry_variables));
00894     }
00895     else
00896     {
00897         input->result = g_strdup (gtk_entry_get_text (window->
00898             entry_result));
00899         input->variables
00900             = g_strdup (gtk_entry_get_text (window->entry_variables));
00901     }
00902     // Setting the algorithm
00903     switch (window_get_algorithm ())
00904     {
00905     case ALGORITHM_MONTE_CARLO:
00906         input->algorithm = ALGORITHM_MONTE_CARLO;
00907         input->nsimulations
00908             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00909         input->niterations
00910             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00911         input->tolerance = gtk_spin_button_get_value (window->
00912             spin_tolerance);
00913         input->nbest = gtk_spin_button_get_value_as_int (window->
00914             spin_bests);
00915         window_save_direction ();
00916         break;
00917     case ALGORITHM_SWEEP:
00918         input->algorithm = ALGORITHM_SWEEP;
00919         input->niterations
00920             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921         input->tolerance = gtk_spin_button_get_value (window->
00922             spin_tolerance);
00923         input->nbest = gtk_spin_button_get_value_as_int (window->
00924             spin_bests);

```

```

00920         window_save_direction ();
00921         break;
00922     default:
00923         input->algorithm = ALGORITHM_GENETIC;
00924         input->nsimulations
00925             = gtk_spin_button_get_value_as_int (window->spin_population);
00926         input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_generations);
00928         input->mutation_ratio
00929             = gtk_spin_button_get_value (window->spin_mutation);
00930         input->reproduction_ratio
00931             = gtk_spin_button_get_value (window->spin_reproduction);
00932         input->adaptation_ratio
00933             = gtk_spin_button_get_value (window->spin_adaptation);
00934         break;
00935     }
00936     input->norm = window_get_norm ();
00937     input->p = gtk_spin_button_get_value (window->spin_p);
00938     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00939
00940     // Saving the XML file
00941     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00942     input_save (buffer);
00943
00944     // Closing and freeing memory
00945     g_free (buffer);
00946     gtk_widget_destroy (GTK_WIDGET (dlg));
00947 #if DEBUG_INTERFACE
00948     fprintf (stderr, "window_save: end\n");
00949 #endif
00950     return 1;
00951 }
00952
00953 // Closing and freeing memory
00954 gtk_widget_destroy (GTK_WIDGET (dlg));
00955 #if DEBUG_INTERFACE
00956     fprintf (stderr, "window_save: end\n");
00957 #endif
00958     return 0;
00959 }

```

Here is the call graph for this function:

5.11.2.11 void window_template_experiment (void * data)

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

Parameters

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

Definition at line 1520 of file [interface.c](#).

```

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

```

```

01540     g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542     fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }

```

5.12 interface.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
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 #elif !defined (__BSD_VISIBLE)
00047 #include <alloca.h>
00048 #endif
00049 #if HAVE_MPI
00050 #include <mpi.h>
00051 #endif
00052 #include <gio/gio.h>
00053 #include <gtk/gtk.h>
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 #include "interface.h"
00061
00062 #define DEBUG_INTERFACE 0
00063
00064 #ifdef G_OS_WIN32
00065 #define INPUT_FILE "test-ga-win.xml"
00066 #else
00067 #define INPUT_FILE "test-ga.xml"
00068 #endif
00069
00070 const char *logo[] = {
00071     "32 32 3 1",
00072     "    c None",
00073     ".    c #0000FF",

```

```

00084 " + c #FF0000",
00085 " ",
00086 " ",
00087 " ",
00088 " . . . . ",
00089 " . . . . ",
00090 " . . . . ",
00091 " . . . . ",
00092 " . . . . +++ ",
00093 " . . . . +++++ ",
00094 " . . . . +++++ ",
00095 " . . . . +++++ ",
00096 " +++ . . . . +++ ",
00097 " +++++ . . . . +++++ ",
00098 " +++++ . . . . +++++ ",
00099 " +++++ . . . . +++++ ",
00100 " +++ . . . . +++ ",
00101 " . . . . . ",
00102 " . . . . . ",
00103 " . . . . . ",
00104 " . . . . . ",
00105 " . . . . . ",
00106 " . . . . . ",
00107 " . . . . . ",
00108 " . . . . . ",
00109 " . . . . . ",
00110 " . . . . . ",
00111 " . . . . . ",
00112 " . . . . . ",
00113 " . . . . . ",
00114 " . . . . . ",
00115 " . . . . . ",
00116 " . . . . . ",
00117 };
00118
00119 /*
00120 const char * logo[] = {
00121 "32 32 3 1",
00122 " c #FFFFFFFFFFFF",
00123 ". c #00000000FFFF",
00124 "X c #FFFF00000000",
00125 " ",
00126 " ",
00127 " ",
00128 " . . . . ",
00129 " . . . . ",
00130 " . . . . ",
00131 " . . . . ",
00132 " . . . . XXX ",
00133 " . . . . XXXXX ",
00134 " . . . . XXXXX ",
00135 " . . . . XXXXX ",
00136 " XXX . . . . XXX ",
00137 " XXXXX . . . . XXXXX ",
00138 " XXXXX . . . . XXXXX ",
00139 " XXXXX . . . . XXXXX ",
00140 " XXX . . . . XXX ",
00141 " . . . . . ",
00142 " . . . . . ",
00143 " . . . . . ",
00144 " . . . . . ",
00145 " . . . . . ",
00146 " . . . . . ",
00147 " . . . . . ",
00148 " . . . . . ",
00149 " . . . . . ",
00150 " . . . . . ",
00151 " . . . . . ",
00152 " . . . . . ",
00153 " . . . . . ",
00154 " . . . . . ",
00155 " . . . . . ",
00156 " . . . . . ";
00157 */
00158
00159 Options options[1];
00161 Running running[1];
00163 Window window[1];
00165
00172 void
00173 input_save_direction_xml (xmlNode * node)
00174 {
00175     #if DEBUG_INTERFACE
00176     fprintf (stderr, "input_save_direction_xml: start\n");
00177     #endif
00178     if (input->nsteps)
00179     {

```

```

00180     xml_node_set_uint (node, (const xmlChar *) LABEL_NSTEPS,
input->nsteps);
00181     if (input->relaxation != DEFAULT_RELAXATION)
00182         xml_node_set_float (node, (const xmlChar *)
LABEL_RELAXATION,
                                input->relaxation);
00183     switch (input->direction)
00184     {
00185     case DIRECTION_METHOD_COORDINATES:
00186         xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
                                (const xmlChar *) LABEL_COORDINATES);
00187         break;
00188     default:
00189         xmlSetProp (node, (const xmlChar *) LABEL_DIRECTION,
                                (const xmlChar *) LABEL_RANDOM);
00190         xml_node_set_uint (node, (const xmlChar *)
LABEL_NESTIMATES,
                                input->nestimates);
00191     }
00192 }
00193 #if DEBUG_INTERFACE
00194 fprintf (stderr, "input_save_direction_xml: end\n");
00195 #endif
00196 }
00197 void
00198 input_save_direction_json (JsonNode * node)
00199 {
00200     JsonObject *object;
00201     #if DEBUG_INTERFACE
00202     fprintf (stderr, "input_save_direction_json: start\n");
00203     #endif
00204     object = json_node_get_object (node);
00205     if (input->nsteps)
00206     {
00207         json_object_set_uint (object, LABEL_NSTEPS,
input->nsteps);
00208         if (input->relaxation != DEFAULT_RELAXATION)
00209             json_object_set_float (object, LABEL_RELAXATION,
input->relaxation);
00210         switch (input->direction)
00211         {
00212         case DIRECTION_METHOD_COORDINATES:
00213             json_object_set_string_member (object, LABEL_DIRECTION,
LABEL_COORDINATES);
00214             break;
00215         default:
00216             json_object_set_string_member (object, LABEL_DIRECTION,
LABEL_RANDOM);
00217         }
00218         json_object_set_uint (object, LABEL_NESTIMATES,
input->nestimates);
00219     }
00220 }
00221 #if DEBUG_INTERFACE
00222 fprintf (stderr, "input_save_direction_json: end\n");
00223 #endif
00224 }
00225 void
00226 input_save_xml (xmlDoc * doc)
00227 {
00228     unsigned int i, j;
00229     char *buffer;
00230     xmlNode *node, *child;
00231     GFile *file, *file2;
00232     #if DEBUG_INTERFACE
00233     fprintf (stderr, "input_save_xml: start\n");
00234     #endif
00235     // Setting root XML node
00236     node = xmlNewDocNode (doc, 0, (const xmlChar *) LABEL_OPTIMIZE, 0);
00237     xmlDocSetRootElement (doc, node);
00238     // Adding properties to the root XML node
00239     if (xmlStrcmp
((const xmlChar *) input->result, (const xmlChar *) result_name))
00240         xmlSetProp (node, (const xmlChar *) LABEL_RESULT_FILE,
                                (xmlChar *) input->result);
00241     if (xmlStrcmp
((const xmlChar *) input->variables, (const xmlChar *)
variables_name))
00242         xmlSetProp (node, (const xmlChar *) LABEL_VARIABLES_FILE,
                                (xmlChar *) input->variables);
00243     file = g_file_new_for_path (input->directory);
00244     file2 = g_file_new_for_path (input->simulator);
00245     buffer = g_file_get_relative_path (file, file2);

```



```

00271 g_object_unref (file2);
00272 xmlSetProp (node, (const xmlChar *) LABEL_SIMULATOR, (xmlChar *) buffer);
00273 g_free (buffer);
00274 if (input->evaluator)
00275 {
00276     file2 = g_file_new_for_path (input->evaluator);
00277     buffer = g_file_get_relative_path (file, file2);
00278     g_object_unref (file2);
00279     if (xmlStrlen ((xmlChar *) buffer))
00280         xmlSetProp (node, (const xmlChar *) LABEL_EVALUATOR,
00281                     (xmlChar *) buffer);
00282     g_free (buffer);
00283 }
00284 if (input->seed != DEFAULT_RANDOM_SEED)
00285     xml_node_set_uint (node, (const xmlChar *) LABEL_SEED,
input->seed);

00286
00287 // Setting the algorithm
00288 buffer = (char *) g_slice_alloc (64);
00289 switch (input->algorithm)
00290 {
00291     case ALGORITHM_MONTE_CARLO:
00292         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00293                     (const xmlChar *) LABEL_MONTE_CARLO);
00294         snprintf (buffer, 64, "%u", input->nsimulations);
00295         xmlSetProp (node, (const xmlChar *) LABEL_NSIMULATIONS,
00296                     (xmlChar *) buffer);
00297         snprintf (buffer, 64, "%u", input->niterations);
00298         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00299                     (xmlChar *) buffer);
00300         snprintf (buffer, 64, "%.3lg", input->tolerance);
00301         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00302         snprintf (buffer, 64, "%u", input->nbest);
00303         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00304         input_save_direction_xml (node);
00305         break;
00306     case ALGORITHM_SWEEP:
00307         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00308                     (const xmlChar *) LABEL_SWEEP);
00309         snprintf (buffer, 64, "%u", input->niterations);
00310         xmlSetProp (node, (const xmlChar *) LABEL_NITERATIONS,
00311                     (xmlChar *) buffer);
00312         snprintf (buffer, 64, "%.3lg", input->tolerance);
00313         xmlSetProp (node, (const xmlChar *) LABEL_TOLERANCE, (xmlChar *) buffer);
00314         snprintf (buffer, 64, "%u", input->nbest);
00315         xmlSetProp (node, (const xmlChar *) LABEL_NBEST, (xmlChar *) buffer);
00316         input_save_direction_xml (node);
00317         break;
00318     default:
00319         xmlSetProp (node, (const xmlChar *) LABEL_ALGORITHM,
00320                     (const xmlChar *) LABEL_GENETIC);
00321         snprintf (buffer, 64, "%u", input->nsimulations);
00322         xmlSetProp (node, (const xmlChar *) LABEL_NPOPULATION,
00323                     (xmlChar *) buffer);
00324         snprintf (buffer, 64, "%u", input->niterations);
00325         xmlSetProp (node, (const xmlChar *) LABEL_NGENERATIONS,
00326                     (xmlChar *) buffer);
00327         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00328         xmlSetProp (node, (const xmlChar *) LABEL_MUTATION, (xmlChar *) buffer);
00329         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00330         xmlSetProp (node, (const xmlChar *) LABEL_REPRODUCTION,
00331                     (xmlChar *) buffer);
00332         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00333         xmlSetProp (node, (const xmlChar *) LABEL_ADAPTATION, (xmlChar *) buffer);
00334         break;
00335 }
00336 g_slice_free1 (64, buffer);
00337 if (input->threshold != 0.)
00338     xml_node_set_float (node, (const xmlChar *)
LABEL_THRESHOLD,
input->threshold);

00339
00340 // Setting the experimental data
00341 for (i = 0; i < input->nexperiments; ++i)
00342 {
00343     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_EXPERIMENT, 0);
00344     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00345                 (xmlChar *) input->experiment[i].name);
00346     if (input->experiment[i].weight != 1.)
00347         xml_node_set_float (child, (const xmlChar *)
LABEL_WEIGHT,
input->experiment[i].weight);
00348     for (j = 0; j < input->experiment->ninputs; ++j)
00349         xmlSetProp (child, (const xmlChar *) template[j],
00350                     (xmlChar *) input->experiment[i].template[j]);
00351 }
00352
00353
00354

```

```

00355 // Setting the variables data
00356 for (i = 0; i < input->nvariables; ++i)
00357 {
00358     child = xmlNewChild (node, 0, (const xmlChar *) LABEL_VARIABLE, 0);
00359     xmlSetProp (child, (const xmlChar *) LABEL_NAME,
00360                 (xmlChar *) input->variable[i].name);
00361     xml_node_set_float (child, (const xmlChar *)
00362 LABEL_MINIMUM,
00363                         input->variable[i].rangemin);
00364     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00365         xml_node_set_float (child, (const xmlChar *)
00366 LABEL_ABSOLUTE_MINIMUM,
00367                             input->variable[i].rangeminabs);
00368     xml_node_set_float (child, (const xmlChar *)
00369 LABEL_MAXIMUM,
00370                         input->variable[i].rangemax);
00371     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00372         xml_node_set_float (child, (const xmlChar *)
00373 LABEL_ABSOLUTE_MAXIMUM,
00374                             input->variable[i].rangemaxabs);
00375     if (input->variable[i].precision !=
00376         DEFAULT_PRECISION)
00377         xml_node_set_uint (child, (const xmlChar *)
00378 LABEL_PRECISION,
00379                             input->variable[i].precision);
00380     if (input->algorithm == ALGORITHM_SWEEP)
00381         xml_node_set_uint (child, (const xmlChar *)
00382 LABEL_NSWEEPS,
00383                             input->variable[i].nsweeps);
00384     else if (input->algorithm == ALGORITHM_GENETIC)
00385         xml_node_set_uint (child, (const xmlChar *) LABEL_NBITS,
00386                             input->variable[i].nbits);
00387     if (input->nsteps)
00388         xml_node_set_float (child, (const xmlChar *)
00389 LABEL_STEP,
00390                             input->variable[i].step);
00391 }
00392 // Saving the error norm
00393 switch (input->norm)
00394 {
00395     case ERROR_NORM_MAXIMUM:
00396         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00397                     (const xmlChar *) LABEL_MAXIMUM);
00398         break;
00399     case ERROR_NORM_P:
00400         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00401                     (const xmlChar *) LABEL_P);
00402         xml_node_set_float (node, (const xmlChar *) LABEL_P,
00403 input->p);
00404         break;
00405     case ERROR_NORM_TAXICAB:
00406         xmlSetProp (node, (const xmlChar *) LABEL_NORM,
00407                     (const xmlChar *) LABEL_TAXICAB);
00408 }
00409 #if DEBUG_INTERFACE
00410 fprintf (stderr, "input_save: end\n");
00411 #endif
00412 }
00413 void
00414 input_save_json (JsonGenerator * generator)
00415 {
00416     unsigned int i, j;
00417     char *buffer;
00418     JsonNode *node, *child;
00419     JsonObject *object, *object2;
00420     JsonArray *array;
00421     GFile *file, *file2;
00422
00423 #if DEBUG_INTERFACE
00424 fprintf (stderr, "input_save_json: start\n");
00425 #endif
00426
00427 // Setting root JSON node
00428 node = json_node_new (JSON_NODE_OBJECT);
00429 object = json_node_get_object (node);
00430 json_generator_set_root (generator, node);
00431
00432 // Adding properties to the root JSON node
00433 if (strcmp (input->result, result_name))
00434     json_object_set_string_member (object, LABEL_RESULT_FILE,
00435 input->result);
00436 if (strcmp (input->variables, variables_name))
00437     json_object_set_string_member (object, LABEL_VARIABLES_FILE,
00438 input->variables);

```

```

00438 file = g_file_new_for_path (input->directory);
00439 file2 = g_file_new_for_path (input->simulator);
00440 buffer = g_file_get_relative_path (file, file2);
00441 g_object_unref (file2);
00442 json_object_set_string_member (object, LABEL_SIMULATOR, buffer);
00443 g_free (buffer);
00444 if (input->evaluator)
00445 {
00446     file2 = g_file_new_for_path (input->evaluator);
00447     buffer = g_file_get_relative_path (file, file2);
00448     g_object_unref (file2);
00449     if (strlen (buffer))
00450         json_object_set_string_member (object, LABEL_EVALUATOR, buffer);
00451     g_free (buffer);
00452 }
00453 if (input->seed != DEFAULT_RANDOM_SEED)
00454     json_object_set_uint (object, LABEL_SEED,
input->seed);
00455
00456 // Setting the algorithm
00457 buffer = (char *) g_slice_alloc (64);
00458 switch (input->algorithm)
00459 {
00460     case ALGORITHM_MONTE_CARLO:
00461         json_object_set_string_member (object, LABEL_ALGORITHM,
00462                                         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,
00475                                         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,
00486                                         LABEL_GENETIC);
00487         snprintf (buffer, 64, "%u", input->nsimulations);
00488         json_object_set_string_member (object, LABEL_NPOPULATION, buffer);
00489         snprintf (buffer, 64, "%u", input->niterations);
00490         json_object_set_string_member (object, LABEL_NGENERATIONS, buffer);
00491         snprintf (buffer, 64, "%.3lg", input->mutation_ratio);
00492         json_object_set_string_member (object, LABEL_MUTATION, buffer);
00493         snprintf (buffer, 64, "%.3lg", input->reproduction_ratio);
00494         json_object_set_string_member (object, LABEL_REPRODUCTION, buffer);
00495         snprintf (buffer, 64, "%.3lg", input->adaptation_ratio);
00496         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,
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,
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
00522 // Setting the variables data

```

```

00520     array = json_array_new ();
00521     for (i = 0; i < input->nvariables; ++i)
00522     {
00523         child = json_node_new (JSON_NODE_OBJECT);
00524         object = json_node_get_object (child);
00525         json_object_set_string_member (object2, LABEL_NAME,
00526                                     input->variable[i].name);
00527         json_object_set_float (object2, LABEL_MINIMUM,
00528                               input->variable[i].rangemin);
00529         if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
00530             json_object_set_float (object2,
00531                                   LABEL_ABSOLUTE_MINIMUM,
00532                                   input->variable[i].rangeminabs);
00533         json_object_set_float (object2, LABEL_MAXIMUM,
00534                               input->variable[i].rangemax);
00535         if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
00536             json_object_set_float (object2,
00537                                   LABEL_ABSOLUTE_MAXIMUM,
00538                                   input->variable[i].rangemaxabs);
00539         if (input->variable[i].precision !=
00540             DEFAULT_PRECISION)
00541             json_object_set_uint (object2, LABEL_PRECISION,
00542                                  input->variable[i].precision);
00543         if (input->algorithm == ALGORITHM_SWEEP)
00544             json_object_set_uint (object2, LABEL_NSWEEPS,
00545                                  input->variable[i].nsweeps);
00546         else if (input->algorithm == ALGORITHM_GENETIC)
00547             json_object_set_uint (object2, LABEL_NBITS,
00548                                  input->variable[i].nbits);
00549         if (input->nsteps)
00550             json_object_set_float (object, LABEL_STEP,
00551                                   input->variable[i].step);
00552         json_array_add_element (array, child);
00553     }
00554     json_object_set_array_member (object, LABEL_VARIABLES, array);
00555
00556     // Saving the error norm
00557     switch (input->norm)
00558     {
00559     case ERROR_NORM_MAXIMUM:
00560         json_object_set_string_member (object, LABEL_NORM, LABEL_MAXIMUM);
00561         break;
00562     case ERROR_NORM_P:
00563         json_object_set_string_member (object, LABEL_NORM, LABEL_P);
00564         json_object_set_float (object, LABEL_P, input->
00565                                p);
00566         break;
00567     case ERROR_NORM_TAXICAB:
00568         json_object_set_string_member (object, LABEL_NORM, LABEL_TAXICAB);
00569     }
00570
00571 #if DEBUG_INTERFACE
00572     fprintf (stderr, "input_save_json: end\n");
00573 #endif
00574 }
00575
00576 void
00577 input_save (char *filename)
00578 {
00579     xmlDoc *doc;
00580     JsonGenerator *generator;
00581
00582 #if DEBUG_INTERFACE
00583     fprintf (stderr, "input_save: start\n");
00584 #endif
00585
00586     // Getting the input file directory
00587     input->name = g_path_get_basename (filename);
00588     input->directory = g_path_get_dirname (filename);
00589
00590     if (input->type == INPUT_TYPE_XML)
00591     {
00592         // Opening the input file
00593         doc = xmlNewDoc ((const xmlChar *) "1.0");
00594         input_save_xml (doc);
00595
00596         // Saving the XML file
00597         xmlSaveFormatFile (filename, doc, 1);
00598
00599         // Freeing memory
00600         xmlFreeDoc (doc);
00601     }
00602     else
00603     {
00604         // Opening the input file
00605         generator = json_generator_new ();
00606         json_generator_set_pretty (generator, TRUE);

```

```

00607     input_save_json (generator);
00608
00609     // Saving the JSON file
00610     json_generator_to_file (generator, filename, NULL);
00611
00612     // Freeing memory
00613     g_object_unref (generator);
00614 }
00615
00616 #if DEBUG_INTERFACE
00617 fprintf (stderr, "input_save: end\n");
00618 #endif
00619 }
00620
00621 void
00622 options_new ()
00623 {
00624     #if DEBUG_INTERFACE
00625     fprintf (stderr, "options_new: start\n");
00626     #endif
00627     options->label_seed = (GtkLabel *)
00628     gtk_label_new (gettext ("Pseudo-random numbers generator seed"));
00629     options->spin_seed = (GtkSpinButton *)
00630     gtk_spin_button_new_with_range (0., (gdouble) G_MAXULONG, 1.);
00631     gtk_widget_set_tooltip_text
00632     (GTK_WIDGET (options->spin_seed),
00633     gettext ("Seed to init the pseudo-random numbers generator"));
00634     gtk_spin_button_set_value (options->spin_seed, (gdouble) input->
00635     seed);
00636     options->label_threads = (GtkLabel *)
00637     gtk_label_new (gettext ("Threads number for the stochastic algorithm"));
00638     options->spin_threads
00639     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00640     gtk_widget_set_tooltip_text
00641     (GTK_WIDGET (options->spin_threads),
00642     gettext ("Number of threads to perform the calibration/optimization for "
00643     "the stochastic algorithm"));
00644     gtk_spin_button_set_value (options->spin_threads, (gdouble)
00645     nthreads);
00646     options->label_direction = (GtkLabel *)
00647     gtk_label_new (gettext ("Threads number for the direction search method"));
00648     options->spin_direction
00649     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
00650     gtk_widget_set_tooltip_text
00651     (GTK_WIDGET (options->spin_direction),
00652     gettext ("Number of threads to perform the calibration/optimization for "
00653     "the direction search method"));
00654     gtk_spin_button_set_value (options->spin_direction,
00655     (gdouble) nthreads_direction);
00656     options->grid = (GtkGrid *) gtk_grid_new ();
00657     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_seed), 0, 0, 1, 1);
00658     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_seed), 1, 0, 1, 1);
00659     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_threads),
00660     0, 1, 1, 1);
00661     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_threads),
00662     1, 1, 1, 1);
00663     gtk_grid_attach (options->grid, GTK_WIDGET (options->label_direction),
00664     0, 2, 1, 1);
00665     gtk_grid_attach (options->grid, GTK_WIDGET (options->spin_direction),
00666     1, 2, 1, 1);
00667     gtk_widget_show_all (GTK_WIDGET (options->grid));
00668     options->dialog = (GtkDialog *)
00669     gtk_dialog_new_with_buttons (gettext ("Options"),
00670     window->window,
00671     GTK_DIALOG_MODAL,
00672     gettext ("OK"), GTK_RESPONSE_OK,
00673     gettext ("Cancel"), GTK_RESPONSE_CANCEL,
00674     NULL);
00675     gtk_container_add
00676     (GTK_CONTAINER (gtk_dialog_get_content_area (options->dialog)),
00677     GTK_WIDGET (options->grid));
00678     if (gtk_dialog_run (options->dialog) == GTK_RESPONSE_OK)
00679     {
00680         input->seed
00681         = (unsigned long int) gtk_spin_button_get_value (options->spin_seed);
00682         nthreads = gtk_spin_button_get_value_as_int (options->spin_threads);
00683         nthreads_direction
00684         = gtk_spin_button_get_value_as_int (options->spin_direction);
00685     }
00686     gtk_widget_destroy (GTK_WIDGET (options->dialog));
00687     #if DEBUG_INTERFACE
00688     fprintf (stderr, "options_new: end\n");
00689     #endif
00690 }
00691
00692 void
00693 running_new ()

```

```

00700 {
00701 #if DEBUG_INTERFACE
00702     fprintf (stderr, "running_new: start\n");
00703 #endif
00704     running->label = (GtkLabel *) gtk_label_new (gettext ("Calculating ..."));
00705     running->spinner = (GtkSpinner *) gtk_spinner_new ();
00706     running->grid = (GtkGrid *) gtk_grid_new ();
00707     gtk_grid_attach (running->grid, GTK_WIDGET (running->label), 0, 0, 1, 1);
00708     gtk_grid_attach (running->grid, GTK_WIDGET (running->spinner), 0, 1, 1, 1);
00709     running->dialog = (GtkDialog *)
00710         gtk_dialog_new_with_buttons (gettext ("Calculating"),
00711                                     window->window, GTK_DIALOG_MODAL, NULL, NULL);
00712     gtk_container_add
00713         (GTK_CONTAINER (gtk_dialog_get_content_area (running->dialog)),
00714          GTK_WIDGET (running->grid));
00715     gtk_spinner_start (running->spinner);
00716     gtk_widget_show_all (GTK_WIDGET (running->dialog));
00717 #if DEBUG_INTERFACE
00718     fprintf (stderr, "running_new: end\n");
00719 #endif
00720 }
00721
00722 unsigned int
00723 window_get_algorithm ()
00724 {
00725     unsigned int i;
00726 #if DEBUG_INTERFACE
00727     fprintf (stderr, "window_get_algorithm: start\n");
00728 #endif
00729     i = gtk_array_get_active (window->button_algorithm,
00730                             NALGORITHMS);
00731 #if DEBUG_INTERFACE
00732     fprintf (stderr, "window_get_algorithm: %u\n", i);
00733     fprintf (stderr, "window_get_algorithm: end\n");
00734 #endif
00735     return i;
00736 }
00737
00738 unsigned int
00739 window_get_direction ()
00740 {
00741     unsigned int i;
00742 #if DEBUG_INTERFACE
00743     fprintf (stderr, "window_get_direction: start\n");
00744 #endif
00745     i = gtk_array_get_active (window->button_direction,
00746                             NDIRECTIONS);
00747 #if DEBUG_INTERFACE
00748     fprintf (stderr, "window_get_direction: %u\n", i);
00749     fprintf (stderr, "window_get_direction: end\n");
00750 #endif
00751     return i;
00752 }
00753
00754 unsigned int
00755 window_get_norm ()
00756 {
00757     unsigned int i;
00758 #if DEBUG_INTERFACE
00759     fprintf (stderr, "window_get_norm: start\n");
00760 #endif
00761     i = gtk_array_get_active (window->button_norm,
00762                             NNORMS);
00763 #if DEBUG_INTERFACE
00764     fprintf (stderr, "window_get_norm: %u\n", i);
00765     fprintf (stderr, "window_get_norm: end\n");
00766 #endif
00767     return i;
00768 }
00769
00770 void
00771 window_save_direction ()
00772 {
00773 #if DEBUG_INTERFACE
00774     fprintf (stderr, "window_save_direction: start\n");
00775 #endif
00776     if (gtk_toggle_button_get_active
00777         (GTK_TOGGLE_BUTTON (window->check_direction)))
00778     {
00779         input->nsteps = gtk_spin_button_get_value_as_int (window->
00780 spin_steps);
00781         input->relaxation = gtk_spin_button_get_value (window->
00782 spin_relaxation);
00783         switch (window_get_direction ())
00784         {
00785             case DIRECTION_METHOD_COORDINATES:
00786                 input->direction = DIRECTION_METHOD_COORDINATES;
00787         }
00788     }
00789 }

```

```

00801         break;
00802     default:
00803         input->direction = DIRECTION_METHOD_RANDOM;
00804         input->nestimates
00805             = gtk_spin_button_get_value_as_int (window->spin_estimates);
00806     }
00807 }
00808 else
00809     input->nsteps = 0;
00810 #if DEBUG_INTERFACE
00811     fprintf (stderr, "window_save_direction: end\n");
00812 #endif
00813 }
00814
00820 int
00821 window_save ()
00822 {
00823     GtkFileChooserDialog *dlg;
00824     GtkFileFilter *filter1, *filter2;
00825     char *buffer;
00826
00827 #if DEBUG_INTERFACE
00828     fprintf (stderr, "window_save: start\n");
00829 #endif
00830
00831     // Opening the saving dialog
00832     dlg = (GtkFileChooserDialog *)
00833         gtk_file_chooser_dialog_new (gettext ("Save file"),
00834                                     window->window,
00835                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00836                                     gettext ("_Cancel"),
00837                                     GTK_RESPONSE_CANCEL,
00838                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00839     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00840     buffer = g_build_filename (input->directory, input->name, NULL);
00841     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00842     g_free (buffer);
00843
00844     // Adding XML filter
00845     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00846     gtk_file_filter_set_name (filter1, "XML");
00847     gtk_file_filter_add_pattern (filter1, "*.xml");
00848     gtk_file_filter_add_pattern (filter1, "*.XML");
00849     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00850
00851     // Adding JSON filter
00852     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00853     gtk_file_filter_set_name (filter2, "JSON");
00854     gtk_file_filter_add_pattern (filter2, "*.json");
00855     gtk_file_filter_add_pattern (filter2, "*.JSON");
00856     gtk_file_filter_add_pattern (filter2, "*.js");
00857     gtk_file_filter_add_pattern (filter2, "*.JS");
00858     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00859
00860     if (input->type == INPUT_TYPE_XML)
00861         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00862     else
00863         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865     // If OK response then saving
00866     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00867     {
00868         // Setting input file type
00869         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));
00870         buffer = (char *) gtk_file_filter_get_name (filter1);
00871         if (!strcmp (buffer, "XML"))
00872             input->type = INPUT_TYPE_XML;
00873         else
00874             input->type = INPUT_TYPE_JSON;
00875
00876         // Adding properties to the root XML node
00877         input->simulator = gtk_file_chooser_get_filename
00878             (GTK_FILE_CHOOSER (window->button_simulator));
00879         if (gtk_toggle_button_get_active
00880             (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00881             input->evaluator = gtk_file_chooser_get_filename
00882                 (GTK_FILE_CHOOSER (window->button_evaluator));
00883         else
00884             input->evaluator = NULL;
00885         if (input->type == INPUT_TYPE_XML)
00886         {
00887             input->result
00888                 = (char *) xmlStrdup ((const xmlChar *)
00889                                     gtk_entry_get_text (window->entry_result));
00890             input->variables
00891                 = (char *) xmlStrdup ((const xmlChar *)
00892                                     gtk_entry_get_text (window->entry_variables));

```

```

00893     }
00894     else
00895     {
00896         input->result = g_strdup (gtk_entry_get_text (window->
entry_result));
00897         input->variables
00898         = g_strdup (gtk_entry_get_text (window->entry_variables));
00899     }
00900
00901     // Setting the algorithm
00902     switch (window_get_algorithm ())
00903     {
00904         case ALGORITHM_MONTE_CARLO:
00905             input->algorithm = ALGORITHM_MONTE_CARLO;
00906             input->nsimulations
00907             = gtk_spin_button_get_value_as_int (window->spin_simulations);
00908             input->niterations
00909             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00910             input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00911             input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00912             window_save_direction ();
00913             break;
00914         case ALGORITHM_SWEEP:
00915             input->algorithm = ALGORITHM_SWEEP;
00916             input->niterations
00917             = gtk_spin_button_get_value_as_int (window->spin_iterations);
00918             input->tolerance = gtk_spin_button_get_value (window->
spin_tolerance);
00919             input->nbest = gtk_spin_button_get_value_as_int (window->
spin_bests);
00920             window_save_direction ();
00921             break;
00922         default:
00923             input->algorithm = ALGORITHM_GENETIC;
00924             input->nsimulations
00925             = gtk_spin_button_get_value_as_int (window->spin_population);
00926             input->niterations
00927             = gtk_spin_button_get_value_as_int (window->spin_generations);
00928             input->mutation_ratio
00929             = gtk_spin_button_get_value (window->spin_mutation);
00930             input->reproduction_ratio
00931             = gtk_spin_button_get_value (window->spin_reproduction);
00932             input->adaptation_ratio
00933             = gtk_spin_button_get_value (window->spin_adaptation);
00934             break;
00935     }
00936     input->norm = window_get_norm ();
00937     input->p = gtk_spin_button_get_value (window->spin_p);
00938     input->threshold = gtk_spin_button_get_value (window->
spin_threshold);
00939
00940     // Saving the XML file
00941     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00942     input_save (buffer);
00943
00944     // Closing and freeing memory
00945     g_free (buffer);
00946     gtk_widget_destroy (GTK_WIDGET (dlg));
00947 #if DEBUG_INTERFACE
00948     fprintf (stderr, "window_save: end\n");
00949 #endif
00950     return 1;
00951 }
00952
00953 // Closing and freeing memory
00954 gtk_widget_destroy (GTK_WIDGET (dlg));
00955 #if DEBUG_INTERFACE
00956 fprintf (stderr, "window_save: end\n");
00957 #endif
00958 return 0;
00959 }
00960
00961 void
00962 window_run ()
00963 {
00964     unsigned int i;
00965     char *msg, *msg2, buffer[64], buffer2[64];
00966 #if DEBUG_INTERFACE
00967     fprintf (stderr, "window_run: start\n");
00968 #endif
00969     if (!window_save ())
00970     {
00971         #if DEBUG_INTERFACE
00972         fprintf (stderr, "window_run: end\n");
00973         #endif
00974     }
00975 }

```



```

00978     return;
00979 }
00980 running_new ();
00981 while (gtk_events_pending ())
00982     gtk_main_iteration ();
00983 optimize_open ();
00984 #if DEBUG_INTERFACE
00985 fprintf (stderr, "window_run: closing running dialog\n");
00986 #endif
00987 gtk_spinner_stop (running->spinner);
00988 gtk_widget_destroy (GTK_WIDGET (running->dialog));
00989 #if DEBUG_INTERFACE
00990 fprintf (stderr, "window_run: displaying results\n");
00991 #endif
00992 snprintf (buffer, 64, "error = %.15le\n", optimize->error_old[0]);
00993 msg2 = g_strdup (buffer);
00994 for (i = 0; i < optimize->nvariables; ++i, msg2 = msg)
00995 {
00996     snprintf (buffer, 64, "%s = %s\n",
00997             input->variable[i].name, format[input->
00998             variable[i].precision]);
00999     snprintf (buffer2, 64, buffer, optimize->value_old[i]);
01000     msg = g_strconcat (msg2, buffer2, NULL);
01001     g_free (msg2);
01002 }
01002 snprintf (buffer, 64, "%s = %.6lg s", gettext ("Calculation time"),
01003         optimize->calculation_time);
01004 msg = g_strconcat (msg2, buffer, NULL);
01005 g_free (msg2);
01006 show_message (gettext ("Best result"), msg, INFO_TYPE);
01007 g_free (msg);
01008 #if DEBUG_INTERFACE
01009 fprintf (stderr, "window_run: freeing memory\n");
01010 #endif
01011 optimize_free ();
01012 #if DEBUG_INTERFACE
01013 fprintf (stderr, "window_run: end\n");
01014 #endif
01015 }
01016
01021 void
01022 window_help ()
01023 {
01024     char *buffer, *buffer2;
01025     #if DEBUG_INTERFACE
01026     fprintf (stderr, "window_help: start\n");
01027     #endif
01028     buffer2 = g_build_filename (window->application_directory, "..", "manuals",
01029             gettext ("user-manual.pdf"), NULL);
01030     buffer = g_filename_to_uri (buffer2, NULL, NULL);
01031     g_free (buffer2);
01032     gtk_show_uri (NULL, buffer, GDK_CURRENT_TIME, NULL);
01033     #if DEBUG_INTERFACE
01034     fprintf (stderr, "window_help: uri=%s\n", buffer);
01035     #endif
01036     g_free (buffer);
01037     #if DEBUG_INTERFACE
01038     fprintf (stderr, "window_help: end\n");
01039     #endif
01040 }
01041
01046 void
01047 window_about ()
01048 {
01049     static const gchar *authors[] = {
01050         "Javier Burguete Tolosa <jburguete@eead.csic.es>",
01051         "Borja Latorre Garcés <borja.latorre@csic.es>",
01052         NULL
01053     };
01054     #if DEBUG_INTERFACE
01055     fprintf (stderr, "window_about: start\n");
01056     #endif
01057     gtk_show_about_dialog
01058     (window->window,
01059      "program_name", "MPCOTool",
01060      "comments",
01061      gettext ("The Multi-Purposes Calibration and Optimization Tool.\n"
01062              "A software to perform calibrations or optimizations of "
01063              "empirical parameters"),
01064      "authors", authors,
01065      "translator-credits",
01066      "Javier Burguete Tolosa <jburguete@eead.csic.es> "
01067      "(english, french and spanish)\n"
01068      "Uğur Çayoğlu (german)",
01069      "version", "3.0.3",
01070      "copyright", "Copyright 2012-2016 Javier Burguete Tolosa",
01071      "logo", window->logo,

```

```

01072     "website", "https://github.com/jburguete/mpcotool",
01073     "license-type", GTK_LICENSE_BSD, NULL);
01074 #if DEBUG_INTERFACE
01075     fprintf (stderr, "window_about: end\n");
01076 #endif
01077 }
01078
01084 void
01085 window_update_direction ()
01086 {
01087     #if DEBUG_INTERFACE
01088         fprintf (stderr, "window_update_direction: start\n");
01089     #endif
01090     gtk_widget_show (GTK_WIDGET (window->check_direction));
01091     if (gtk_toggle_button_get_active
01092         (GTK_TOGGLE_BUTTON (window->check_direction)))
01093     {
01094         gtk_widget_show (GTK_WIDGET (window->grid_direction));
01095         gtk_widget_show (GTK_WIDGET (window->label_step));
01096         gtk_widget_show (GTK_WIDGET (window->spin_step));
01097     }
01098     switch (window_get_direction ())
01099     {
01100         case DIRECTION_METHOD_COORDINATES:
01101             gtk_widget_hide (GTK_WIDGET (window->label_estimates));
01102             gtk_widget_hide (GTK_WIDGET (window->spin_estimates));
01103             break;
01104         default:
01105             gtk_widget_show (GTK_WIDGET (window->label_estimates));
01106             gtk_widget_show (GTK_WIDGET (window->spin_estimates));
01107     }
01108     #if DEBUG_INTERFACE
01109         fprintf (stderr, "window_update_direction: end\n");
01110     #endif
01111 }
01112
01117 void
01118 window_update ()
01119 {
01120     unsigned int i;
01121     #if DEBUG_INTERFACE
01122         fprintf (stderr, "window_update: start\n");
01123     #endif
01124     gtk_widget_set_sensitive
01125         (GTK_WIDGET (window->button_evaluator),
01126          gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
01127                                         (window->check_evaluator)));
01128     gtk_widget_hide (GTK_WIDGET (window->label_simulations));
01129     gtk_widget_hide (GTK_WIDGET (window->spin_simulations));
01130     gtk_widget_hide (GTK_WIDGET (window->label_iterations));
01131     gtk_widget_hide (GTK_WIDGET (window->spin_iterations));
01132     gtk_widget_hide (GTK_WIDGET (window->label_tolerance));
01133     gtk_widget_hide (GTK_WIDGET (window->spin_tolerance));
01134     gtk_widget_hide (GTK_WIDGET (window->label_bests));
01135     gtk_widget_hide (GTK_WIDGET (window->spin_bests));
01136     gtk_widget_hide (GTK_WIDGET (window->label_population));
01137     gtk_widget_hide (GTK_WIDGET (window->spin_population));
01138     gtk_widget_hide (GTK_WIDGET (window->label_generations));
01139     gtk_widget_hide (GTK_WIDGET (window->spin_generations));
01140     gtk_widget_hide (GTK_WIDGET (window->label_mutation));
01141     gtk_widget_hide (GTK_WIDGET (window->spin_mutation));
01142     gtk_widget_hide (GTK_WIDGET (window->label_reproduction));
01143     gtk_widget_hide (GTK_WIDGET (window->spin_reproduction));
01144     gtk_widget_hide (GTK_WIDGET (window->label_adaptation));
01145     gtk_widget_hide (GTK_WIDGET (window->spin_adaptation));
01146     gtk_widget_hide (GTK_WIDGET (window->label_sweeps));
01147     gtk_widget_hide (GTK_WIDGET (window->spin_sweeps));
01148     gtk_widget_hide (GTK_WIDGET (window->label_bits));
01149     gtk_widget_hide (GTK_WIDGET (window->spin_bits));
01150     gtk_widget_hide (GTK_WIDGET (window->check_direction));
01151     gtk_widget_hide (GTK_WIDGET (window->grid_direction));
01152     gtk_widget_hide (GTK_WIDGET (window->label_step));
01153     gtk_widget_hide (GTK_WIDGET (window->spin_step));
01154     gtk_widget_hide (GTK_WIDGET (window->label_p));
01155     gtk_widget_hide (GTK_WIDGET (window->spin_p));
01156     i = gtk_spin_button_get_value_as_int (window->spin_iterations);
01157     switch (window_get_algorithm ())
01158     {
01159         case ALGORITHM_MONTE_CARLO:
01160             gtk_widget_show (GTK_WIDGET (window->label_simulations));
01161             gtk_widget_show (GTK_WIDGET (window->spin_simulations));
01162             gtk_widget_show (GTK_WIDGET (window->label_iterations));
01163             gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01164             if (i > 1)
01165             {
01166                 gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01167                 gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01168             }
01169         }
01170     }

```

```

01168         gtk_widget_show (GTK_WIDGET (window->label_bests));
01169         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01170     }
01171     window_update_direction ();
01172     break;
01173 case ALGORITHM_SWEEP:
01174     gtk_widget_show (GTK_WIDGET (window->label_iterations));
01175     gtk_widget_show (GTK_WIDGET (window->spin_iterations));
01176     if (i > 1)
01177     {
01178         gtk_widget_show (GTK_WIDGET (window->label_tolerance));
01179         gtk_widget_show (GTK_WIDGET (window->spin_tolerance));
01180         gtk_widget_show (GTK_WIDGET (window->label_bests));
01181         gtk_widget_show (GTK_WIDGET (window->spin_bests));
01182     }
01183     gtk_widget_show (GTK_WIDGET (window->label_sweeps));
01184     gtk_widget_show (GTK_WIDGET (window->spin_sweeps));
01185     gtk_widget_show (GTK_WIDGET (window->check_direction));
01186     window_update_direction ();
01187     break;
01188 default:
01189     gtk_widget_show (GTK_WIDGET (window->label_population));
01190     gtk_widget_show (GTK_WIDGET (window->spin_population));
01191     gtk_widget_show (GTK_WIDGET (window->label_generations));
01192     gtk_widget_show (GTK_WIDGET (window->spin_generations));
01193     gtk_widget_show (GTK_WIDGET (window->label_mutation));
01194     gtk_widget_show (GTK_WIDGET (window->spin_mutation));
01195     gtk_widget_show (GTK_WIDGET (window->label_reproduction));
01196     gtk_widget_show (GTK_WIDGET (window->spin_reproduction));
01197     gtk_widget_show (GTK_WIDGET (window->label_adaptation));
01198     gtk_widget_show (GTK_WIDGET (window->spin_adaptation));
01199     gtk_widget_show (GTK_WIDGET (window->label_bits));
01200     gtk_widget_show (GTK_WIDGET (window->spin_bits));
01201 }
01202 gtk_widget_set_sensitive
01203 (GTK_WIDGET (window->button_remove_experiment),
01204  input->nexperiments > 1);
01204     gtk_widget_set_sensitive
01205 (GTK_WIDGET (window->button_remove_variable), input->
01206  nvariables > 1);
01206     for (i = 0; i < input->experiment->ninputs; ++i)
01207     {
01208         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01209         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01210         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 0);
01211         gtk_widget_set_sensitive (GTK_WIDGET (window->button_template[i]), 1);
01212         g_signal_handler_block
01213         (window->check_template[i], window->id_template[i]);
01214         g_signal_handler_block (window->button_template[i], window->
01215         id_input[i]);
01215         gtk_toggle_button_set_active
01216         (GTK_TOGGLE_BUTTON (window->check_template[i]), 1);
01217         g_signal_handler_unblock
01218         (window->button_template[i], window->id_input[i]);
01219         g_signal_handler_unblock
01220         (window->check_template[i], window->id_template[i]);
01221     }
01222     if (i > 0)
01223     {
01224         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i - 1]), 1);
01225         gtk_widget_set_sensitive
01226         (GTK_WIDGET (window->button_template[i - 1]),
01227          gtk_toggle_button_get_active
01228          (GTK_TOGGLE_BUTTON (window->check_template[i - 1])));
01229     }
01230     if (i < MAX_NINPUTS)
01231     {
01232         gtk_widget_show (GTK_WIDGET (window->check_template[i]));
01233         gtk_widget_show (GTK_WIDGET (window->button_template[i]));
01234         gtk_widget_set_sensitive (GTK_WIDGET (window->check_template[i]), 1);
01235         gtk_widget_set_sensitive
01236         (GTK_WIDGET (window->button_template[i]),
01237          gtk_toggle_button_get_active
01238          (GTK_TOGGLE_BUTTON (window->check_template[i])));
01239         g_signal_handler_block
01240         (window->check_template[i], window->id_template[i]);
01241         g_signal_handler_block (window->button_template[i], window->
01242         id_input[i]);
01242         gtk_toggle_button_set_active
01243         (GTK_TOGGLE_BUTTON (window->check_template[i]), 0);
01244         g_signal_handler_unblock
01245         (window->button_template[i], window->id_input[i]);
01246         g_signal_handler_unblock
01247         (window->check_template[i], window->id_template[i]);
01248     }
01249     while (++i < MAX_NINPUTS)
01250     {

```

```

01251     gtk_widget_hide (GTK_WIDGET (window->check_template[i]));
01252     gtk_widget_hide (GTK_WIDGET (window->button_template[i]));
01253 }
01254 gtk_widget_set_sensitive
01255 (GTK_WIDGET (window->spin_minabs),
01256  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_minabs)));
01257 gtk_widget_set_sensitive
01258 (GTK_WIDGET (window->spin_maxabs),
01259  gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (window->check_maxabs)));
01260 if (window_get_norm () == ERROR_NORM_P)
01261 {
01262     gtk_widget_show (GTK_WIDGET (window->label_p));
01263     gtk_widget_show (GTK_WIDGET (window->spin_p));
01264 }
01265 #if DEBUG_INTERFACE
01266 fprintf (stderr, "window_update: end\n");
01267 #endif
01268 }
01269
01274 void
01275 window_set_algorithm ()
01276 {
01277     int i;
01278     #if DEBUG_INTERFACE
01279     fprintf (stderr, "window_set_algorithm: start\n");
01280     #endif
01281     i = window_get_algorithm ();
01282     switch (i)
01283     {
01284     case ALGORITHM_SWEEP:
01285         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01286         if (i < 0)
01287             i = 0;
01288         gtk_spin_button_set_value (window->spin_sweeps,
01289                                   (gdouble) input->variable[i].
01290                                   nsweeps);
01291         break;
01292     case ALGORITHM_GENETIC:
01293         i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01294         if (i < 0)
01295             i = 0;
01296         gtk_spin_button_set_value (window->spin_bits,
01297                                   (gdouble) input->variable[i].nbits);
01298     }
01299     window_update ();
01300     #if DEBUG_INTERFACE
01301     fprintf (stderr, "window_set_algorithm: end\n");
01302     #endif
01303 }
01304
01308 void
01309 window_set_experiment ()
01310 {
01311     unsigned int i, j;
01312     char *buffer1, *buffer2;
01313     #if DEBUG_INTERFACE
01314     fprintf (stderr, "window_set_experiment: start\n");
01315     #endif
01316     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01317     gtk_spin_button_set_value (window->spin_weight, input->
01318                               experiment[i].weight);
01319     buffer1 = gtk_combo_box_text_get_active_text (window->combo_experiment);
01320     buffer2 = g_build_filename (input->directory, buffer1, NULL);
01321     g_free (buffer1);
01322     g_signal_handler_block
01323     (window->button_experiment, window->id_experiment_name);
01324     gtk_file_chooser_set_filename
01325     (GTK_FILE_CHOOSER (window->button_experiment), buffer2);
01326     g_signal_handler_unblock
01327     (window->button_experiment, window->id_experiment_name);
01328     g_free (buffer2);
01329     for (j = 0; j < input->experiment->ninputs; ++j)
01330     {
01331         g_signal_handler_block (window->button_template[j], window->
01332                               id_input[j]);
01333         buffer2 = g_build_filename (input->directory,
01334                                   input->experiment[i].template[j], NULL);
01335         gtk_file_chooser_set_filename
01336         (GTK_FILE_CHOOSER (window->button_template[j]), buffer2);
01337         g_free (buffer2);
01338         g_signal_handler_unblock
01339         (window->button_template[j], window->id_input[j]);
01340     }
01341     #if DEBUG_INTERFACE
01342     fprintf (stderr, "window_set_experiment: end\n");
01343     #endif
01344 }

```

```

01343
01348 void
01349 window_remove_experiment ()
01350 {
01351     unsigned int i, j;
01352     #if DEBUG_INTERFACE
01353         fprintf (stderr, "window_remove_experiment: start\n");
01354     #endif
01355     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01356     g_signal_handler_block (window->combo_experiment, window->
01357         id_experiment);
01357     gtk_combo_box_text_remove (window->combo_experiment, i);
01358     g_signal_handler_unblock (window->combo_experiment, window->
01359         id_experiment);
01359     experiment_free (input->experiment + i, input->
01360         type);
01360     --input->nexperiments;
01361     for (j = i; j < input->nexperiments; ++j)
01362         memcpy (input->experiment + j, input->experiment + j + 1,
01363             sizeof (Experiment));
01364     j = input->nexperiments - 1;
01365     if (i > j)
01366         i = j;
01367     for (j = 0; j < input->experiment->ninputs; ++j)
01368         g_signal_handler_block (window->button_template[j], window->
01369             id_input[j]);
01369     g_signal_handler_block
01370         (window->button_experiment, window->id_experiment_name);
01371     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01372     g_signal_handler_unblock
01373         (window->button_experiment, window->id_experiment_name);
01374     for (j = 0; j < input->experiment->ninputs; ++j)
01375         g_signal_handler_unblock (window->button_template[j], window->
01376             id_input[j]);
01376     window_update ();
01377     #if DEBUG_INTERFACE
01378         fprintf (stderr, "window_remove_experiment: end\n");
01379     #endif
01380 }
01381
01386 void
01387 window_add_experiment ()
01388 {
01389     unsigned int i, j;
01390     #if DEBUG_INTERFACE
01391         fprintf (stderr, "window_add_experiment: start\n");
01392     #endif
01393     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01394     g_signal_handler_block (window->combo_experiment, window->
01395         id_experiment);
01395     gtk_combo_box_text_insert_text
01396         (window->combo_experiment, i, input->experiment[i].
01397             name);
01397     g_signal_handler_unblock (window->combo_experiment, window->
01398         id_experiment);
01398     input->experiment = (Experiment *) g_realloc
01399         (input->experiment, (input->nexperiments + 1) * sizeof (
01400             Experiment));
01400     for (j = input->nexperiments - 1; j > i; --j)
01401         memcpy (input->experiment + j + 1, input->experiment + j,
01402             sizeof (Experiment));
01403     input->experiment[j + 1].weight = input->experiment[j].
01404         weight;
01404     input->experiment[j + 1].ninputs = input->
01405         experiment[j].ninputs;
01405     if (input->type == INPUT_TYPE_XML)
01406     {
01407         input->experiment[j + 1].name
01408             = (char *) xmlStrdup ((xmlChar *) input->experiment[j].
01409                 name);
01409         for (j = 0; j < input->experiment->ninputs; ++j)
01410             input->experiment[i + 1].template[j]
01411                 = (char *) xmlStrdup ((xmlChar *) input->experiment[i].
01412                     template[j]);
01412     }
01413     else
01414     {
01415         input->experiment[j + 1].name = g_strdup (input->
01416             experiment[j].name);
01416         for (j = 0; j < input->experiment->ninputs; ++j)
01417             input->experiment[i + 1].template[j]
01418                 = g_strdup (input->experiment[i].template[j]);
01419     }
01420     ++input->nexperiments;
01421     for (j = 0; j < input->experiment->ninputs; ++j)
01422         g_signal_handler_block (window->button_template[j], window->
01423             id_input[j]);

```

```

01423 g_signal_handler_block
01424 (window->button_experiment, window->id_experiment_name);
01425 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i + 1);
01426 g_signal_handler_unblock
01427 (window->button_experiment, window->id_experiment_name);
01428 for (j = 0; j < input->experiment->ninputs; ++j)
01429 g_signal_handler_unblock (window->button_template[j], window->
id_input[j]);
01430 window_update ();
01431 #if DEBUG_INTERFACE
01432 fprintf (stderr, "window_add_experiment: end\n");
01433 #endif
01434 }
01435
01440 void
01441 window_name_experiment ()
01442 {
01443     unsigned int i;
01444     char *buffer;
01445     GFile *file1, *file2;
01446     #if DEBUG_INTERFACE
01447     fprintf (stderr, "window_name_experiment: start\n");
01448     #endif
01449     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01450     file1
01451     = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_experiment));
01452     file2 = g_file_new_for_path (input->directory);
01453     buffer = g_file_get_relative_path (file2, file1);
01454     g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01455     gtk_combo_box_text_remove (window->combo_experiment, i);
01456     gtk_combo_box_text_insert_text (window->combo_experiment, i, buffer);
01457     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), i);
01458     g_signal_handler_unblock (window->combo_experiment, window->
id_experiment);
01459     g_free (buffer);
01460     g_object_unref (file2);
01461     g_object_unref (file1);
01462     #if DEBUG_INTERFACE
01463     fprintf (stderr, "window_name_experiment: end\n");
01464     #endif
01465 }
01466
01471 void
01472 window_weight_experiment ()
01473 {
01474     unsigned int i;
01475     #if DEBUG_INTERFACE
01476     fprintf (stderr, "window_weight_experiment: start\n");
01477     #endif
01478     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01479     input->experiment[i].weight = gtk_spin_button_get_value (window->
spin_weight);
01480     #if DEBUG_INTERFACE
01481     fprintf (stderr, "window_weight_experiment: end\n");
01482     #endif
01483 }
01484
01490 void
01491 window_inputs_experiment ()
01492 {
01493     unsigned int j;
01494     #if DEBUG_INTERFACE
01495     fprintf (stderr, "window_inputs_experiment: start\n");
01496     #endif
01497     j = input->experiment->ninputs - 1;
01498     if (j
01499         && !gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
(window->check_template[j])))
01500         --input->experiment->ninputs;
01501     if (input->experiment->ninputs < MAX_NINPUTS
01502         && gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON
(window->check_template[j])))
01503         ++input->experiment->ninputs;
01504     window_update ();
01505     #if DEBUG_INTERFACE
01506     fprintf (stderr, "window_inputs_experiment: end\n");
01507     #endif
01508 }
01509
01519 void
01520 window_template_experiment (void *data)
01521 {
01522     unsigned int i, j;
01523     char *buffer;
01524     GFile *file1, *file2;
01525     #if DEBUG_INTERFACE

```

```

01526     fprintf (stderr, "window_template_experiment: start\n");
01527 #endif
01528     i = (size_t) data;
01529     j = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_experiment));
01530     file1
01531         = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (window->button_template[i]));
01532     file2 = g_file_new_for_path (input->directory);
01533     buffer = g_file_get_relative_path (file2, file1);
01534     if (input->type == INPUT_TYPE_XML)
01535         input->experiment[j].template[i] = (char *) xmlStrdup ((xmlChar *) buffer);
01536     else
01537         input->experiment[j].template[i] = g_strdup (buffer);
01538     g_free (buffer);
01539     g_object_unref (file2);
01540     g_object_unref (file1);
01541 #if DEBUG_INTERFACE
01542     fprintf (stderr, "window_template_experiment: end\n");
01543 #endif
01544 }
01545
01550 void
01551 window_set_variable ()
01552 {
01553     unsigned int i;
01554 #if DEBUG_INTERFACE
01555     fprintf (stderr, "window_set_variable: start\n");
01556 #endif
01557     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01558     g_signal_handler_block (window->entry_variable, window->
01559         id_variable_label);
01559     gtk_entry_set_text (window->entry_variable, input->variable[i].
01560         name);
01560     g_signal_handler_unblock (window->entry_variable, window->
01561         id_variable_label);
01561     gtk_spin_button_set_value (window->spin_min, input->variable[i].
01562         rangemin);
01562     gtk_spin_button_set_value (window->spin_max, input->variable[i].
01563         rangemax);
01563     if (input->variable[i].rangeminabs != -G_MAXDOUBLE)
01564     {
01565         gtk_spin_button_set_value (window->spin_minabs,
01566             input->variable[i].rangeminabs);
01567         gtk_toggle_button_set_active
01568             (GTK_TOGGLE_BUTTON (window->check_minabs), 1);
01569     }
01570     else
01571     {
01572         gtk_spin_button_set_value (window->spin_minabs, -G_MAXDOUBLE);
01573         gtk_toggle_button_set_active
01574             (GTK_TOGGLE_BUTTON (window->check_minabs), 0);
01575     }
01576     if (input->variable[i].rangemaxabs != G_MAXDOUBLE)
01577     {
01578         gtk_spin_button_set_value (window->spin_maxabs,
01579             input->variable[i].rangemaxabs);
01580         gtk_toggle_button_set_active
01581             (GTK_TOGGLE_BUTTON (window->check_maxabs), 1);
01582     }
01583     else
01584     {
01585         gtk_spin_button_set_value (window->spin_maxabs, G_MAXDOUBLE);
01586         gtk_toggle_button_set_active
01587             (GTK_TOGGLE_BUTTON (window->check_maxabs), 0);
01588     }
01589     gtk_spin_button_set_value (window->spin_precision,
01590         input->variable[i].precision);
01591     gtk_spin_button_set_value (window->spin_steps, (gdouble) input->
01592         nsteps);
01592     if (input->nsteps)
01593         gtk_spin_button_set_value (window->spin_step, input->variable[i].
01594             step);
01594 #if DEBUG_INTERFACE
01595     fprintf (stderr, "window_set_variable: precision[%u]=%u\n", i,
01596         input->variable[i].precision);
01597 #endif
01598     switch (window_get_algorithm ())
01599     {
01600     case ALGORITHM_SWEEP:
01601         gtk_spin_button_set_value (window->spin_sweeps,
01602             (gdouble) input->variable[i].
01603                 nsweeps);
01603 #if DEBUG_INTERFACE
01604         fprintf (stderr, "window_set_variable: nsweeps[%u]=%u\n", i,
01605             input->variable[i].nsweeps);
01606 #endif
01607         break;
01608     case ALGORITHM_GENETIC:

```

```

01609     gtk_spin_button_set_value (window->spin_bits,
01610                               (gdouble) input->variable[i].nbits);
01611 #if DEBUG_INTERFACE
01612     fprintf (stderr, "window_set_variable: nbits[%u]=%u\n", i,
01613             input->variable[i].nbits);
01614 #endif
01615     break;
01616 }
01617 window_update ();
01618 #if DEBUG_INTERFACE
01619     fprintf (stderr, "window_set_variable: end\n");
01620 #endif
01621 }
01622
01627 void
01628 window_remove_variable ()
01629 {
01630     unsigned int i, j;
01631 #if DEBUG_INTERFACE
01632     fprintf (stderr, "window_remove_variable: start\n");
01633 #endif
01634     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01635     g_signal_handler_block (window->combo_variable, window->
01636                             id_variable);
01637     gtk_combo_box_text_remove (window->combo_variable, i);
01638     g_signal_handler_unblock (window->combo_variable, window->
01639                              id_variable);
01640     xmlFree (input->variable[i].name);
01641     --input->nvariables;
01642     for (j = i; j < input->nvariables; ++j)
01643         memcpy (input->variable + j, input->variable + j + 1, sizeof (
01644                 Variable));
01645     j = input->nvariables - 1;
01646     if (i > j)
01647         i = j;
01648     g_signal_handler_block (window->entry_variable, window->
01649                             id_variable_label);
01650     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01651     g_signal_handler_unblock (window->entry_variable, window->
01652                              id_variable_label);
01653     window_update ();
01654 #if DEBUG_INTERFACE
01655     fprintf (stderr, "window_remove_variable: end\n");
01656 #endif
01657 }
01658 void
01659 window_add_variable ()
01660 {
01661     unsigned int i, j;
01662 #if DEBUG_INTERFACE
01663     fprintf (stderr, "window_add_variable: start\n");
01664 #endif
01665     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01666     g_signal_handler_block (window->combo_variable, window->
01667                             id_variable);
01668     gtk_combo_box_text_insert_text (window->combo_variable, i,
01669                                     input->variable[i].name);
01670     g_signal_handler_unblock (window->combo_variable, window->
01671                              id_variable);
01672     input->variable = (Variable *) g_realloc
01673         (input->variable, (input->nvariables + 1) * sizeof (
01674         Variable));
01675     for (j = input->nvariables - 1; j > i; --j)
01676         memcpy (input->variable + j + 1, input->variable + j, sizeof (
01677                 Variable));
01678     memcpy (input->variable + j + 1, input->variable + j, sizeof (
01679                 Variable));
01680     if (input->type == INPUT_TYPE_XML)
01681         input->variable[j + 1].name
01682             = (char *) xmlStrdup ((xmlChar *) input->variable[j].name);
01683     else
01684         input->variable[j + 1].name = g_strdup (input->
01685         variable[j].name);
01686     ++input->nvariables;
01687     g_signal_handler_block (window->entry_variable, window->
01688                             id_variable_label);
01689     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i + 1);
01690     g_signal_handler_unblock (window->entry_variable, window->
01691                              id_variable_label);
01692     window_update ();
01693 #if DEBUG_INTERFACE
01694     fprintf (stderr, "window_add_variable: end\n");
01695 #endif
01696 }
01697 void

```



```

01695 window_label_variable ()
01696 {
01697     unsigned int i;
01698     const char *buffer;
01699     #if DEBUG_INTERFACE
01700     fprintf (stderr, "window_label_variable: start\n");
01701     #endif
01702     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01703     buffer = gtk_entry_get_text (window->entry_variable);
01704     g_signal_handler_block (window->combo_variable, window->
01705         id_variable);
01705     gtk_combo_box_text_remove (window->combo_variable, i);
01706     gtk_combo_box_text_insert_text (window->combo_variable, i, buffer);
01707     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), i);
01708     g_signal_handler_unblock (window->combo_variable, window->
01709         id_variable);
01709     #if DEBUG_INTERFACE
01710     fprintf (stderr, "window_label_variable: end\n");
01711     #endif
01712 }
01713
01718 void
01719 window_precision_variable ()
01720 {
01721     unsigned int i;
01722     #if DEBUG_INTERFACE
01723     fprintf (stderr, "window_precision_variable: start\n");
01724     #endif
01725     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01726     input->variable[i].precision
01727     = (unsigned int) gtk_spin_button_get_value_as_int (window->spin_precision);
01728     gtk_spin_button_set_digits (window->spin_min, input->variable[i].
01729         precision);
01729     gtk_spin_button_set_digits (window->spin_max, input->variable[i].
01730         precision);
01730     gtk_spin_button_set_digits (window->spin_minabs,
01731         input->variable[i].precision);
01732     gtk_spin_button_set_digits (window->spin_maxabs,
01733         input->variable[i].precision);
01734     #if DEBUG_INTERFACE
01735     fprintf (stderr, "window_precision_variable: end\n");
01736     #endif
01737 }
01738
01743 void
01744 window_rangemin_variable ()
01745 {
01746     unsigned int i;
01747     #if DEBUG_INTERFACE
01748     fprintf (stderr, "window_rangemin_variable: start\n");
01749     #endif
01750     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01751     input->variable[i].rangemin = gtk_spin_button_get_value (window->
01752         spin_min);
01752     #if DEBUG_INTERFACE
01753     fprintf (stderr, "window_rangemin_variable: end\n");
01754     #endif
01755 }
01756
01761 void
01762 window_rangemax_variable ()
01763 {
01764     unsigned int i;
01765     #if DEBUG_INTERFACE
01766     fprintf (stderr, "window_rangemax_variable: start\n");
01767     #endif
01768     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01769     input->variable[i].rangemax = gtk_spin_button_get_value (window->
01770         spin_max);
01770     #if DEBUG_INTERFACE
01771     fprintf (stderr, "window_rangemax_variable: end\n");
01772     #endif
01773 }
01774
01779 void
01780 window_rangeminabs_variable ()
01781 {
01782     unsigned int i;
01783     #if DEBUG_INTERFACE
01784     fprintf (stderr, "window_rangeminabs_variable: start\n");
01785     #endif
01786     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01787     input->variable[i].rangeminabs
01788     = gtk_spin_button_get_value (window->spin_minabs);
01789     #if DEBUG_INTERFACE
01790     fprintf (stderr, "window_rangeminabs_variable: end\n");
01791     #endif

```

```

01792 }
01793
01798 void
01799 window_rangemaxabs_variable ()
01800 {
01801     unsigned int i;
01802     #if DEBUG_INTERFACE
01803     fprintf (stderr, "window_rangemaxabs_variable: start\n");
01804     #endif
01805     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01806     input->variable[i].rangemaxabs
01807     = gtk_spin_button_get_value (window->spin_maxabs);
01808     #if DEBUG_INTERFACE
01809     fprintf (stderr, "window_rangemaxabs_variable: end\n");
01810     #endif
01811 }
01812
01817 void
01818 window_step_variable ()
01819 {
01820     unsigned int i;
01821     #if DEBUG_INTERFACE
01822     fprintf (stderr, "window_step_variable: start\n");
01823     #endif
01824     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01825     input->variable[i].step = gtk_spin_button_get_value (window->
    spin_step);
01826     #if DEBUG_INTERFACE
01827     fprintf (stderr, "window_step_variable: end\n");
01828     #endif
01829 }
01830
01835 void
01836 window_update_variable ()
01837 {
01838     int i;
01839     #if DEBUG_INTERFACE
01840     fprintf (stderr, "window_update_variable: start\n");
01841     #endif
01842     i = gtk_combo_box_get_active (GTK_COMBO_BOX (window->combo_variable));
01843     if (i < 0)
01844         i = 0;
01845     switch (window_get_algorithm ())
01846     {
01847         case ALGORITHM_SWEEP:
01848             input->variable[i].nsweeps
01849             = gtk_spin_button_get_value_as_int (window->spin_sweeps);
01850             #if DEBUG_INTERFACE
01851             fprintf (stderr, "window_update_variable: nsweeps[%d]=%u\n", i,
01852                 input->variable[i].nsweeps);
01853             #endif
01854             break;
01855         case ALGORITHM_GENETIC:
01856             input->variable[i].nbits
01857             = gtk_spin_button_get_value_as_int (window->spin_bits);
01858             #if DEBUG_INTERFACE
01859             fprintf (stderr, "window_update_variable: nbits[%d]=%u\n", i,
01860                 input->variable[i].nbits);
01861             #endif
01862     }
01863     #if DEBUG_INTERFACE
01864     fprintf (stderr, "window_update_variable: end\n");
01865     #endif
01866 }
01867
01875 int
01876 window_read (char *filename)
01877 {
01878     unsigned int i;
01879     char *buffer;
01880     #if DEBUG_INTERFACE
01881     fprintf (stderr, "window_read: start\n");
01882     #endif
01883
01884     // Reading new input file
01885     input_free ();
01886     if (!input_open (filename))
01887     {
01888         #if DEBUG_INTERFACE
01889         fprintf (stderr, "window_read: end\n");
01890         #endif
01891         return 0;
01892     }
01893
01894     // Setting GTK+ widgets data
01895     gtk_entry_set_text (window->entry_result, input->result);
01896     gtk_entry_set_text (window->entry_variables, input->

```

```

    variables);
01897     buffer = g_build_filename (input->directory, input->
    simulator, NULL);
01898     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01899                                  (window->button_simulator), buffer);
01900     g_free (buffer);
01901     gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01902                                  (size_t) input->evaluator);
01903     if (input->evaluator)
01904     {
01905         buffer = g_build_filename (input->directory, input->
    evaluator, NULL);
01906         gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01907                                         (window->button_evaluator), buffer);
01908         g_free (buffer);
01909     }
01910     gtk_toggle_button_set_active
01911     (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
    algorithm]), TRUE);
01912     switch (input->algorithm)
01913     {
01914         case ALGORITHM_MONTE_CARLO:
01915             gtk_spin_button_set_value (window->spin_simulations,
01916                                         (gdouble) input->nsimulations);
01917         case ALGORITHM_SWEEP:
01918             gtk_spin_button_set_value (window->spin_iterations,
01919                                         (gdouble) input->niterations);
01920             gtk_spin_button_set_value (window->spin_bests, (gdouble) input->
    nbest);
01921             gtk_spin_button_set_value (window->spin_tolerance, input->
    tolerance);
01922             gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_direction),
01923                                           input->nsteps);
01924             if (input->nsteps)
01925             {
01926                 gtk_toggle_button_set_active
01927                 (GTK_TOGGLE_BUTTON (window->button_direction
01928                                     [input->direction]), TRUE);
01929                 gtk_spin_button_set_value (window->spin_steps,
01930                                             (gdouble) input->nsteps);
01931                 gtk_spin_button_set_value (window->spin_relaxation,
01932                                             (gdouble) input->relaxation);
01933                 switch (input->direction)
01934                 {
01935                     case DIRECTION_METHOD_RANDOM:
01936                         gtk_spin_button_set_value (window->spin_estimates,
01937                                                     (gdouble) input->nestimates);
01938                 }
01939             }
01940             break;
01941         default:
01942             gtk_spin_button_set_value (window->spin_population,
01943                                         (gdouble) input->nsimulations);
01944             gtk_spin_button_set_value (window->spin_generations,
01945                                         (gdouble) input->niterations);
01946             gtk_spin_button_set_value (window->spin_mutation, input->
    mutation_ratio);
01947             gtk_spin_button_set_value (window->spin_reproduction,
01948                                         input->reproduction_ratio);
01949             gtk_spin_button_set_value (window->spin_adaptation,
01950                                         input->adaptation_ratio);
01951     }
01952     gtk_toggle_button_set_active
01953     (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01954     gtk_spin_button_set_value (window->spin_p, input->p);
01955     gtk_spin_button_set_value (window->spin_threshold, input->
    threshold);
01956     g_signal_handler_block (window->combo_experiment, window->
    id_experiment);
01957     g_signal_handler_block (window->button_experiment,
01958                             window->id_experiment_name);
01959     gtk_combo_box_text_remove_all (window->combo_experiment);
01960     for (i = 0; i < input->nexperiments; ++i)
01961         gtk_combo_box_text_append_text (window->combo_experiment,
01962                                         input->experiment[i].name);
01963     g_signal_handler_unblock
01964     (window->button_experiment, window->id_experiment_name);
01965     g_signal_handler_unblock (window->combo_experiment, window->
    id_experiment);
01966     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01967     g_signal_handler_block (window->combo_variable, window->
    id_variable);
01968     g_signal_handler_block (window->entry_variable, window->
    id_variable_label);
01969     gtk_combo_box_text_remove_all (window->combo_variable);
01970     for (i = 0; i < input->nvariables; ++i)
01971         gtk_combo_box_text_append_text (window->combo_variable,

```

```

01972                                     input->variable[i].name);
01973 g_signal_handler_unblock (window->entry_variable, window->
id_variable_label);
01974 g_signal_handler_unblock (window->combo_variable, window->
id_variable);
01975 gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976 window_set_variable ();
01977 window_update ();
01978
01979 #if DEBUG_INTERFACE
01980 fprintf (stderr, "window_read: end\n");
01981 #endif
01982 return 1;
01983 }
01984
01989 void
01990 window_open ()
01991 {
01992     GtkFileChooserDialog *dlg;
01993     GtkFileFilter *filter;
01994     char *buffer, *directory, *name;
01995
01996     #if DEBUG_INTERFACE
01997         fprintf (stderr, "window_open: start\n");
01998     #endif
01999
02000     // Saving a backup of the current input file
02001     directory = g_strdup (input->directory);
02002     name = g_strdup (input->name);
02003
02004     // Opening dialog
02005     dlg = (GtkFileChooserDialog *)
02006         gtk_file_chooser_dialog_new (gettext ("Open input file"),
02007                                     window->window,
02008                                     GTK_FILE_CHOOSER_ACTION_OPEN,
02009                                     gettext ("Cancel"), GTK_RESPONSE_CANCEL,
02010                                     gettext ("OK"), GTK_RESPONSE_OK, NULL);
02011
02012     // Adding XML filter
02013     filter = (GtkFileFilter *) gtk_file_filter_new ();
02014     gtk_file_filter_set_name (filter, "XML");
02015     gtk_file_filter_add_pattern (filter, "*.xml");
02016     gtk_file_filter_add_pattern (filter, "*.XML");
02017     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02018
02019     // Adding JSON filter
02020     filter = (GtkFileFilter *) gtk_file_filter_new ();
02021     gtk_file_filter_set_name (filter, "JSON");
02022     gtk_file_filter_add_pattern (filter, "*.json");
02023     gtk_file_filter_add_pattern (filter, "*.JSON");
02024     gtk_file_filter_add_pattern (filter, "*.js");
02025     gtk_file_filter_add_pattern (filter, "*.JS");
02026     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter);
02027
02028     // If OK saving
02029     while (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
02030     {
02031
02032         // Traying to open the input file
02033         buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
02034         if (!window_read (buffer))
02035         {
02036             #if DEBUG_INTERFACE
02037                 fprintf (stderr, "window_open: error reading input file\n");
02038             #endif
02039             g_free (buffer);
02040
02041             // Reading backup file on error
02042             buffer = g_build_filename (directory, name, NULL);
02043             if (!input_open (buffer))
02044             {
02045
02046                 // Closing on backup file reading error
02047                 #if DEBUG_INTERFACE
02048                     fprintf (stderr, "window_read: error reading backup file\n");
02049                 #endif
02050                 g_free (buffer);
02051                 break;
02052             }
02053             g_free (buffer);
02054         }
02055         else
02056         {
02057             g_free (buffer);
02058             break;
02059         }
02060     }

```

```

02061
02062 // Freeing and closing
02063 g_free (name);
02064 g_free (directory);
02065 gtk_widget_destroy (GTK_WIDGET (dlg));
02066 #if DEBUG_INTERFACE
02067 fprintf (stderr, "window_open: end\n");
02068 #endif
02069 }
02070
02071 void
02072 window_new ()
02073 {
02074     unsigned int i;
02075     char *buffer, *buffer2, buffer3[64];
02076     char *label_algorithm[NALGORITHMS] = {
02077         "Monte-Carlo", gettext ("_Sweep"), gettext ("_Genetic")
02078     };
02079     char *tip_algorithm[NALGORITHMS] = {
02080         gettext ("Monte-Carlo brute force algorithm"),
02081         gettext ("Sweep brute force algorithm"),
02082         gettext ("Genetic algorithm")
02083     };
02084     char *label_direction[N DIRECTIONS] = {
02085         gettext ("_Coordinates descent"), gettext ("_Random")
02086     };
02087     char *tip_direction[N DIRECTIONS] = {
02088         gettext ("Coordinates direction estimate method"),
02089         gettext ("Random direction estimate method")
02090     };
02091     char *label_norm[NNORMS] = { "L2", "L", "Lp", "L1" };
02092     char *tip_norm[NNORMS] = {
02093         gettext ("Euclidean error norm (L2)"),
02094         gettext ("Maximum error norm (L)"),
02095         gettext ("P error norm (Lp)"),
02096         gettext ("Taxicab error norm (L1)")
02097     };
02098     #if DEBUG_INTERFACE
02099     fprintf (stderr, "window_new: start\n");
02100     #endif
02101     // Creating the window
02102     window->window = main_window
02103     = (GtkWindow *) gtk_window_new (GTK_WINDOW_TOPLEVEL);
02104
02105     // Finish when closing the window
02106     g_signal_connect (window->window, "delete-event", gtk_main_quit, NULL);
02107
02108     // Setting the window title
02109     gtk_window_set_title (window->window, "MPCOTool");
02110
02111     // Creating the open button
02112     window->button_open = (GtkToolButton *) gtk_tool_button_new
02113     (gtk_image_new_from_icon_name ("document-open",
02114     GTK_ICON_SIZE_LARGE_TOOLBAR),
02115     gettext ("Open"));
02116     g_signal_connect (window->button_open, "clicked", window_open, NULL);
02117
02118     // Creating the save button
02119     window->button_save = (GtkToolButton *) gtk_tool_button_new
02120     (gtk_image_new_from_icon_name ("document-save",
02121     GTK_ICON_SIZE_LARGE_TOOLBAR),
02122     gettext ("Save"));
02123     g_signal_connect (window->button_save, "clicked", (void (*)(void))
02124     window_save,
02125     NULL);
02126
02127     // Creating the run button
02128     window->button_run = (GtkToolButton *) gtk_tool_button_new
02129     (gtk_image_new_from_icon_name ("system-run",
02130     GTK_ICON_SIZE_LARGE_TOOLBAR),
02131     gettext ("Run"));
02132     g_signal_connect (window->button_run, "clicked", window_run, NULL);
02133
02134     // Creating the options button
02135     window->button_options = (GtkToolButton *) gtk_tool_button_new
02136     (gtk_image_new_from_icon_name ("preferences-system",
02137     GTK_ICON_SIZE_LARGE_TOOLBAR),
02138     gettext ("Options"));
02139     g_signal_connect (window->button_options, "clicked", options_new, NULL);
02140
02141     // Creating the help button
02142     window->button_help = (GtkToolButton *) gtk_tool_button_new
02143     (gtk_image_new_from_icon_name ("help-browser",
02144     GTK_ICON_SIZE_LARGE_TOOLBAR),
02145     gettext ("Help"));
02146 }

```

```

02151 g_signal_connect (window->button_help, "clicked", window_help, NULL);
02152
02153 // Creating the about button
02154 window->button_about = (GtkToolButton *) gtk_tool_button_new
02155     (gtk_image_new_from_icon_name ("help-about",
02156         GTK_ICON_SIZE_LARGE_TOOLBAR),
02157     gettext ("About"));
02158 g_signal_connect (window->button_about, "clicked", window_about, NULL);
02159
02160 // Creating the exit button
02161 window->button_exit = (GtkToolButton *) gtk_tool_button_new
02162     (gtk_image_new_from_icon_name ("application-exit",
02163         GTK_ICON_SIZE_LARGE_TOOLBAR),
02164     gettext ("Exit"));
02165 g_signal_connect (window->button_exit, "clicked", gtk_main_quit, NULL);
02166
02167 // Creating the buttons bar
02168 window->bar_buttons = (GtkToolbar *) gtk_toolbar_new ();
02169 gtk_toolbar_insert
02170     (window->bar_buttons, GTK_TOOL_ITEM (window->button_open), 0);
02171 gtk_toolbar_insert
02172     (window->bar_buttons, GTK_TOOL_ITEM (window->button_save), 1);
02173 gtk_toolbar_insert
02174     (window->bar_buttons, GTK_TOOL_ITEM (window->button_run), 2);
02175 gtk_toolbar_insert
02176     (window->bar_buttons, GTK_TOOL_ITEM (window->button_options), 3);
02177 gtk_toolbar_insert
02178     (window->bar_buttons, GTK_TOOL_ITEM (window->button_help), 4);
02179 gtk_toolbar_insert
02180     (window->bar_buttons, GTK_TOOL_ITEM (window->button_about), 5);
02181 gtk_toolbar_insert
02182     (window->bar_buttons, GTK_TOOL_ITEM (window->button_exit), 6);
02183 gtk_toolbar_set_style (window->bar_buttons, GTK_TOOLBAR_BOTH);
02184
02185 // Creating the simulator program label and entry
02186 window->label_simulator
02187     = (GtkLabel *) gtk_label_new (gettext ("Simulator program"));
02188 window->button_simulator = (GtkFileChooserButton *)
02189     gtk_file_chooser_button_new (gettext ("Simulator program"),
02190         GTK_FILE_CHOOSER_ACTION_OPEN);
02191 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_simulator),
02192     gettext ("Simulator program executable file"));
02193 gtk_widget_set_hexexpand (GTK_WIDGET (window->button_simulator), TRUE);
02194
02195 // Creating the evaluator program label and entry
02196 window->check_evaluator = (GtkCheckButton *)
02197     gtk_check_button_new_with_mnemonic (gettext ("Evaluator program"));
02198 g_signal_connect (window->check_evaluator, "toggled",
02199     window_update, NULL);
02200 window->button_evaluator = (GtkFileChooserButton *)
02201     gtk_file_chooser_button_new (gettext ("Evaluator program"),
02202         GTK_FILE_CHOOSER_ACTION_OPEN);
02203 gtk_widget_set_tooltip_text
02204     (GTK_WIDGET (window->button_evaluator),
02205     gettext ("Optional evaluator program executable file"));
02206
02207 // Creating the results files labels and entries
02208 window->label_result = (GtkLabel *) gtk_label_new (gettext ("Result file"));
02209 window->entry_result = (GtkEntry *) gtk_entry_new ();
02210 gtk_widget_set_tooltip_text
02211     (GTK_WIDGET (window->entry_result), gettext ("Best results file"));
02212 window->label_variables
02213     = (GtkLabel *) gtk_label_new (gettext ("Variables file"));
02214 window->entry_variables = (GtkEntry *) gtk_entry_new ();
02215 gtk_widget_set_tooltip_text
02216     (GTK_WIDGET (window->entry_variables),
02217     gettext ("All simulated results file"));
02218
02219 // Creating the files grid and attaching widgets
02220 window->grid_files = (GtkGrid *) gtk_grid_new ();
02221 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02222     label_simulator),
02223     0, 0, 1, 1);
02224 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02225     button_simulator),
02226     1, 0, 1, 1);
02227 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02228     check_evaluator),
02229     0, 1, 1, 1);
02230 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02231     button_evaluator),
02232     1, 1, 1, 1);
02233 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02234     label_result),
02235     0, 2, 1, 1);
02236 gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
02237     entry_result),

```

```

02231         1, 2, 1, 1);
02232     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
label_variables),
02233         0, 3, 1, 1);
02234     gtk_grid_attach (window->grid_files, GTK_WIDGET (window->
entry_variables),
02235         1, 3, 1, 1);
02236
02237     // Creating the algorithm properties
02238     window->label_simulations = (GtkLabel *) gtk_label_new
02239         (gettext ("Simulations number"));
02240     window->spin_simulations
02241         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02242     gtk_widget_set_tooltip_text
02243         (GTK_WIDGET (window->spin_simulations),
02244         gettext ("Number of simulations to perform for each iteration"));
02245     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_simulations), TRUE);
02246     window->label_iterations = (GtkLabel *)
02247         gtk_label_new (gettext ("Iterations number"));
02248     window->spin_iterations
02249         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02250     gtk_widget_set_tooltip_text
02251         (GTK_WIDGET (window->spin_iterations), gettext ("Number of iterations"));
02252     g_signal_connect
02253         (window->spin_iterations, "value-changed", window_update, NULL);
02254     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_iterations), TRUE);
02255     window->label_tolerance = (GtkLabel *) gtk_label_new (gettext ("Tolerance"));
02256     window->spin_tolerance
02257         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02258     gtk_widget_set_tooltip_text
02259         (GTK_WIDGET (window->spin_tolerance),
02260         gettext ("Tolerance to set the variable interval on the next iteration"));
02261     window->label_best = (GtkLabel *) gtk_label_new (gettext ("Bests number"));
02262     window->spin_best
02263         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02264     gtk_widget_set_tooltip_text
02265         (GTK_WIDGET (window->spin_best),
02266         gettext ("Number of best simulations used to set the variable interval "
02267         "on the next iteration"));
02268     window->label_population
02269         = (GtkLabel *) gtk_label_new (gettext ("Population number"));
02270     window->spin_population
02271         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02272     gtk_widget_set_tooltip_text
02273         (GTK_WIDGET (window->spin_population),
02274         gettext ("Number of population for the genetic algorithm"));
02275     gtk_widget_set_hexexpand (GTK_WIDGET (window->spin_population), TRUE);
02276     window->label_generations
02277         = (GtkLabel *) gtk_label_new (gettext ("Generations number"));
02278     window->spin_generations
02279         = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e6, 1.);
02280     gtk_widget_set_tooltip_text
02281         (GTK_WIDGET (window->spin_generations),
02282         gettext ("Number of generations for the genetic algorithm"));
02283     window->label_mutation
02284         = (GtkLabel *) gtk_label_new (gettext ("Mutation ratio"));
02285     window->spin_mutation
02286         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02287     gtk_widget_set_tooltip_text
02288         (GTK_WIDGET (window->spin_mutation),
02289         gettext ("Ratio of mutation for the genetic algorithm"));
02290     window->label_reproduction
02291         = (GtkLabel *) gtk_label_new (gettext ("Reproduction ratio"));
02292     window->spin_reproduction
02293         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02294     gtk_widget_set_tooltip_text
02295         (GTK_WIDGET (window->spin_reproduction),
02296         gettext ("Ratio of reproduction for the genetic algorithm"));
02297     window->label_adaptation
02298         = (GtkLabel *) gtk_label_new (gettext ("Adaptation ratio"));
02299     window->spin_adaptation
02300         = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02301     gtk_widget_set_tooltip_text
02302         (GTK_WIDGET (window->spin_adaptation),
02303         gettext ("Ratio of adaptation for the genetic algorithm"));
02304     window->label_threshold = (GtkLabel *) gtk_label_new (gettext ("Threshold"));
02305     window->spin_threshold = (GtkSpinButton *) gtk_spin_button_new_with_range
02306         (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02307     gtk_widget_set_tooltip_text
02308         (GTK_WIDGET (window->spin_threshold),
02309         gettext ("Threshold in the objective function to finish the simulations"));
02310     window->scrolled_threshold
02311         = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02312     gtk_container_add (GTK_CONTAINER (window->scrolled_threshold),
02313         GTK_WIDGET (window->spin_threshold));
02314     // gtk_widget_set_hexexpand (GTK_WIDGET (window->scrolled_threshold), TRUE);
02315     // gtk_widget_set_halign (GTK_WIDGET (window->scrolled_threshold),

```

```

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

```



```

02396 gtk_grid_attach (window->grid_algorithm,
02397                 GTK_WIDGET (window->label_iterations), 0,
02398                 NALGORITHMS + 1, 1, 1);
02399 gtk_grid_attach (window->grid_algorithm,
02400                 GTK_WIDGET (window->spin_iterations), 1,
02401                 NALGORITHMS + 1, 1, 1);
02402 gtk_grid_attach (window->grid_algorithm,
02403                 GTK_WIDGET (window->label_tolerance), 0,
02404                 NALGORITHMS + 2, 1, 1);
02405 gtk_grid_attach (window->grid_algorithm,
02406                 GTK_WIDGET (window->spin_tolerance), 1,
02407                 NALGORITHMS + 2, 1, 1);
02408 gtk_grid_attach (window->grid_algorithm,
02409                 GTK_WIDGET (window->label_bests), 0, NALGORITHMS + 3, 1, 1);
02410 gtk_grid_attach (window->grid_algorithm,
02411                 GTK_WIDGET (window->spin_bests), 1, NALGORITHMS + 3, 1, 1);
02412 gtk_grid_attach (window->grid_algorithm,
02413                 GTK_WIDGET (window->label_population), 0,
02414                 NALGORITHMS + 4, 1, 1);
02415 gtk_grid_attach (window->grid_algorithm,
02416                 GTK_WIDGET (window->spin_population), 1,
02417                 NALGORITHMS + 4, 1, 1);
02418 gtk_grid_attach (window->grid_algorithm,
02419                 GTK_WIDGET (window->label_generations), 0,
02420                 NALGORITHMS + 5, 1, 1);
02421 gtk_grid_attach (window->grid_algorithm,
02422                 GTK_WIDGET (window->spin_generations), 1,
02423                 NALGORITHMS + 5, 1, 1);
02424 gtk_grid_attach (window->grid_algorithm,
02425                 GTK_WIDGET (window->label_mutation), 0,
02426                 NALGORITHMS + 6, 1, 1);
02427 gtk_grid_attach (window->grid_algorithm,
02428                 GTK_WIDGET (window->spin_mutation), 1,
02429                 NALGORITHMS + 6, 1, 1);
02430 gtk_grid_attach (window->grid_algorithm,
02431                 GTK_WIDGET (window->label_reproduction), 0,
02432                 NALGORITHMS + 7, 1, 1);
02433 gtk_grid_attach (window->grid_algorithm,
02434                 GTK_WIDGET (window->spin_reproduction), 1,
02435                 NALGORITHMS + 7, 1, 1);
02436 gtk_grid_attach (window->grid_algorithm,
02437                 GTK_WIDGET (window->label_adaptation), 0,
02438                 NALGORITHMS + 8, 1, 1);
02439 gtk_grid_attach (window->grid_algorithm,
02440                 GTK_WIDGET (window->spin_adaptation), 1,
02441                 NALGORITHMS + 8, 1, 1);
02442 gtk_grid_attach (window->grid_algorithm,
02443                 GTK_WIDGET (window->check_direction), 0,
02444                 NALGORITHMS + 9, 2, 1);
02445 gtk_grid_attach (window->grid_algorithm,
02446                 GTK_WIDGET (window->grid_direction), 0,
02447                 NALGORITHMS + 10, 2, 1);
02448 gtk_grid_attach (window->grid_algorithm, GTK_WIDGET (window->
label_threshold),
02449                 0, NALGORITHMS + 11, 1, 1);
02450 gtk_grid_attach (window->grid_algorithm,
02451                 GTK_WIDGET (window->scrolled_threshold), 1,
02452                 NALGORITHMS + 11, 1, 1);
02453 window->frame_algorithm = (GtkFrame *) gtk_frame_new (gettext ("Algorithm"));
02454 gtk_container_add (GTK_CONTAINER (window->frame_algorithm),
02455                 GTK_WIDGET (window->grid_algorithm));
02456
02457 // Creating the variable widgets
02458 window->combo_variable = (GtkComboBoxText *) gtk_combo_box_text_new ();
02459 gtk_widget_set_tooltip_text
02460     (GTK_WIDGET (window->combo_variable), gettext ("Variables selector"));
02461 window->id_variable = g_signal_connect
02462     (window->combo_variable, "changed", window_set_variable, NULL);
02463 window->button_add_variable
02464     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02465                 GTK_ICON_SIZE_BUTTON);
02466 g_signal_connect
02467     (window->button_add_variable, "clicked",
window_add_variable, NULL);
02468 gtk_widget_set_tooltip_text
02469     (GTK_WIDGET (window->button_add_variable), gettext ("Add variable"));
02470 window->button_remove_variable
02471     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02472                 GTK_ICON_SIZE_BUTTON);
02473 g_signal_connect
02474     (window->button_remove_variable, "clicked",
window_remove_variable, NULL);
02475 gtk_widget_set_tooltip_text
02476     (GTK_WIDGET (window->button_remove_variable), gettext ("Remove variable"));
02477 window->label_variable = (GtkLabel *) gtk_label_new (gettext ("Name"));
02478 window->entry_variable = (GtkEntry *) gtk_entry_new ();
02479 gtk_widget_set_tooltip_text

```

```

02480     (GTK_WIDGET (window->entry_variable), gettext ("Variable name"));
02481     gtk_widget_set_hexpand (GTK_WIDGET (window->entry_variable), TRUE);
02482     window->id_variable_label = g_signal_connect
02483     (window->entry_variable, "changed", window_label_variable, NULL);
02484     window->label_min = (GtkLabel *) gtk_label_new (gettext ("Minimum"));
02485     window->spin_min = (GtkSpinButton *) gtk_spin_button_new_with_range
02486     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02487     gtk_widget_set_tooltip_text
02488     (GTK_WIDGET (window->spin_min),
02489     gettext ("Minimum initial value of the variable"));
02490     window->scrolled_min
02491     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02492     gtk_container_add (GTK_CONTAINER (window->scrolled_min),
02493     GTK_WIDGET (window->spin_min));
02494     g_signal_connect (window->spin_min, "value-changed",
02495     window_rangemin_variable, NULL);
02496     window->label_max = (GtkLabel *) gtk_label_new (gettext ("Maximum"));
02497     window->spin_max = (GtkSpinButton *) gtk_spin_button_new_with_range
02498     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02499     gtk_widget_set_tooltip_text
02500     (GTK_WIDGET (window->spin_max),
02501     gettext ("Maximum initial value of the variable"));
02502     window->scrolled_max
02503     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02504     gtk_container_add (GTK_CONTAINER (window->scrolled_max),
02505     GTK_WIDGET (window->spin_max));
02506     g_signal_connect (window->spin_max, "value-changed",
02507     window_rangemax_variable, NULL);
02508     window->check_minabs = (GtkCheckButton *)
02509     gtk_check_button_new_with_mnemonic (gettext ("Absolute minimum"));
02510     g_signal_connect (window->check_minabs, "toggled", window_update, NULL);
02511     window->spin_minabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02512     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02513     gtk_widget_set_tooltip_text
02514     (GTK_WIDGET (window->spin_minabs),
02515     gettext ("Minimum allowed value of the variable"));
02516     window->scrolled_minabs
02517     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02518     gtk_container_add (GTK_CONTAINER (window->scrolled_minabs),
02519     GTK_WIDGET (window->spin_minabs));
02520     g_signal_connect (window->spin_minabs, "value-changed",
02521     window_rangeminabs_variable, NULL);
02522     window->check_maxabs = (GtkCheckButton *)
02523     gtk_check_button_new_with_mnemonic (gettext ("Absolute maximum"));
02524     g_signal_connect (window->check_maxabs, "toggled", window_update, NULL);
02525     window->spin_maxabs = (GtkSpinButton *) gtk_spin_button_new_with_range
02526     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02527     gtk_widget_set_tooltip_text
02528     (GTK_WIDGET (window->spin_maxabs),
02529     gettext ("Maximum allowed value of the variable"));
02530     window->scrolled_maxabs
02531     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02532     gtk_container_add (GTK_CONTAINER (window->scrolled_maxabs),
02533     GTK_WIDGET (window->spin_maxabs));
02534     g_signal_connect (window->spin_maxabs, "value-changed",
02535     window_rangemaxabs_variable, NULL);
02536     window->label_precision
02537     = (GtkLabel *) gtk_label_new (gettext ("Precision digits"));
02538     window->spin_precision = (GtkSpinButton *)
02539     gtk_spin_button_new_with_range (0., (gdouble) DEFAULT_PRECISION, 1.);
02540     gtk_widget_set_tooltip_text
02541     (GTK_WIDGET (window->spin_precision),
02542     gettext ("Number of precision floating point digits\n"
02543     "0 is for integer numbers"));
02544     g_signal_connect (window->spin_precision, "value-changed",
02545     window_precision_variable, NULL);
02546     window->label_sweeps = (GtkLabel *) gtk_label_new (gettext ("Sweeps number"));
02547     window->spin_sweeps
02548     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 1.e12, 1.);
02549     gtk_widget_set_tooltip_text
02550     (GTK_WIDGET (window->spin_sweeps),
02551     gettext ("Number of steps sweeping the variable"));
02552     g_signal_connect
02553     (window->spin_sweeps, "value-changed", window_update_variable, NULL);
02554     window->label_bits = (GtkLabel *) gtk_label_new (gettext ("Bits number"));
02555     window->spin_bits
02556     = (GtkSpinButton *) gtk_spin_button_new_with_range (1., 64., 1.);
02557     gtk_widget_set_tooltip_text
02558     (GTK_WIDGET (window->spin_bits),
02559     gettext ("Number of bits to encode the variable"));
02560     g_signal_connect
02561     (window->spin_bits, "value-changed", window_update_variable, NULL);
02562     window->label_step = (GtkLabel *) gtk_label_new (gettext ("Step size"));
02563     window->spin_step = (GtkSpinButton *) gtk_spin_button_new_with_range
02564     (-G_MAXDOUBLE, G_MAXDOUBLE, precision[DEFAULT_PRECISION]);
02565     gtk_widget_set_tooltip_text
02566     (GTK_WIDGET (window->spin_step),

```

```

02567     gettext ("Initial step size for the direction search method"));
02568 window->scrolled_step
02569     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02570 gtk_container_add (GTK_CONTAINER (window->scrolled_step),
02571     GTK_WIDGET (window->spin_step));
02572 g_signal_connect
02573     (window->spin_step, "value-changed", window_step_variable, NULL);
02574 window->grid_variable = (GtkGrid *) gtk_grid_new ();
02575 gtk_grid_attach (window->grid_variable,
02576     GTK_WIDGET (window->combo_variable), 0, 0, 2, 1);
02577 gtk_grid_attach (window->grid_variable,
02578     GTK_WIDGET (window->button_add_variable), 2, 0, 1, 1);
02579 gtk_grid_attach (window->grid_variable,
02580     GTK_WIDGET (window->button_remove_variable), 3, 0, 1, 1);
02581 gtk_grid_attach (window->grid_variable,
02582     GTK_WIDGET (window->label_variable), 0, 1, 1, 1);
02583 gtk_grid_attach (window->grid_variable,
02584     GTK_WIDGET (window->entry_variable), 1, 1, 3, 1);
02585 gtk_grid_attach (window->grid_variable,
02586     GTK_WIDGET (window->label_min), 0, 2, 1, 1);
02587 gtk_grid_attach (window->grid_variable,
02588     GTK_WIDGET (window->scrolled_min), 1, 2, 3, 1);
02589 gtk_grid_attach (window->grid_variable,
02590     GTK_WIDGET (window->label_max), 0, 3, 1, 1);
02591 gtk_grid_attach (window->grid_variable,
02592     GTK_WIDGET (window->scrolled_max), 1, 3, 3, 1);
02593 gtk_grid_attach (window->grid_variable,
02594     GTK_WIDGET (window->check_minabs), 0, 4, 1, 1);
02595 gtk_grid_attach (window->grid_variable,
02596     GTK_WIDGET (window->scrolled_minabs), 1, 4, 3, 1);
02597 gtk_grid_attach (window->grid_variable,
02598     GTK_WIDGET (window->check_maxabs), 0, 5, 1, 1);
02599 gtk_grid_attach (window->grid_variable,
02600     GTK_WIDGET (window->scrolled_maxabs), 1, 5, 3, 1);
02601 gtk_grid_attach (window->grid_variable,
02602     GTK_WIDGET (window->label_precision), 0, 6, 1, 1);
02603 gtk_grid_attach (window->grid_variable,
02604     GTK_WIDGET (window->spin_precision), 1, 6, 3, 1);
02605 gtk_grid_attach (window->grid_variable,
02606     GTK_WIDGET (window->label_sweeps), 0, 7, 1, 1);
02607 gtk_grid_attach (window->grid_variable,
02608     GTK_WIDGET (window->spin_sweeps), 1, 7, 3, 1);
02609 gtk_grid_attach (window->grid_variable,
02610     GTK_WIDGET (window->label_bits), 0, 8, 1, 1);
02611 gtk_grid_attach (window->grid_variable,
02612     GTK_WIDGET (window->spin_bits), 1, 8, 3, 1);
02613 gtk_grid_attach (window->grid_variable,
02614     GTK_WIDGET (window->label_step), 0, 9, 1, 1);
02615 gtk_grid_attach (window->grid_variable,
02616     GTK_WIDGET (window->scrolled_step), 1, 9, 3, 1);
02617 window->frame_variable = (GtkFrame *) gtk_frame_new (gettext ("Variable"));
02618 gtk_container_add (GTK_CONTAINER (window->frame_variable),
02619     GTK_WIDGET (window->grid_variable));
02620
02621 // Creating the experiment widgets
02622 window->combo_experiment = (GtkComboBoxText *) gtk_combo_box_text_new ();
02623 gtk_widget_set_tooltip_text (GTK_WIDGET (window->combo_experiment),
02624     gettext ("Experiment selector"));
02625 window->id_experiment = g_signal_connect
02626     (window->combo_experiment, "changed", window_set_experiment, NULL)
02627 ;
02628 window->button_add_experiment
02629     = (GtkButton *) gtk_button_new_from_icon_name ("list-add",
02630     GTK_ICON_SIZE_BUTTON);
02631 g_signal_connect
02632     (window->button_add_experiment, "clicked",
02633     window_add_experiment, NULL);
02634 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_add_experiment),
02635     gettext ("Add experiment"));
02636 window->button_remove_experiment
02637     = (GtkButton *) gtk_button_new_from_icon_name ("list-remove",
02638     GTK_ICON_SIZE_BUTTON);
02639 g_signal_connect (window->button_remove_experiment, "clicked",
02640     window_remove_experiment, NULL);
02641 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_remove_experiment),
02642     gettext ("Remove experiment"));
02643 window->label_experiment
02644     = (GtkLabel *) gtk_label_new (gettext ("Experimental data file"));
02645 window->button_experiment = (GtkFileChooserButton *)
02646     gtk_file_chooser_button_new (gettext ("Experimental data file"),
02647     GTK_FILE_CHOOSER_ACTION_OPEN);
02648 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_experiment),
02649     gettext ("Experimental data file"));
02650 window->id_experiment_name
02651     = g_signal_connect (window->button_experiment, "selection-changed",
02652     window_name_experiment, NULL);
02653 gtk_widget_set_hexpand (GTK_WIDGET (window->button_experiment), TRUE);

```

```

02652 window->label_weight = (GtkLabel *) gtk_label_new (gettext ("Weight"));
02653 window->spin_weight
02654     = (GtkSpinButton *) gtk_spin_button_new_with_range (0., 1., 0.001);
02655 gtk_widget_set_tooltip_text
02656     (GTK_WIDGET (window->spin_weight),
02657      gettext ("Weight factor to build the objective function"));
02658 g_signal_connect
02659     (window->spin_weight, "value-changed", window_weight_experiment,
NULL);
02660 window->grid_experiment = (GtkGrid *) gtk_grid_new ();
02661 gtk_grid_attach (window->grid_experiment,
02662     GTK_WIDGET (window->combo_experiment), 0, 0, 2, 1);
02663 gtk_grid_attach (window->grid_experiment,
02664     GTK_WIDGET (window->button_add_experiment), 2, 0, 1, 1);
02665 gtk_grid_attach (window->grid_experiment,
02666     GTK_WIDGET (window->button_remove_experiment), 3, 0, 1, 1);
02667 gtk_grid_attach (window->grid_experiment,
02668     GTK_WIDGET (window->label_experiment), 0, 1, 1, 1);
02669 gtk_grid_attach (window->grid_experiment,
02670     GTK_WIDGET (window->button_experiment), 1, 1, 3, 1);
02671 gtk_grid_attach (window->grid_experiment,
02672     GTK_WIDGET (window->label_weight), 0, 2, 1, 1);
02673 gtk_grid_attach (window->grid_experiment,
02674     GTK_WIDGET (window->spin_weight), 1, 2, 3, 1);
02675 for (i = 0; i < MAX_NINPITS; ++i)
02676 {
02677     snprintf (buffer3, 64, "%s %u", gettext ("Input template"), i + 1);
02678     window->check_template[i] = (GtkCheckButton *)
02679         gtk_check_button_new_with_label (buffer3);
02680     window->id_template[i]
02681         = g_signal_connect (window->check_template[i], "toggled",
02682             window_inputs_experiment, NULL);
02683     gtk_grid_attach (window->grid_experiment,
02684         GTK_WIDGET (window->check_template[i]), 0, 3 + i, 1, 1);
02685     window->button_template[i] = (GtkFileChooserButton *)
02686         gtk_file_chooser_button_new (gettext ("Input template"),
02687             GTK_FILE_CHOOSER_ACTION_OPEN);
02688     gtk_widget_set_tooltip_text
02689         (GTK_WIDGET (window->button_template[i]),
02690          gettext ("Experimental input template file"));
02691     window->id_input[i]
02692         = g_signal_connect_swapped (window->button_template[i],
02693             "selection-changed",
02694             (void (*)(void *)) window_template_experiment,
02695             (void *) (size_t) i);
02696     gtk_grid_attach (window->grid_experiment,
02697         GTK_WIDGET (window->button_template[i]), 1, 3 + i, 3, 1);
02698 }
02699 window->frame_experiment
02700     = (GtkFrame *) gtk_frame_new (gettext ("Experiment"));
02701 gtk_container_add (GTK_CONTAINER (window->frame_experiment),
02702     GTK_WIDGET (window->grid_experiment));
02703
02704 // Creating the error norm widgets
02705 window->frame_norm = (GtkFrame *) gtk_frame_new (gettext ("Error norm"));
02706 window->grid_norm = (GtkGrid *) gtk_grid_new ();
02707 gtk_container_add (GTK_CONTAINER (window->frame_norm),
02708     GTK_WIDGET (window->grid_norm));
02709 window->button_norm[0] = (GtkRadioButton *)
02710     gtk_radio_button_new_with_mnemonic (NULL, label_norm[0]);
02711 gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[0]),
02712     tip_norm[0]);
02713 gtk_grid_attach (window->grid_norm,
02714     GTK_WIDGET (window->button_norm[0]), 0, 0, 1, 1);
02715 g_signal_connect (window->button_norm[0], "clicked", window_update, NULL);
02716 for (i = 0; ++i < NNORMS;)
02717 {
02718     window->button_norm[i] = (GtkRadioButton *)
02719         gtk_radio_button_new_with_mnemonic
02720         (gtk_radio_button_get_group (window->button_norm[0]), label_norm[i]);
02721     gtk_widget_set_tooltip_text (GTK_WIDGET (window->button_norm[i]),
02722         tip_norm[i]);
02723     gtk_grid_attach (window->grid_norm,
02724         GTK_WIDGET (window->button_norm[i]), 0, i, 1, 1);
02725     g_signal_connect (window->button_norm[i], "clicked",
02726         window_update, NULL);
02726 }
02727 window->label_p = (GtkLabel *) gtk_label_new (gettext ("P parameter"));
02728 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->label_p), 1, 1, 1, 1);
02729 window->spin_p = (GtkSpinButton *)
02730     gtk_spin_button_new_with_range (-G_MAXDOUBLE, G_MAXDOUBLE, 0.01);
02731 gtk_widget_set_tooltip_text
02732     (GTK_WIDGET (window->spin_p), gettext ("P parameter for the P error norm"));
02733 window->scrolled_p
02734     = (GtkScrolledWindow *) gtk_scrolled_window_new (NULL, NULL);
02735 gtk_container_add (GTK_CONTAINER (window->scrolled_p),
02736     GTK_WIDGET (window->spin_p));

```

```

02737 gtk_widget_set_hexpand (GTK_WIDGET (window->scrolled_p), TRUE);
02738 gtk_widget_set_halign (GTK_WIDGET (window->scrolled_p), GTK_ALIGN_FILL);
02739 gtk_grid_attach (window->grid_norm, GTK_WIDGET (window->scrolled_p),
02740 1, 2, 1, 2);
02741
02742 // Creating the grid and attaching the widgets to the grid
02743 window->grid = (GtkGrid *) gtk_grid_new ();
02744 gtk_grid_attach (window->grid, GTK_WIDGET (window->bar_buttons), 0, 0, 3, 1);
02745 gtk_grid_attach (window->grid, GTK_WIDGET (window->grid_files), 0, 1, 1, 1);
02746 gtk_grid_attach (window->grid,
02747 GTK_WIDGET (window->frame_algorithm), 0, 2, 1, 1);
02748 gtk_grid_attach (window->grid,
02749 GTK_WIDGET (window->frame_variable), 1, 2, 1, 1);
02750 gtk_grid_attach (window->grid,
02751 GTK_WIDGET (window->frame_experiment), 2, 2, 1, 1);
02752 gtk_grid_attach (window->grid, GTK_WIDGET (window->frame_norm), 1, 1, 2, 1);
02753 gtk_container_add (GTK_CONTAINER (window->window), GTK_WIDGET (window->
grid));
02754
02755 // Setting the window logo
02756 window->logo = gdk_pixbuf_new_from_xpm_data (logo);
02757 gtk_window_set_icon (window->window, window->logo);
02758
02759 // Showing the window
02760 gtk_widget_show_all (GTK_WIDGET (window->window));
02761
02762 // In GTK+ 3.16 and 3.18 the default scrolled size is wrong
02763 #if GTK_MINOR_VERSION >= 16
02764 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_min), -1, 40);
02765 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_max), -1, 40);
02766 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_minabs), -1, 40);
02767 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_maxabs), -1, 40);
02768 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_step), -1, 40);
02769 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_p), -1, 40);
02770 gtk_widget_set_size_request (GTK_WIDGET (window->scrolled_threshold), -1, 40);
02771 #endif
02772
02773 // Reading initial example
02774 input_new ();
02775 buffer2 = g_get_current_dir ();
02776 buffer = g_build_filename (buffer2, "..", "tests", "test1", INPUT_FILE, NULL);
02777 g_free (buffer2);
02778 printf ("Initial file=%s\n", buffer);
02779 window_read (buffer);
02780 g_free (buffer);
02781
02782 #if DEBUG_INTERFACE
02783 fprintf (stderr, "window_new: start\n");
02784 #endif
02785 }

```

5.13 interface.h File Reference

Header file to define the graphical interface functions.

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

Data Structures

- struct [Options](#)
Struct to define the options dialog.
- struct [Running](#)
Struct to define the running dialog.
- struct [Window](#)
Struct to define the main window.

Macros

- #define [MAX_LENGTH](#) (DEFAULT_PRECISION + 8)
Max length of texts allowed in GtkSpinButtons.

Functions

- static GtkWidget * **gtk_button_new_from_icon_name** (const char *name, GtkIconSize size)
- unsigned int **gtk_array_get_active** (GtkRadioButton *array[], unsigned int n)
Function to get the active GtkRadioButton.
- void **input_save** (char *filename)
Function to save the input file.
- void **options_new** ()
Function to open the options dialog.
- void **running_new** ()
Function to open the running dialog.
- unsigned int **window_get_algorithm** ()
Function to get the stochastic algorithm number.
- unsigned int **window_get_direction** ()
Function to get the direction search method number.
- unsigned int **window_get_norm** ()
Function to get the norm method number.
- void **window_save_direction** ()
Function to save the direction search method data in the input file.
- int **window_save** ()
Function to save the input file.
- void **window_run** ()
Function to run a optimization.
- void **window_help** ()
Function to show a help dialog.
- void **window_update_direction** ()
Function to update direction search method widgets view in the main window.
- void **window_update** ()
Function to update the main window view.
- void **window_set_algorithm** ()
Function to avoid memory errors changing the algorithm.
- void **window_set_experiment** ()
Function to set the experiment data in the main window.
- void **window_remove_experiment** ()
Function to remove an experiment in the main window.
- void **window_add_experiment** ()
Function to add an experiment in the main window.
- void **window_name_experiment** ()
Function to set the experiment name in the main window.
- void **window_weight_experiment** ()
Function to update the experiment weight in the main window.
- void **window_inputs_experiment** ()
Function to update the experiment input templates number in the main window.
- void **window_template_experiment** (void *data)
Function to update the experiment i-th input template in the main window.
- void **window_set_variable** ()
Function to set the variable data in the main window.
- void **window_remove_variable** ()
Function to remove a variable in the main window.
- void **window_add_variable** ()
Function to add a variable in the main window.

- void [window_label_variable](#) ()
Function to set the variable label in the main window.
- void [window_precision_variable](#) ()
Function to update the variable precision in the main window.
- void [window_rangemin_variable](#) ()
Function to update the variable rangemin in the main window.
- void [window_rangemax_variable](#) ()
Function to update the variable rangemax in the main window.
- void [window_rangeminabs_variable](#) ()
Function to update the variable rangeminabs in the main window.
- void [window_rangemaxabs_variable](#) ()
Function to update the variable rangemaxabs in the main window.
- void [window_update_variable](#) ()
Function to update the variable data in the main window.
- int [window_read](#) (char *filename)
Function to read the input data of a file.
- void [window_open](#) ()
Function to open the input data.
- void [window_new](#) ()
Function to open the main window.

Variables

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

5.13.1 Detailed Description

Header file to define the graphical interface functions.

Authors

Javier Burquete.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.13.2 Function Documentation

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

Function to get the active GtkWidget.

Parameters

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

Returns

Active GtkRadioButton.

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

```

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

5.13.2.2 void input_save (char * filename)

Function to save the input file.

Parameters

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

Definition at line 577 of file [interface.c](#).

```

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



```
00613     g_object_unref (generator);
00614 }
00615
00616 #if DEBUG_INTERFACE
00617 fprintf (stderr, "input_save: end\n");
00618 #endif
00619 }
```

Here is the call graph for this function:

5.13.2.3 unsigned int window_get_algorithm ()

Function to get the stochastic algorithm number.

Returns

Stochastic algorithm number.

Definition at line 728 of file [interface.c](#).

```
00729 {
00730     unsigned int i;
00731     #if DEBUG_INTERFACE
00732     fprintf (stderr, "window_get_algorithm: start\n");
00733     #endif
00734     i = gtk_array_get_active (window->button_algorithm,
00735                             NALGORITHMS);
00736     #if DEBUG_INTERFACE
00737     fprintf (stderr, "window_get_algorithm: %u\n", i);
00738     fprintf (stderr, "window_get_algorithm: end\n");
00739     #endif
00740     return i;
00741 }
```

Here is the call graph for this function:

5.13.2.4 unsigned int window_get_direction ()

Function to get the direction search method number.

Returns

Direction search method number.

Definition at line 748 of file [interface.c](#).

```
00749 {
00750     unsigned int i;
00751     #if DEBUG_INTERFACE
00752     fprintf (stderr, "window_get_direction: start\n");
00753     #endif
00754     i = gtk_array_get_active (window->button_direction,
00755                             NDIRECTIONS);
00756     #if DEBUG_INTERFACE
00757     fprintf (stderr, "window_get_direction: %u\n", i);
00758     fprintf (stderr, "window_get_direction: end\n");
00759     #endif
00760     return i;
00761 }
```

Here is the call graph for this function:

5.13.2.5 unsigned int window_get_norm ()

Function to get the norm method number.

Returns

Norm method number.

Definition at line 768 of file [interface.c](#).

```
00769 {
00770     unsigned int i;
00771     #if DEBUG_INTERFACE
00772     fprintf (stderr, "window_get_norm: start\n");
00773     #endif
00774     i = gtk_array_get_active (window->button_norm,
00775                             NNORMS);
00776     #if DEBUG_INTERFACE
00777     fprintf (stderr, "window_get_norm: %u\n", i);
00778     fprintf (stderr, "window_get_norm: end\n");
00779     #endif
00780     return i;
00781 }
```

Here is the call graph for this function:

5.13.2.6 int window_read (char * filename)

Function to read the input data of a file.

Parameters

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

Returns

1 on succes, 0 on error.

Definition at line 1876 of file [interface.c](#).

```
01877 {
01878     unsigned int i;
01879     char *buffer;
01880     #if DEBUG_INTERFACE
01881     fprintf (stderr, "window_read: start\n");
01882     #endif
01883     // Reading new input file
01884     input_free ();
01885     if (!input_open (filename))
01886     {
01887         #if DEBUG_INTERFACE
01888         fprintf (stderr, "window_read: end\n");
01889         #endif
01890         return 0;
01891     }
01892     // Setting GTK+ widgets data
01893     gtk_entry_set_text (window->entry_result, input->result);
01894     gtk_entry_set_text (window->entry_variables, input->
01895                       variables);
01896     buffer = g_build_filename (input->directory, input->
```

```

    simulator, NULL);
01898   gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01899       (window->button_simulator), buffer);
01900   g_free (buffer);
01901   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->check_evaluator),
01902       (size_t) input->evaluator);
01903   if (input->evaluator)
01904   {
01905       buffer = g_build_filename (input->directory, input->
evaluator, NULL);
01906       gtk_file_chooser_set_filename (GTK_FILE_CHOOSER
01907           (window->button_evaluator), buffer);
01908       g_free (buffer);
01909   }
01910   gtk_toggle_button_set_active
01911       (GTK_TOGGLE_BUTTON (window->button_algorithm[input->
algorithm]), TRUE);
01912   switch (input->algorithm)
01913   {
01914       case ALGORITHM_MONTE_CARLO:
01915           gtk_spin_button_set_value (window->spin_simulations,
01916               (gdouble) input->nsimulations);
01917       case ALGORITHM_SWEEP:
01918           gtk_spin_button_set_value (window->spin_iterations,
01919               (gdouble) input->niterations);
01920           gtk_spin_button_set_value (window->spin_bests, (gdouble)
input->nbest);
01921           gtk_spin_button_set_value (window->spin_tolerance,
input->tolerance);
01922           gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (window->
check_direction),
input->nsteps);
01923           if (input->nsteps)
01924           {
01925               gtk_toggle_button_set_active
01926                   (GTK_TOGGLE_BUTTON (window->button_direction
[input->direction]), TRUE);
01927               gtk_spin_button_set_value (window->spin_steps,
01928                   (gdouble) input->nsteps);
01929               gtk_spin_button_set_value (window->spin_relaxation,
01930                   (gdouble) input->relaxation);
01931               switch (input->direction)
01932               {
01933                   case DIRECTION_METHOD_RANDOM:
01934                       gtk_spin_button_set_value (window->spin_estimates,
01935                           (gdouble) input->nestimates);
01936                   }
01937               }
01938           }
01939           break;
01940       default:
01941           gtk_spin_button_set_value (window->spin_population,
01942               (gdouble) input->nsimulations);
01943           gtk_spin_button_set_value (window->spin_generations,
01944               (gdouble) input->niterations);
01945           gtk_spin_button_set_value (window->spin_mutation, input->
mutation_ratio);
01946           gtk_spin_button_set_value (window->spin_reproduction,
input->reproduction_ratio);
01947           gtk_spin_button_set_value (window->spin_adaptation,
input->adaptation_ratio);
01948           }
01949       }
01950       gtk_toggle_button_set_active
01951           (GTK_TOGGLE_BUTTON (window->button_norm[input->norm]), TRUE);
01952       gtk_spin_button_set_value (window->spin_p, input->p);
01953       gtk_spin_button_set_value (window->spin_threshold, input->
threshold);
01954       g_signal_handler_block (window->combo_experiment, window->
id_experiment);
01955       g_signal_handler_block (window->button_experiment,
window->id_experiment_name);
01956       gtk_combo_box_text_remove_all (window->combo_experiment);
01957       for (i = 0; i < input->nexperiments; ++i)
01958       {
01959           gtk_combo_box_text_append_text (window->combo_experiment,
input->experiment[i].name);
01960       }
01961       g_signal_handler_unblock
01962           (window->button_experiment, window->
id_experiment_name);
01963       g_signal_handler_unblock (window->combo_experiment,
window->id_experiment);
01964       gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_experiment), 0);
01965       g_signal_handler_block (window->combo_variable, window->
id_variable);
01966       g_signal_handler_block (window->entry_variable, window->
id_variable_label);
01967       gtk_combo_box_text_remove_all (window->combo_variable);
01968       for (i = 0; i < input->nvariables; ++i)
01969       {
01970           gtk_combo_box_text_append_text (window->combo_variable,

```

```

01972                                     input->variable[i].name);
01973     g_signal_handler_unblock (window->entry_variable, window->
    id_variable_label);
01974     g_signal_handler_unblock (window->combo_variable, window->
    id_variable);
01975     gtk_combo_box_set_active (GTK_COMBO_BOX (window->combo_variable), 0);
01976     window_set_variable ();
01977     window_update ();
01978
01979 #if DEBUG_INTERFACE
01980     fprintf (stderr, "window_read: end\n");
01981 #endif
01982     return 1;
01983 }

```

Here is the call graph for this function:

5.13.2.7 int window_save ()

Function to save the input file.

Returns

1 on OK, 0 on Cancel.

Definition at line 821 of file [interface.c](#).

```

00822 {
00823     GtkFileChooserDialog *dlg;
00824     GtkFileFilter *filter1, *filter2;
00825     char *buffer;
00826
00827 #if DEBUG_INTERFACE
00828     fprintf (stderr, "window_save: start\n");
00829 #endif
00830
00831     // Opening the saving dialog
00832     dlg = (GtkFileChooserDialog *)
00833         gtk_file_chooser_dialog_new (gettext ("Save file"),
00834                                     window->window,
00835                                     GTK_FILE_CHOOSER_ACTION_SAVE,
00836                                     gettext ("_Cancel"),
00837                                     GTK_RESPONSE_CANCEL,
00838                                     gettext ("_OK"), GTK_RESPONSE_OK, NULL);
00839     gtk_file_chooser_set_do_overwrite_confirmation (GTK_FILE_CHOOSER (dlg), TRUE);
00840     buffer = g_build_filename (input->directory, input->name, NULL);
00841     gtk_file_chooser_set_filename (GTK_FILE_CHOOSER (dlg), buffer);
00842     g_free (buffer);
00843
00844     // Adding XML filter
00845     filter1 = (GtkFileFilter *) gtk_file_filter_new ();
00846     gtk_file_filter_set_name (filter1, "XML");
00847     gtk_file_filter_add_pattern (filter1, "*.xml");
00848     gtk_file_filter_add_pattern (filter1, "*.XML");
00849     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter1);
00850
00851     // Adding JSON filter
00852     filter2 = (GtkFileFilter *) gtk_file_filter_new ();
00853     gtk_file_filter_set_name (filter2, "JSON");
00854     gtk_file_filter_add_pattern (filter2, "*.json");
00855     gtk_file_filter_add_pattern (filter2, "*.JSON");
00856     gtk_file_filter_add_pattern (filter2, "*.js");
00857     gtk_file_filter_add_pattern (filter2, "*.JS");
00858     gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dlg), filter2);
00859
00860     if (input->type == INPUT_TYPE_XML)
00861         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter1);
00862     else
00863         gtk_file_chooser_set_filter (GTK_FILE_CHOOSER (dlg), filter2);
00864
00865     // If OK response then saving
00866     if (gtk_dialog_run (GTK_DIALOG (dlg)) == GTK_RESPONSE_OK)
00867     {
00868         // Setting input file type
00869         filter1 = gtk_file_chooser_get_filter (GTK_FILE_CHOOSER (dlg));

```

```

00870     buffer = (char *) gtk_file_filter_get_name (filter1);
00871     if (!strcmp (buffer, "XML"))
00872         input->type = INPUT_TYPE_XML;
00873     else
00874         input->type = INPUT_TYPE_JSON;
00875
00876     // Adding properties to the root XML node
00877     input->simulator = gtk_file_chooser_get_filename
00878         (GTK_FILE_CHOOSER (window->button_simulator));
00879     if (gtk_toggle_button_get_active
00880         (GTK_TOGGLE_BUTTON (window->check_evaluator)))
00881         input->evaluator = gtk_file_chooser_get_filename
00882             (GTK_FILE_CHOOSER (window->button_evaluator));
00883     else
00884         input->evaluator = NULL;
00885     if (input->type == INPUT_TYPE_XML)
00886     {
00887         input->result
00888             = (char *) xmlStrdup ((const xmlChar *)
00889                                     gtk_entry_get_text (window->entry_result));
00890         input->variables
00891             = (char *) xmlStrdup ((const xmlChar *)
00892                                     gtk_entry_get_text (window->
00893 entry_variables));
00894     }
00895     else
00896     {
00897         input->result = g_strdup (gtk_entry_get_text (window->
00898 entry_result));
00899         input->variables
00900             = g_strdup (gtk_entry_get_text (window->entry_variables));
00901     }
00902     // Setting the algorithm
00903     switch (window_get_algorithm ())
00904     {
00905         case ALGORITHM_MONTE_CARLO:
00906             input->algorithm = ALGORITHM_MONTE_CARLO;
00907             input->nsimulations
00908                 = gtk_spin_button_get_value_as_int (window->spin_simulations);
00909             input->niterations
00910                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00911             input->tolerance = gtk_spin_button_get_value (window->
00912 spin_tolerance);
00913             input->nbest = gtk_spin_button_get_value_as_int (window->
00914 spin_bests);
00915             window_save_direction ();
00916             break;
00917         case ALGORITHM_SWEEP:
00918             input->algorithm = ALGORITHM_SWEEP;
00919             input->niterations
00920                 = gtk_spin_button_get_value_as_int (window->spin_iterations);
00921             input->tolerance = gtk_spin_button_get_value (window->
00922 spin_tolerance);
00923             input->nbest = gtk_spin_button_get_value_as_int (window->
00924 spin_bests);
00925             window_save_direction ();
00926             break;
00927         default:
00928             input->algorithm = ALGORITHM_GENETIC;
00929             input->nsimulations
00930                 = gtk_spin_button_get_value_as_int (window->spin_population);
00931             input->niterations
00932                 = gtk_spin_button_get_value_as_int (window->spin_generations);
00933             input->mutation_ratio
00934                 = gtk_spin_button_get_value (window->spin_mutation);
00935             input->reproduction_ratio
00936                 = gtk_spin_button_get_value (window->spin_reproduction);
00937             input->adaptation_ratio
00938                 = gtk_spin_button_get_value (window->spin_adaptation);
00939             break;
00940     }
00941     input->norm = window_get_norm ();
00942     input->p = gtk_spin_button_get_value (window->spin_p);
00943     input->threshold = gtk_spin_button_get_value (window->
00944 spin_threshold);
00945
00946     // Saving the XML file
00947     buffer = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dlg));
00948     input_save (buffer);
00949
00950     // Closing and freeing memory
00951     g_free (buffer);
00952     gtk_widget_destroy (GTK_WIDGET (dlg));
00953 #if DEBUG_INTERFACE
00954     fprintf (stderr, "window_save: end\n");
00955 #endif

```

```

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

```

Here is the call graph for this function:

5.13.2.8 void window_template_experiment (void * data)

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

Parameters

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

Definition at line 1520 of file [interface.c](#).

```

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

```

5.14 interface.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.

```

```

00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
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     GtkSpinner *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 *button_simulator;
00072     GtkWidget *check_evaluator;
00073     GtkWidget *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 *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 *button_algorithm[NALGORITHMS];
00087     GtkWidget *label_simulations;
00088     GtkWidget *spin_simulations;
00089     GtkWidget *label_iterations;
00090     GtkWidget *spin_iterations;
00091     GtkWidget *label_tolerance;
00092     GtkWidget *spin_tolerance;
00093     GtkWidget *label_bests;
00094     GtkWidget *spin_bests;
00095     GtkWidget *label_population;
00096     GtkWidget *spin_population;
00097     GtkWidget *label_generations;
00098     GtkWidget *spin_generations;
00099     GtkWidget *label_mutation;
00100     GtkWidget *spin_mutation;
00101     GtkWidget *label_reproduction;
00102     GtkWidget *spin_reproduction;
00103     GtkWidget *label_adaptation;
00104     GtkWidget *spin_adaptation;
00105     GtkWidget *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   GtkWidget *button_add_variable;
00162   GtkWidget *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   GtkWidget *button_add_experiment;
00190   GtkWidget *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_MAJOR_VERSION <= 3 && GTK_MINOR_VERSION < 10
00224 static inline GtkWidget *
00225 gtk_button_new_from_icon_name (const char *name, GtkIconSize size)
00226 {
00227     GtkWidget *button;
00228     GtkWidget *image;
00229     button = (GtkWidget *) gtk_button_new ();
00230     image = (GtkWidget *) gtk_image_new_from_icon_name (name, size);
00231     gtk_button_set_image (button, GTK_WIDGET (image));
00232     return button;
00233 }
00234 #endif
00235
00236 // Public functions
00237 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00238 void input_save (char *filename);

```



```

00239 void options_new ();
00240 void running_new ();
00241 unsigned int window_get_algorithm ();
00242 unsigned int window_get_direction ();
00243 unsigned int window_get_norm ();
00244 void window_save_direction ();
00245 int window_save ();
00246 void window_run ();
00247 void window_help ();
00248 void window_update_direction ();
00249 void window_update ();
00250 void window_set_algorithm ();
00251 void window_set_experiment ();
00252 void window_remove_experiment ();
00253 void window_add_experiment ();
00254 void window_name_experiment ();
00255 void window_weight_experiment ();
00256 void window_inputs_experiment ();
00257 void window_template_experiment (void *data);
00258 void window_set_variable ();
00259 void window_remove_variable ();
00260 void window_add_variable ();
00261 void window_label_variable ();
00262 void window_precision_variable ();
00263 void window_rangemin_variable ();
00264 void window_rangemax_variable ();
00265 void window_rangeminabs_variable ();
00266 void window_rangemaxabs_variable ();
00267 void window_update_variable ();
00268 int window_read (char *filename);
00269 void window_open ();
00270 void window_new ();
00271
00272 #endif

```

5.15 main.c File Reference

Main source file.

```

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

```

Include dependency graph for main.c:

Macros

- #define **_GNU_SOURCE**
 - #define **DEBUG_MAIN** 0
- Macro to debug main functions.*

Functions

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

5.15.1 Detailed Description

Main source file.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.16 main.c

```

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

```

```

00056 #if HAVE_GTK
00057 #include <gio/gio.h>
00058 #include <gtk/gtk.h>
00059 #endif
00060 #include "genetic/genetic.h"
00061 #include "utils.h"
00062 #include "experiment.h"
00063 #include "variable.h"
00064 #include "input.h"
00065 #include "optimize.h"
00066 #if HAVE_GTK
00067 #include "interface.h"
00068 #endif
00069
00070 #define DEBUG_MAIN 0
00071
00072
00081 int
00082 main (int argn, char **argc)
00083 {
00084     #if HAVE_GTK
00085         char *buffer;
00086     #endif
00087
00088     // Starting pseudo-random numbers generator
00089     #if DEBUG_MAIN
00090         fprintf (stderr, "main: starting pseudo-random numbers generator\n");
00091     #endif
00092     optimize->rng = gsl_rng_alloc (gsl_rng_taus2);
00093
00094     // Allowing spaces in the XML data file
00095     #if DEBUG_MAIN
00096         fprintf (stderr, "main: allowing spaces in the XML data file\n");
00097     #endif
00098     xmlKeepBlanksDefault (0);
00099
00100     // Starting MPI
00101     #if HAVE_MPI
00102     #if DEBUG_MAIN
00103         fprintf (stderr, "main: starting MPI\n");
00104     #endif
00105         MPI_Init (&argn, &argc);
00106         MPI_Comm_size (MPI_COMM_WORLD, &ntasks);
00107         MPI_Comm_rank (MPI_COMM_WORLD, &optimize->mpi_rank);
00108         printf ("rank=%d tasks=%d\n", optimize->mpi_rank, ntasks);
00109     #else
00110         ntasks = 1;
00111     #endif
00112
00113     // Resetting result and variables file names
00114     #if DEBUG_MAIN
00115         fprintf (stderr, "main: resetting result and variables file names\n");
00116     #endif
00117     input->result = input->variables = NULL;
00118
00119     #if HAVE_GTK
00120
00121     // Getting threads number and pseudo-random numbers generator seed
00122     nthreads_direction = nthreads = cores_number ();
00123     optimize->seed = DEFAULT_RANDOM_SEED;
00124
00125     // Setting local language and international floating point numbers notation
00126     setlocale (LC_ALL, "");
00127     setlocale (LC_NUMERIC, "C");
00128     window->application_directory = g_get_current_dir ();
00129     buffer = g_build_filename (window->application_directory,
00130                               LOCALE_DIR, NULL);
00130     bindtextdomain (PROGRAM_INTERFACE, buffer);
00131     bind_textdomain_codeset (PROGRAM_INTERFACE, "UTF-8");
00132     textdomain (PROGRAM_INTERFACE);
00133
00134     // Initing GTK+
00135     gtk_disable_setlocale ();
00136     gtk_init (&argn, &argc);
00137
00138     // Opening the main window
00139     window_new ();
00140     gtk_main ();
00141
00142     // Freeing memory
00143     input_free ();
00144     g_free (buffer);
00145     gtk_widget_destroy (GTK_WIDGET (window->window));
00146     g_free (window->application_directory);
00147
00148     #else
00149

```

```

00150 // Checking syntax
00151 if (argn < 2)
00152 {
00153     printf ("The syntax is:\n"
00154             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00155             "[variables_file]\n");
00156     return 1;
00157 }
00158
00159 // Getting threads number and pseudo-random numbers generator seed
00160 #if DEBUG_MAIN
00161 fprintf (stderr, "main: getting threads number and pseudo-random numbers "
00162          "generator seed\n");
00163 #endif
00164 nthreads_direction = nthreads = cores_number ();
00165 optimize->seed = DEFAULT_RANDOM_SEED;
00166 if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00167 {
00168     nthreads_direction = nthreads = atoi (argc[2]);
00169     if (!nthreads)
00170     {
00171         printf ("Bad threads number\n");
00172         return 2;
00173     }
00174     argc += 2;
00175     argn -= 2;
00176     if (argn > 2 && !strcmp (argc[1], "-seed"))
00177     {
00178         optimize->seed = atoi (argc[2]);
00179         argc += 2;
00180         argn -= 2;
00181     }
00182 }
00183 else if (argn > 2 && !strcmp (argc[1], "-seed"))
00184 {
00185     optimize->seed = atoi (argc[2]);
00186     argc += 2;
00187     argn -= 2;
00188     if (argn > 2 && !strcmp (argc[1], "-nthreads"))
00189     {
00190         nthreads_direction = nthreads = atoi (argc[2]);
00191         if (!nthreads)
00192         {
00193             printf ("Bad threads number\n");
00194             return 2;
00195         }
00196         argc += 2;
00197         argn -= 2;
00198     }
00199 }
00200 printf ("nthreads=%u\n", nthreads);
00201 printf ("seed=%lu\n", optimize->seed);
00202
00203 // Checking arguments
00204 #if DEBUG_MAIN
00205 fprintf (stderr, "main: checking arguments\n");
00206 #endif
00207 if (argn > 4 || argn < 2)
00208 {
00209     printf ("The syntax is:\n"
00210             "../mpcotoolbin [-nthreads x] [-seed s] data_file [result_file] "
00211             "[variables_file]\n");
00212     return 1;
00213 }
00214 if (argn > 2)
00215     input->result = (char *) xmlStrdup ((xmlChar *) argc[2]);
00216 if (argn == 4)
00217     input->variables = (char *) xmlStrdup ((xmlChar *) argc[3]);
00218
00219 // Making optimization
00220 #if DEBUG_MAIN
00221 fprintf (stderr, "main: making optimization\n");
00222 #endif
00223 if (input_open (argc[1]))
00224     optimize_open ();
00225
00226 // Freeing memory
00227 #if DEBUG_MAIN
00228 fprintf (stderr, "main: freeing memory and closing\n");
00229 #endif
00230 optimize_free ();
00231
00232 #endif
00233
00234 // Closing MPI
00235 #if HAVE_MPI
00236 MPI_Finalize ();

```

```

00237 #endif
00238
00239 // Freeing memory
00240 gsl_rng_free (optimize->rng);
00241
00242 // Closing
00243 return 0;
00244 }

```

5.17 optimize.c File Reference

Source file to define the optimization functions.

```

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

```

Include dependency graph for optimize.c:

Macros

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

Functions

- void **optimize_input** (unsigned int simulation, char *input, GMappedFile *template)
Function to write the simulation input file.
- double **optimize_parse** (unsigned int simulation, unsigned int experiment)
Function to parse input files, simulating and calculating the \ objective function.
- double **optimize_norm_euclidian** (unsigned int simulation)
Function to calculate the Euclidian error norm.
- double **optimize_norm_maximum** (unsigned int simulation)
Function to calculate the maximum error norm.
- double **optimize_norm_p** (unsigned int simulation)

- Function to calculate the P error norm.*

 - double `optimize_norm_taxicab` (unsigned int simulation)
- Function to calculate the taxicab error norm.*

 - void `optimize_print` ()
- Function to print the results.*

 - void `optimize_save_variables` (unsigned int simulation, double error)
- Function to save in a file the variables and the error.*

 - void `optimize_best` (unsigned int simulation, double value)
- Function to save the best simulations.*

 - void `optimize_sequential` ()
- Function to optimize sequentially.*

 - void * `optimize_thread` (`ParallelData` *data)
- Function to optimize on a thread.*

 - void `optimize_merge` (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
- Function to merge the 2 optimization results.*

 - void `optimize_synchronise` ()
- Function to synchronise the optimization results of MPI tasks.*

 - void `optimize_sweep` ()
- Function to optimize with the sweep algorithm.*

 - void `optimize_MonteCarlo` ()
- Function to optimize with the Monte-Carlo algorithm.*

 - void `optimize_best_direction` (unsigned int simulation, double value)
- Function to save the best simulation in a direction search method.*

 - void `optimize_direction_sequential` (unsigned int simulation)
- Function to estimate the direction search sequentially.*

 - void * `optimize_direction_thread` (`ParallelData` *data)
- Function to estimate the direction search on a thread.*

 - double `optimize_estimate_direction_random` (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the direction search vector.*

 - double `optimize_estimate_direction_coordinates` (unsigned int variable, unsigned int estimate)
- Function to estimate a component of the direction search vector.*

 - void `optimize_step_direction` (unsigned int simulation)
- Function to do a step of the direction search method.*

 - void `optimize_direction` ()
- Function to optimize with a direction search method.*

 - double `optimize_genetic_objective` (Entity *entity)
- Function to calculate the objective function of an entity.*

 - void `optimize_genetic` ()
- Function to optimize with the genetic algorithm.*

 - void `optimize_save_old` ()
- Function to save the best results on iterative methods.*

 - void `optimize_merge_old` ()
- Function to merge the best results with the previous step best results on iterative methods.*

 - void `optimize_refine` ()
- Function to refine the search ranges of the variables in iterative algorithms.*

 - void `optimize_step` ()
- Function to do a step of the iterative algorithm.*

 - void `optimize_iterate` ()
- Function to iterate the algorithm.*

 - void `optimize_free` ()
- Function to free the memory used by the `Optimize` struct.*

 - void `optimize_open` ()
- Function to open and perform a optimization.*

Variables

- int [ntasks](#)
Number of tasks.
- unsigned int [nthreads](#)
Number of threads.
- unsigned int [nthreads_direction](#)
Number of threads for the direction search method.
- GMutex [mutex](#) [1]
Mutex struct.
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_direction](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the direction.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize optimize](#) [1]
Optimization data.

5.17.1 Detailed Description

Source file to define the optimization functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.17.2 Function Documentation

5.17.2.1 void [optimize_best](#) (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

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

Definition at line [462](#) of file [optimize.c](#).

00463 {

```

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

```

5.17.2.2 void optimize_best_direction (unsigned int *simulation*, double *value*)

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

Parameters

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

Definition at line 787 of file [optimize.c](#).

```

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

```


5.17.2.3 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

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

Definition at line 817 of file [optimize.c](#).

```

00818 {
00819     unsigned int i, j;
00820     double e;
00821     #if DEBUG_OPTIMIZE
00822         fprintf (stderr, "optimize_direction_sequential: start\n");
00823         fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00824                 "nend_direction=%u\n",
00825                 optimize->nstart_direction, optimize->
nend_direction);
00826     #endif
00827     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829         j = simulation + i;
00830         e = optimize_norm (j);
00831         optimize_best_direction (j, e);
00832         optimize_save_variables (j, e);
00833         if (e < optimize->threshold)
00834         {
00835             optimize->stop = 1;
00836             break;
00837         }
00838     #if DEBUG_OPTIMIZE
00839         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00840     #endif
00841     }
00842     #if DEBUG_OPTIMIZE
00843         fprintf (stderr, "optimize_direction_sequential: end\n");
00844     #endif
00845 }
```

Here is the call graph for this function:

5.17.2.4 void * optimize_direction_thread (ParallelData * data)

Function to estimate the direction search on a thread.

Parameters

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

Returns

NULL

Definition at line 855 of file [optimize.c](#).

```

00856 {
00857     unsigned int i, thread;
00858     double e;
00859     #if DEBUG_OPTIMIZE
00860         fprintf (stderr, "optimize_direction_thread: start\n");
00861     #endif
00862     thread = data->thread;
00863     #if DEBUG_OPTIMIZE
00864         fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865                 thread,
00866                 optimize->thread_direction[thread],
00867                 optimize->thread_direction[thread + 1]);
00868     #endif
```

```

00869     for (i = optimize->thread_direction[thread];
00870          i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872         e = optimize_norm (i);
00873         g_mutex_lock (mutex);
00874         optimize_best_direction (i, e);
00875         optimize_save_variables (i, e);
00876         if (e < optimize->threshold)
00877             optimize->stop = 1;
00878         g_mutex_unlock (mutex);
00879         if (optimize->stop)
00880             break;
00881 #if DEBUG_OPTIMIZE
00882         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886     fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888     g_thread_exit (NULL);
00889     return NULL;
00890 }

```

Here is the call graph for this function:

5.17.2.5 double optimize_estimate_direction_coordinates (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 929 of file [optimize.c](#).

```

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

```

5.17.2.6 double optimize_estimate_direction_random (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 902 of file `optimize.c`.

```

00904 {
00905     double x;
00906     #if DEBUG_OPTIMIZE
00907     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908     #endif
00909     x = optimize->direction[variable]
00910         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00911         step[variable];
00912     #if DEBUG_OPTIMIZE
00913     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00914             variable, x);
00915     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00916     #endif
00917     return x;
00918 }
```

5.17.2.7 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 1096 of file `optimize.c`.

```

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

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

Function to write the simulation input file.

Parameters

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

Definition at line 103 of file [optimize.c](#).

```

00104 {
00105     unsigned int i;
00106     char buffer[32], value[32], *buffer2, *buffer3, *content;
00107     FILE *file;
00108     gsize length;
00109     GRegex *regex;
00110
00111     #if DEBUG_OPTIMIZE
00112     fprintf (stderr, "optimize_input: start\n");
00113     #endif
00114
00115     // Checking the file
00116     if (!template)
00117         goto optimize_input_end;
00118
00119     // Opening template
00120     content = g_mapped_file_get_contents (template);
00121     length = g_mapped_file_get_length (template);
00122     #if DEBUG_OPTIMIZE
00123     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00124     #endif
00125     file = g_fopen (input, "w");
00126
00127     // Parsing template
00128     for (i = 0; i < optimize->nvariables; ++i)
00129     {
00130         #if DEBUG_OPTIMIZE
00131         fprintf (stderr, "optimize_input: variable=%u\n", i);
00132         #endif
00133         snprintf (buffer, 32, "@variable%u@", i + 1);
00134         regex = g_regex_new (buffer, 0, 0, NULL);
00135         if (i == 0)
00136         {
00137             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                             optimize->label[i], 0, NULL);
00139             #if DEBUG_OPTIMIZE
00140             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00141             #endif
00142         }
00143         else
00144         {
00145             length = strlen (buffer3);
00146             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                             optimize->label[i], 0, NULL);
00148             g_free (buffer3);
00149         }
00150         g_regex_unref (regex);
00151         length = strlen (buffer2);
00152         snprintf (buffer, 32, "@value%u@", i + 1);
00153         regex = g_regex_new (buffer, 0, 0, NULL);
00154         snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->
nvariables + i]);
00156
00157         #if DEBUG_OPTIMIZE
00158         fprintf (stderr, "optimize_input: value=%s\n", value);
00159         #endif
00160         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00161                                         0, NULL);
00162         g_free (buffer2);
00163         g_regex_unref (regex);
00164     }
00165
00166     // Saving input file
00167     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168     g_free (buffer3);
00169     fclose (file);
00170
00171 optimize_input_end:
00172     #if DEBUG_OPTIMIZE
00173     fprintf (stderr, "optimize_input: end\n");
00174     #endif
00175     return;
00176 }

```

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

Function to merge the 2 optimization results.

Parameters

<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 585 of file [optimize.c](#).

```

00587 {
00588     unsigned int i, j, k, s[optimize->nbest];
00589     double e[optimize->nbest];
00590     #if DEBUG_OPTIMIZE
00591     fprintf (stderr, "optimize_merge: start\n");
00592     #endif
00593     i = j = k = 0;
00594     do
00595     {
00596         if (i == optimize->nsaveds)
00597         {
00598             s[k] = simulation_best[j];
00599             e[k] = error_best[j];
00600             ++j;
00601             ++k;
00602             if (j == nsaveds)
00603                 break;
00604         }
00605         else if (j == nsaveds)
00606         {
00607             s[k] = optimize->simulation_best[i];
00608             e[k] = optimize->error_best[i];
00609             ++i;
00610             ++k;
00611             if (i == optimize->nsaveds)
00612                 break;
00613         }
00614         else if (optimize->error_best[i] > error_best[j])
00615         {
00616             s[k] = simulation_best[j];
00617             e[k] = error_best[j];
00618             ++j;
00619             ++k;
00620         }
00621         else
00622         {
00623             s[k] = optimize->simulation_best[i];
00624             e[k] = optimize->error_best[i];
00625             ++i;
00626             ++k;
00627         }
00628     }
00629     while (k < optimize->nbest);
00630     optimize->nsaveds = k;
00631     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632     memcpy (optimize->error_best, e, k * sizeof (double));
00633     #if DEBUG_OPTIMIZE
00634     fprintf (stderr, "optimize_merge: end\n");
00635     #endif
00636 }

```

5.17.2.10 double optimize_norm_euclidian (unsigned int *simulation*)

Function to calculate the Euclidian error norm.

Parameters

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

Returns

Euclidian error norm.

Definition at line 295 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.17.2.11 double optimize_norm_maximum (unsigned int *simulation*)

Function to calculate the maximum error norm.

Parameters

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

Returns

Maximum error norm.

Definition at line 324 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.17.2.12 double optimize_norm_p (unsigned int *simulation*)

Function to calculate the P error norm.

Parameters

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

Returns

P error norm.

Definition at line 352 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.17.2.13 double optimize_norm_taxicab (unsigned int *simulation*)

Function to calculate the taxicab error norm.

Parameters

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

Returns

Taxicab error norm.

Definition at line 381 of file [optimize.c](#).

```

00382 {
00383     double e;
00384     unsigned int i;
00385     #if DEBUG_OPTIMIZE
00386     fprintf (stderr, "optimize_norm_taxicab: start\n");
00387     #endif
00388     e = 0.;
00389     for (i = 0; i < optimize->nexperiments; ++i)
00390     {
00391         e += fabs (optimize_parse (simulation, i));
00392     }
00393     #if DEBUG_OPTIMIZE
00394     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00395     fprintf (stderr, "optimize_norm_taxicab: end\n");
00396     #endif
00397     return e;
00398 }
```

Here is the call graph for this function:

5.17.2.14 double optimize_parse (unsigned int *simulation*, unsigned int *experiment*)

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

Parameters

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

Returns

Objective function value.

Definition at line 189 of file [optimize.c](#).

```

00190 {
00191     unsigned int i;
00192     double e;
00193     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00194         *buffer3, *buffer4;
00195     FILE *file_result;
00196
00197     #if DEBUG_OPTIMIZE
00198         fprintf (stderr, "optimize_parse: start\n");
00199         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00200             experiment);
00201     #endif
00202
00203     // Opening input files
00204     for (i = 0; i < optimize->ninputs; ++i)
00205     {
00206         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00207         #if DEBUG_OPTIMIZE
00208             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00209         #endif
00210         optimize_input (simulation, &input[i][0], optimize->
00211             file[i][experiment]);
00212     }
00213     for (; i < MAX_NINPUTS; ++i)
00214         strcpy (&input[i][0], "");
00215     #if DEBUG_OPTIMIZE
00216         fprintf (stderr, "optimize_parse: parsing end\n");
00217     #endif
00218
00219     // Performing the simulation
00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "%s\n" %s %s %s %s %s %s %s %s %s %s",
00225         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00226         input[6], input[7], output);
00227     g_free (buffer4);
00228     g_free (buffer3);
00229     g_free (buffer2);
00230     #if DEBUG_OPTIMIZE
00231         fprintf (stderr, "optimize_parse: %s\n", buffer);
00232     #endif
00233     system (buffer);
00234
00235     // Checking the objective value function
00236     if (optimize->evaluator)
00237     {
00238         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00239         buffer2 = g_path_get_dirname (optimize->evaluator);
00240         buffer3 = g_path_get_basename (optimize->evaluator);
00241         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00242         snprintf (buffer, 512, "%s\n" %s %s %s",
00243             buffer4, output, optimize->experiment[experiment], result);
00244         g_free (buffer4);
00245         g_free (buffer3);
00246         g_free (buffer2);
00247         #if DEBUG_OPTIMIZE
00248             fprintf (stderr, "optimize_parse: %s\n", buffer);
00249         #endif
00250     }
00251 }
```

```

00249     system (buffer);
00250     file_result = g_fopen (result, "r");
00251     e = atof (fgets (buffer, 512, file_result));
00252     fclose (file_result);
00253 }
00254 else
00255 {
00256     strcpy (result, "");
00257     file_result = g_fopen (output, "r");
00258     e = atof (fgets (buffer, 512, file_result));
00259     fclose (file_result);
00260 }
00261
00262 // Removing files
00263 #if !DEBUG_OPTIMIZE
00264 for (i = 0; i < optimize->ninputs; ++i)
00265 {
00266     if (optimize->file[i][0])
00267     {
00268         snprintf (buffer, 512, RM " %s", &input[i][0]);
00269         system (buffer);
00270     }
00271 }
00272 snprintf (buffer, 512, RM " %s %s", output, result);
00273 system (buffer);
00274 #endif
00275
00276 // Processing pending events
00277 show_pending ();
00278
00279 #if DEBUG_OPTIMIZE
00280 fprintf (stderr, "optimize_parse: end\n");
00281 #endif
00282
00283 // Returning the objective function
00284 return e * optimize->weight[experiment];
00285 }

```

Here is the call graph for this function:

5.17.2.15 void optimize_save_variables (unsigned int *simulation*, double *error*)

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

Parameters

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

Definition at line 434 of file [optimize.c](#).

```

00435 {
00436     unsigned int i;
00437     char buffer[64];
00438     #if DEBUG_OPTIMIZE
00439     fprintf (stderr, "optimize_save_variables: start\n");
00440     #endif
00441     for (i = 0; i < optimize->nvariables; ++i)
00442     {
00443         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444         fprintf (optimize->file_variables, buffer,
00445             optimize->value[simulation * optimize->
nvariables + i]);
00446     }
00447     fprintf (optimize->file_variables, "%.14le\n", error);
00448     #if DEBUG_OPTIMIZE
00449     fprintf (stderr, "optimize_save_variables: end\n");
00450     #endif
00451 }

```

5.17.2.16 void optimize_step_direction (unsigned int *simulation*)

Function to do a step of the direction search method.

Parameters

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

Definition at line 960 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:

5.17.2.17 void * optimize_thread (ParallelData * data)

Function to optimize on a thread.

Parameters

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

Returns

NULL

Definition at line 539 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.18 optimize.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burquete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
```

```

00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <stdlib.h>
00036 #include <string.h>
00037 #include <math.h>
00038 #include <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 #elif !defined(__BSD_VISIBLE)
00047 #include <alloca.h>
00048 #endif
00049 #if HAVE_MPI
00050 #include <mpi.h>
00051 #endif
00052 #include "genetic/genetic.h"
00053 #include "utils.h"
00054 #include "experiment.h"
00055 #include "variable.h"
00056 #include "input.h"
00057 #include "optimize.h"
00058
00059 #define DEBUG_OPTIMIZE 0
00060
00061 #ifdef G_OS_WIN32
00062 #define RM "del"
00063 #else
00064 #define RM "rm"
00065 #endif
00066
00067 int ntasks;
00068 unsigned int nthreads;
00069 unsigned int nthreads_direction;
00070 GMutex mutex[1];
00071 void (*optimize_algorithm) ();
00072 double (*optimize_estimate_direction) (unsigned int variable,
00073                                       unsigned int estimate);
00074 double (*optimize_norm) (unsigned int simulation);
00075 Optimize optimize[1];
00076
00077 void
00078 optimize_input (unsigned int simulation, char *input, GMappedFile * template)
00079 {
00080     unsigned int i;
00081     char buffer[32], value[32], *buffer2, *buffer3, *content;
00082     FILE *file;
00083     gsize length;
00084     GRegex *regex;
00085
00086     #if DEBUG_OPTIMIZE
00087         fprintf (stderr, "optimize_input: start\n");
00088     #endif
00089
00090     // Checking the file
00091     if (!template)
00092         goto optimize_input_end;
00093
00094     // Opening template
00095     content = g_mapped_file_get_contents (template);
00096     length = g_mapped_file_get_length (template);
00097     #if DEBUG_OPTIMIZE
00098         fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00099     #endif
00100     file = g_fopen (input, "w");
00101
00102     // Parsing template
00103     for (i = 0; i < optimize->nvariables; ++i)

```

```

00129     {
00130     #if DEBUG_OPTIMIZE
00131         fprintf (stderr, "optimize_input: variable=%u\n", i);
00132     #endif
00133         snprintf (buffer, 32, "@variable%u@", i + 1);
00134         regex = g_regex_new (buffer, 0, 0, NULL);
00135         if (i == 0)
00136         {
00137             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                             optimize->label[i], 0, NULL);
00139     #if DEBUG_OPTIMIZE
00140             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00141     #endif
00142         }
00143         else
00144         {
00145             length = strlen (buffer3);
00146             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                             optimize->label[i], 0, NULL);
00148             g_free (buffer3);
00149         }
00150         g_regex_unref (regex);
00151         length = strlen (buffer2);
00152         snprintf (buffer, 32, "@value%u@", i + 1);
00153         regex = g_regex_new (buffer, 0, 0, NULL);
00154         snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->nvariables + i]);
00156     #if DEBUG_OPTIMIZE
00157         fprintf (stderr, "optimize_input: value=%s\n", value);
00158     #endif
00159         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00160                                         0, NULL);
00161         g_free (buffer2);
00162         g_regex_unref (regex);
00163     }
00164 }
00165
00166 // Saving input file
00167 fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168 g_free (buffer3);
00169 fclose (file);
00170
00171 optimize_input_end:
00172 #if DEBUG_OPTIMIZE
00173     fprintf (stderr, "optimize_input: end\n");
00174 #endif
00175     return;
00176 }
00177
00178 double
00179 optimize_parse (unsigned int simulation, unsigned int experiment)
00180 {
00181     unsigned int i;
00182     double e;
00183     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00184         *buffer3, *buffer4;
00185     FILE *file_result;
00186
00187     #if DEBUG_OPTIMIZE
00188         fprintf (stderr, "optimize_parse: start\n");
00189         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00190                 experiment);
00191     #endif
00192
00193     // Opening input files
00194     for (i = 0; i < optimize->ninputs; ++i)
00195     {
00196         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00197     #if DEBUG_OPTIMIZE
00198         fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00199     #endif
00200         optimize_input (simulation, &input[i][0], optimize->file[i][experiment]);
00201     }
00202     for (; i < MAX_NINPUTS; ++i)
00203         strcpy (&input[i][0], "");
00204     #if DEBUG_OPTIMIZE
00205         fprintf (stderr, "optimize_parse: parsing end\n");
00206     #endif
00207
00208     // Performing the simulation
00209     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00210     buffer2 = g_path_get_dirname (optimize->simulator);
00211     buffer3 = g_path_get_basename (optimize->simulator);
00212     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00213     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00214             buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00215             input[6], input[7], output);

```

```

00226     g_free (buffer4);
00227     g_free (buffer3);
00228     g_free (buffer2);
00229 #if DEBUG_OPTIMIZE
00230     fprintf (stderr, "optimize_parse: %s\n", buffer);
00231 #endif
00232     system (buffer);
00233
00234     // Checking the objective value function
00235     if (optimize->evaluator)
00236     {
00237         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00238         buffer2 = g_path_get_dirname (optimize->evaluator);
00239         buffer3 = g_path_get_basename (optimize->evaluator);
00240         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00241         snprintf (buffer, 512, "\"%s\" %s %s %s",
00242                 buffer4, output, optimize->experiment[experiment], result);
00243         g_free (buffer4);
00244         g_free (buffer3);
00245         g_free (buffer2);
00246 #if DEBUG_OPTIMIZE
00247         fprintf (stderr, "optimize_parse: %s\n", buffer);
00248 #endif
00249         system (buffer);
00250         file_result = g_fopen (result, "r");
00251         e = atof (fgets (buffer, 512, file_result));
00252         fclose (file_result);
00253     }
00254     else
00255     {
00256         strcpy (result, "");
00257         file_result = g_fopen (output, "r");
00258         e = atof (fgets (buffer, 512, file_result));
00259         fclose (file_result);
00260     }
00261
00262     // Removing files
00263 #if !DEBUG_OPTIMIZE
00264     for (i = 0; i < optimize->ninputs; ++i)
00265     {
00266         if (optimize->file[i][0])
00267         {
00268             snprintf (buffer, 512, RM " %s", &input[i][0]);
00269             system (buffer);
00270         }
00271     }
00272     snprintf (buffer, 512, RM " %s %s", output, result);
00273     system (buffer);
00274 #endif
00275
00276     // Processing pending events
00277     show_pending ();
00278
00279 #if DEBUG_OPTIMIZE
00280     fprintf (stderr, "optimize_parse: end\n");
00281 #endif
00282
00283     // Returning the objective function
00284     return e * optimize->weight[experiment];
00285 }
00286
00294 double
00295 optimize_norm_euclidian (unsigned int simulation)
00296 {
00297     double e, ei;
00298     unsigned int i;
00299 #if DEBUG_OPTIMIZE
00300     fprintf (stderr, "optimize_norm_euclidian: start\n");
00301 #endif
00302     e = 0.;
00303     for (i = 0; i < optimize->nexperiments; ++i)
00304     {
00305         ei = optimize_parse (simulation, i);
00306         e += ei * ei;
00307     }
00308     e = sqrt (e);
00309 #if DEBUG_OPTIMIZE
00310     fprintf (stderr, "optimize_norm_euclidian: error=%lg\n", e);
00311     fprintf (stderr, "optimize_norm_euclidian: end\n");
00312 #endif
00313     return e;
00314 }
00315
00323 double
00324 optimize_norm_maximum (unsigned int simulation)
00325 {
00326     double e, ei;

```

```

00327     unsigned int i;
00328     #if DEBUG_OPTIMIZE
00329     fprintf (stderr, "optimize_norm_maximum: start\n");
00330     #endif
00331     e = 0.;
00332     for (i = 0; i < optimize->nexperiments; ++i)
00333     {
00334         ei = fabs (optimize_parse (simulation, i));
00335         e = fmax (e, ei);
00336     }
00337     #if DEBUG_OPTIMIZE
00338     fprintf (stderr, "optimize_norm_maximum: error=%lg\n", e);
00339     fprintf (stderr, "optimize_norm_maximum: end\n");
00340     #endif
00341     return e;
00342 }
00343
00351 double
00352 optimize_norm_p (unsigned int simulation)
00353 {
00354     double e, ei;
00355     unsigned int i;
00356     #if DEBUG_OPTIMIZE
00357     fprintf (stderr, "optimize_norm_p: start\n");
00358     #endif
00359     e = 0.;
00360     for (i = 0; i < optimize->nexperiments; ++i)
00361     {
00362         ei = fabs (optimize_parse (simulation, i));
00363         e += pow (ei, optimize->p);
00364     }
00365     e = pow (e, 1. / optimize->p);
00366     #if DEBUG_OPTIMIZE
00367     fprintf (stderr, "optimize_norm_p: error=%lg\n", e);
00368     fprintf (stderr, "optimize_norm_p: end\n");
00369     #endif
00370     return e;
00371 }
00372
00380 double
00381 optimize_norm_taxicab (unsigned int simulation)
00382 {
00383     double e;
00384     unsigned int i;
00385     #if DEBUG_OPTIMIZE
00386     fprintf (stderr, "optimize_norm_taxicab: start\n");
00387     #endif
00388     e = 0.;
00389     for (i = 0; i < optimize->nexperiments; ++i)
00390         e += fabs (optimize_parse (simulation, i));
00391     #if DEBUG_OPTIMIZE
00392     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00393     fprintf (stderr, "optimize_norm_taxicab: end\n");
00394     #endif
00395     return e;
00396 }
00397
00402 void
00403 optimize_print ()
00404 {
00405     unsigned int i;
00406     char buffer[512];
00407     #if HAVE_MPI
00408     if (optimize->mpi_rank)
00409         return;
00410     #endif
00411     printf ("%s\n", gettext ("Best result"));
00412     fprintf (optimize->file_result, "%s\n", gettext ("Best result"));
00413     printf ("error = %.15le\n", optimize->error_old[0]);
00414     fprintf (optimize->file_result, "error = %.15le\n", optimize->
error_old[0]);
00415     for (i = 0; i < optimize->nvariables; ++i)
00416     {
00417         snprintf (buffer, 512, "%s = %s\n",
00418                 optimize->label[i], format[optimize->precision[i]]);
00419         printf (buffer, optimize->value_old[i]);
00420         fprintf (optimize->file_result, buffer, optimize->value_old[i]);
00421     }
00422     fflush (optimize->file_result);
00423 }
00424
00433 void
00434 optimize_save_variables (unsigned int simulation, double error)
00435 {
00436     unsigned int i;
00437     char buffer[64];
00438     #if DEBUG_OPTIMIZE

```



```

00439     fprintf (stderr, "optimize_save_variables: start\n");
00440 #endif
00441     for (i = 0; i < optimize->nvariables; ++i)
00442     {
00443         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444         fprintf (optimize->file_variables, buffer,
00445                 optimize->value[simulation * optimize->nvariables + i]);
00446     }
00447     fprintf (optimize->file_variables, "%.14le\n", error);
00448 #if DEBUG_OPTIMIZE
00449     fprintf (stderr, "optimize_save_variables: end\n");
00450 #endif
00451 }
00452
00461 void
00462 optimize_best (unsigned int simulation, double value)
00463 {
00464     unsigned int i, j;
00465     double e;
00466 #if DEBUG_OPTIMIZE
00467     fprintf (stderr, "optimize_best: start\n");
00468     fprintf (stderr, "optimize_best: nsaveds=%u nbest=%u\n",
00469             optimize->nsaveds, optimize->nbest);
00470 #endif
00471     if (optimize->nsaveds < optimize->nbest
00472         || value < optimize->error_best[optimize->nsaveds - 1])
00473     {
00474         if (optimize->nsaveds < optimize->nbest)
00475             ++optimize->nsaveds;
00476         optimize->error_best[optimize->nsaveds - 1] = value;
00477         optimize->simulation_best[optimize->nsaveds - 1] = simulation;
00478         for (i = optimize->nsaveds; --i;)
00479         {
00480             if (optimize->error_best[i] < optimize->error_best[i - 1])
00481             {
00482                 j = optimize->simulation_best[i];
00483                 e = optimize->error_best[i];
00484                 optimize->simulation_best[i] = optimize->
simulation_best[i - 1];
00485                 optimize->error_best[i] = optimize->error_best[i - 1];
00486                 optimize->simulation_best[i - 1] = j;
00487                 optimize->error_best[i - 1] = e;
00488             }
00489             else
00490                 break;
00491         }
00492     }
00493 #if DEBUG_OPTIMIZE
00494     fprintf (stderr, "optimize_best: end\n");
00495 #endif
00496 }
00497
00502 void
00503 optimize_sequential ()
00504 {
00505     unsigned int i;
00506     double e;
00507 #if DEBUG_OPTIMIZE
00508     fprintf (stderr, "optimize_sequential: start\n");
00509     fprintf (stderr, "optimize_sequential: nstart=%u nend=%u\n",
00510             optimize->nstart, optimize->nend);
00511 #endif
00512     for (i = optimize->nstart; i < optimize->nend; ++i)
00513     {
00514         e = optimize_norm (i);
00515         optimize_best (i, e);
00516         optimize_save_variables (i, e);
00517         if (e < optimize->threshold)
00518         {
00519             optimize->stop = 1;
00520             break;
00521         }
00522 #if DEBUG_OPTIMIZE
00523         fprintf (stderr, "optimize_sequential: i=%u e=%lg\n", i, e);
00524 #endif
00525     }
00526 #if DEBUG_OPTIMIZE
00527     fprintf (stderr, "optimize_sequential: end\n");
00528 #endif
00529 }
00530
00538 void *
00539 optimize_thread (ParallelData * data)
00540 {
00541     unsigned int i, thread;
00542     double e;
00543 #if DEBUG_OPTIMIZE

```

```

00544     fprintf (stderr, "optimize_thread: start\n");
00545 #endif
00546     thread = data->thread;
00547 #if DEBUG_OPTIMIZE
00548     fprintf (stderr, "optimize_thread: thread=%u start=%u end=%u\n", thread,
00549             optimize->thread[thread], optimize->thread[thread + 1]);
00550 #endif
00551     for (i = optimize->thread[thread]; i < optimize->thread[thread + 1]; ++i)
00552     {
00553         e = optimize_norm (i);
00554         g_mutex_lock (mutex);
00555         optimize_best (i, e);
00556         optimize_save_variables (i, e);
00557         if (e < optimize->threshold)
00558             optimize->stop = 1;
00559         g_mutex_unlock (mutex);
00560         if (optimize->stop)
00561             break;
00562 #if DEBUG_OPTIMIZE
00563         fprintf (stderr, "optimize_thread: i=%u e=%lg\n", i, e);
00564 #endif
00565     }
00566 #if DEBUG_OPTIMIZE
00567     fprintf (stderr, "optimize_thread: end\n");
00568 #endif
00569     g_thread_exit (NULL);
00570     return NULL;
00571 }
00572
00584 void
00585 optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00586                double *error_best)
00587 {
00588     unsigned int i, j, k, s[optimize->nbest];
00589     double e[optimize->nbest];
00590 #if DEBUG_OPTIMIZE
00591     fprintf (stderr, "optimize_merge: start\n");
00592 #endif
00593     i = j = k = 0;
00594     do
00595     {
00596         if (i == optimize->nsaveds)
00597         {
00598             s[k] = simulation_best[j];
00599             e[k] = error_best[j];
00600             ++j;
00601             ++k;
00602             if (j == nsaveds)
00603                 break;
00604         }
00605         else if (j == nsaveds)
00606         {
00607             s[k] = optimize->simulation_best[i];
00608             e[k] = optimize->error_best[i];
00609             ++i;
00610             ++k;
00611             if (i == optimize->nsaveds)
00612                 break;
00613         }
00614         else if (optimize->error_best[i] > error_best[j])
00615         {
00616             s[k] = simulation_best[j];
00617             e[k] = error_best[j];
00618             ++j;
00619             ++k;
00620         }
00621         else
00622         {
00623             s[k] = optimize->simulation_best[i];
00624             e[k] = optimize->error_best[i];
00625             ++i;
00626             ++k;
00627         }
00628     }
00629     while (k < optimize->nbest);
00630     optimize->nsaveds = k;
00631     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632     memcpy (optimize->error_best, e, k * sizeof (double));
00633 #if DEBUG_OPTIMIZE
00634     fprintf (stderr, "optimize_merge: end\n");
00635 #endif
00636 }
00637
00642 #if HAVE_MPI
00643 void
00644 optimize_synchronise ()
00645 {

```

```

00646 unsigned int i, nsaveds, simulation_best[optimize->nbest], stop;
00647 double error_best[optimize->nbest];
00648 MPI_Status mpi_stat;
00649 #if DEBUG_OPTIMIZE
00650 fprintf (stderr, "optimize_synchronise: start\n");
00651 #endif
00652 if (optimize->mpi_rank == 0)
00653 {
00654     for (i = 1; i < ntasks; ++i)
00655     {
00656         MPI_Recv (&nsaveds, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &mpi_stat);
00657         MPI_Recv (simulation_best, nsaveds, MPI_INT, i, 1,
00658                 MPI_COMM_WORLD, &mpi_stat);
00659         MPI_Recv (error_best, nsaveds, MPI_DOUBLE, i, 1,
00660                 MPI_COMM_WORLD, &mpi_stat);
00661         optimize_merge (nsaveds, simulation_best, error_best);
00662         MPI_Recv (&stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD, &mpi_stat);
00663         if (stop)
00664             optimize->stop = 1;
00665     }
00666     for (i = 1; i < ntasks; ++i)
00667         MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, i, 1, MPI_COMM_WORLD);
00668 }
00669 else
00670 {
00671     MPI_Send (&optimize->nsaveds, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
00672     MPI_Send (optimize->simulation_best, optimize->nsaveds, MPI_INT, 0, 1,
00673             MPI_COMM_WORLD);
00674     MPI_Send (optimize->error_best, optimize->nsaveds, MPI_DOUBLE, 0, 1,
00675             MPI_COMM_WORLD);
00676     MPI_Send (&optimize->stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD);
00677     MPI_Recv (&stop, 1, MPI_UNSIGNED, 0, 1, MPI_COMM_WORLD, &mpi_stat);
00678     if (stop)
00679         optimize->stop = 1;
00680 }
00681 #if DEBUG_OPTIMIZE
00682 fprintf (stderr, "optimize_synchronise: end\n");
00683 #endif
00684 }
00685 #endif
00686
00691 void
00692 optimize_sweep ()
00693 {
00694     unsigned int i, j, k, l;
00695     double e;
00696     GThread *thread[nthreads];
00697     ParallelData data[nthreads];
00698 #if DEBUG_OPTIMIZE
00699 fprintf (stderr, "optimize_sweep: start\n");
00700 #endif
00701 for (i = 0; i < optimize->nsimulations; ++i)
00702 {
00703     k = i;
00704     for (j = 0; j < optimize->nvariables; ++j)
00705     {
00706         l = k % optimize->nsweeps[j];
00707         k /= optimize->nsweeps[j];
00708         e = optimize->rangemin[j];
00709         if (optimize->nsweeps[j] > 1)
00710             e += 1 * (optimize->rangemax[j] - optimize->rangemin[j])
00711                 / (optimize->nsweeps[j] - 1);
00712         optimize->value[i * optimize->nvariables + j] = e;
00713     }
00714 }
00715 optimize->nsaveds = 0;
00716 if (nthreads <= 1)
00717     optimize_sequential ();
00718 else
00719 {
00720     for (i = 0; i < nthreads; ++i)
00721     {
00722         data[i].thread = i;
00723         thread[i] = g_thread_new (NULL, (void *) optimize_thread, &data[i]);
00724     }
00725     for (i = 0; i < nthreads; ++i)
00726         g_thread_join (thread[i]);
00727 }
00728 #if HAVE_MPI
00729 // Communicating tasks results
00730 optimize_synchronise ();
00731 #endif
00732 #if DEBUG_OPTIMIZE
00733 fprintf (stderr, "optimize_sweep: end\n");
00734 #endif
00735 }
00736

```

```

00741 void
00742 optimize_MonteCarlo ()
00743 {
00744     unsigned int i, j;
00745     GThread *thread[nthreads];
00746     ParallelData data[nthreads];
00747     #if DEBUG_OPTIMIZE
00748     fprintf (stderr, "optimize_MonteCarlo: start\n");
00749     #endif
00750     for (i = 0; i < optimize->nsimulations; ++i)
00751         for (j = 0; j < optimize->nvariables; ++j)
00752             optimize->value[i * optimize->nvariables + j]
00753                 = optimize->rangemin[j] + gsl_rng_uniform (optimize->rng)
00754                   * (optimize->rangemax[j] - optimize->rangemin[j]);
00755     optimize->nsaveds = 0;
00756     if (nthreads <= 1)
00757         optimize_sequential ();
00758     else
00759     {
00760         for (i = 0; i < nthreads; ++i)
00761         {
00762             data[i].thread = i;
00763             thread[i] = g_thread_new (NULL, (void (*)) optimize_thread, &data[i]);
00764         }
00765         for (i = 0; i < nthreads; ++i)
00766             g_thread_join (thread[i]);
00767     }
00768     #if HAVE_MPI
00769     // Communicating tasks results
00770     optimize_synchronise ();
00771     #endif
00772     #if DEBUG_OPTIMIZE
00773     fprintf (stderr, "optimize_MonteCarlo: end\n");
00774     #endif
00775 }
00776
00786 void
00787 optimize_best_direction (unsigned int simulation, double value)
00788 {
00789     #if DEBUG_OPTIMIZE
00790     fprintf (stderr, "optimize_best_direction: start\n");
00791     fprintf (stderr,
00792             "optimize_best_direction: simulation=%u value=%.14le best=%.14le\n",
00793             simulation, value, optimize->error_best[0]);
00794     #endif
00795     if (value < optimize->error_best[0])
00796     {
00797         optimize->error_best[0] = value;
00798         optimize->simulation_best[0] = simulation;
00799     }
00800     #if DEBUG_OPTIMIZE
00801     fprintf (stderr,
00802             "optimize_best_direction: BEST simulation=%u value=%.14le\n",
00803             simulation, value);
00804     #endif
00805     #if DEBUG_OPTIMIZE
00806     fprintf (stderr, "optimize_best_direction: end\n");
00807     #endif
00808 }
00809
00816 void
00817 optimize_direction_sequential (unsigned int simulation)
00818 {
00819     unsigned int i, j;
00820     double e;
00821     #if DEBUG_OPTIMIZE
00822     fprintf (stderr, "optimize_direction_sequential: start\n");
00823     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00824             "nend_direction=%u\n",
00825             optimize->nstart_direction, optimize->nend_direction);
00826     #endif
00827     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829         j = simulation + i;
00830         e = optimize_norm (j);
00831         optimize_best_direction (j, e);
00832         optimize_save_variables (j, e);
00833         if (e < optimize->threshold)
00834         {
00835             optimize->stop = 1;
00836             break;
00837         }
00838     }
00839     #if DEBUG_OPTIMIZE
00840     fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00841     #endif
00842     #if DEBUG_OPTIMIZE

```

```

00843     fprintf (stderr, "optimize_direction_sequential: end\n");
00844 #endif
00845 }
00846
00854 void *
00855 optimize_direction_thread (ParallelData * data)
00856 {
00857     unsigned int i, thread;
00858     double e;
00859 #if DEBUG_OPTIMIZE
00860     fprintf (stderr, "optimize_direction_thread: start\n");
00861 #endif
00862     thread = data->thread;
00863 #if DEBUG_OPTIMIZE
00864     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865             thread,
00866             optimize->thread_direction[thread],
00867             optimize->thread_direction[thread + 1]);
00868 #endif
00869     for (i = optimize->thread_direction[thread];
00870          i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872         e = optimize_norm (i);
00873         g_mutex_lock (mutex);
00874         optimize_best_direction (i, e);
00875         optimize_save_variables (i, e);
00876         if (e < optimize->threshold)
00877             optimize->stop = 1;
00878         g_mutex_unlock (mutex);
00879         if (optimize->stop)
00880             break;
00881 #if DEBUG_OPTIMIZE
00882         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886     fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888     g_thread_exit (NULL);
00889     return NULL;
00890 }
00891
00901 double
00902 optimize_estimate_direction_random (unsigned int variable,
00903                                     unsigned int estimate)
00904 {
00905     double x;
00906 #if DEBUG_OPTIMIZE
00907     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908 #endif
00909     x = optimize->direction[variable]
00910         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->step[variable];
00911 #if DEBUG_OPTIMIZE
00912     fprintf (stderr, "optimize_estimate_direction_random: direction=%lg\n",
00913             variable, x);
00914     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00915 #endif
00916     return x;
00917 }
00918
00928 double
00929 optimize_estimate_direction_coordinates (unsigned int variable,
00930                                         unsigned int estimate)
00931 {
00932     double x;
00933 #if DEBUG_OPTIMIZE
00934     fprintf (stderr, "optimize_estimate_direction_coordinates: start\n");
00935 #endif
00936     x = optimize->direction[variable];
00937     if (estimate >= (2 * variable) && estimate < (2 * variable + 2))
00938     {
00939         if (estimate & 1)
00940             x += optimize->step[variable];
00941         else
00942             x -= optimize->step[variable];
00943     }
00944 #if DEBUG_OPTIMIZE
00945     fprintf (stderr,
00946             "optimize_estimate_direction_coordinates: direction=%lg\n",
00947             variable, x);
00948     fprintf (stderr, "optimize_estimate_direction_coordinates: end\n");
00949 #endif
00950     return x;
00951 }
00952
00959 void
00960 optimize_step_direction (unsigned int simulation)

```

```

00961 {
00962     GThread *thread[nthreads_direction];
00963     ParallelData data[nthreads_direction];
00964     unsigned int i, j, k, b;
00965     #if DEBUG_OPTIMIZE
00966     fprintf (stderr, "optimize_step_direction: start\n");
00967     #endif
00968     for (i = 0; i < optimize->nestimates; ++i)
00969     {
00970         k = (simulation + i) * optimize->nvariables;
00971         b = optimize->simulation_best[0] * optimize->nvariables;
00972         #if DEBUG_OPTIMIZE
00973         fprintf (stderr, "optimize_step_direction: simulation=%u best=%u\n",
00974                 simulation + i, optimize->simulation_best[0]);
00975         #endif
00976         for (j = 0; j < optimize->nvariables; ++j, ++k, ++b)
00977         {
00978             #if DEBUG_OPTIMIZE
00979             fprintf (stderr,
00980                     "optimize_step_direction: estimate=%u best%u=%.14le\n",
00981                     i, j, optimize->value[b]);
00982             #endif
00983             optimize->value[k]
00984                 = optimize->value[b] + optimize_estimate_direction (j, i);
00985             optimize->value[k] = fmin (fmax (optimize->value[k],
00986                                             optimize->rangeminabs[j]),
00987                                       optimize->rangemaxabs[j]);
00988             #if DEBUG_OPTIMIZE
00989             fprintf (stderr,
00990                     "optimize_step_direction: estimate=%u variable%u=%.14le\n",
00991                     i, j, optimize->value[k]);
00992             #endif
00993         }
00994     }
00995     if (nthreads_direction == 1)
00996         optimize_direction_sequential (simulation);
00997     else
00998     {
00999         for (i = 0; i <= nthreads_direction; ++i)
01000         {
01001             optimize->thread_direction[i]
01002                 = simulation + optimize->nstart_direction
01003                 + i * (optimize->nend_direction - optimize->
01004                       nstart_direction)
01005                 / nthreads_direction;
01006             #if DEBUG_OPTIMIZE
01007             fprintf (stderr,
01008                     "optimize_step_direction: i=%u thread_direction=%u\n",
01009                     i, optimize->thread_direction[i]);
01010             #endif
01011             for (i = 0; i < nthreads_direction; ++i)
01012             {
01013                 data[i].thread = i;
01014                 thread[i] = g_thread_new
01015                     (NULL, (void (*)(void *)) optimize_direction_thread, &data[i]);
01016             }
01017             for (i = 0; i < nthreads_direction; ++i)
01018                 g_thread_join (thread[i]);
01019         }
01020         #if DEBUG_OPTIMIZE
01021         fprintf (stderr, "optimize_step_direction: end\n");
01022         #endif
01023     }
01024 }
01025 void
01026 optimize_direction ()
01027 {
01028     unsigned int i, j, k, b, s, adjust;
01029     #if DEBUG_OPTIMIZE
01030     fprintf (stderr, "optimize_direction: start\n");
01031     #endif
01032     for (i = 0; i < optimize->nvariables; ++i)
01033         optimize->direction[i] = 0.;
01034     b = optimize->simulation_best[0] * optimize->nvariables;
01035     s = optimize->nsimulations;
01036     adjust = 1;
01037     for (i = 0; i < optimize->nsteps; ++i, s += optimize->nestimates, b = k)
01038     {
01039         #if DEBUG_OPTIMIZE
01040         fprintf (stderr, "optimize_direction: step=%u old_best=%u\n",
01041                 i, optimize->simulation_best[0]);
01042         #endif
01043         optimize_step_direction (s);
01044         k = optimize->simulation_best[0] * optimize->nvariables;
01045         #if DEBUG_OPTIMIZE
01046         fprintf (stderr, "optimize_direction: step=%u best=%u\n",

```

```

01051         i, optimize->simulation_best[0]);
01052 #endif
01053     if (k == b)
01054     {
01055         if (adjust)
01056             for (j = 0; j < optimize->nvariables; ++j)
01057                 optimize->step[j] *= 0.5;
01058         for (j = 0; j < optimize->nvariables; ++j)
01059             optimize->direction[j] = 0.;
01060         adjust = 1;
01061     }
01062     else
01063     {
01064         for (j = 0; j < optimize->nvariables; ++j)
01065         {
01066 #if DEBUG_OPTIMIZE
01067             fprintf (stderr,
01068                     "optimize_direction: best%u=%%.14le old%u=%%.14le\n",
01069                     j, optimize->value[k + j], j, optimize->value[b + j]);
01070 #endif
01071             optimize->direction[j]
01072                 = (1. - optimize->relaxation) * optimize->direction[j]
01073                   + optimize->relaxation
01074                     * (optimize->value[k + j] - optimize->value[b + j]);
01075 #if DEBUG_OPTIMIZE
01076             fprintf (stderr, "optimize_direction: direction%u=%%.14le\n",
01077                     j, optimize->direction[j]);
01078 #endif
01079         }
01080         adjust = 0;
01081     }
01082 }
01083 #if DEBUG_OPTIMIZE
01084 fprintf (stderr, "optimize_direction: end\n");
01085 #endif
01086 }
01087
01095 double
01096 optimize_genetic_objective (Entity * entity)
01097 {
01098     unsigned int j;
01099     double objective;
01100     char buffer[64];
01101 #if DEBUG_OPTIMIZE
01102     fprintf (stderr, "optimize_genetic_objective: start\n");
01103 #endif
01104     for (j = 0; j < optimize->nvariables; ++j)
01105     {
01106         optimize->value[entity->id * optimize->nvariables + j]
01107             = genetic_get_variable (entity, optimize->genetic_variable + j);
01108     }
01109     objective = optimize_norm (entity->id);
01110     g_mutex_lock (mutex);
01111     for (j = 0; j < optimize->nvariables; ++j)
01112     {
01113         snprintf (buffer, 64, "%s ", format[optimize->precision[j]]);
01114         fprintf (optimize->file_variables, buffer,
01115                 genetic_get_variable (entity, optimize->genetic_variable + j));
01116     }
01117     fprintf (optimize->file_variables, "%.14le\n", objective);
01118     g_mutex_unlock (mutex);
01119 #if DEBUG_OPTIMIZE
01120     fprintf (stderr, "optimize_genetic_objective: end\n");
01121 #endif
01122     return objective;
01123 }
01124
01129 void
01130 optimize_genetic ()
01131 {
01132     char *best_genome;
01133     double best_objective, *best_variable;
01134 #if DEBUG_OPTIMIZE
01135     fprintf (stderr, "optimize_genetic: start\n");
01136     fprintf (stderr, "optimize_genetic: ntasks=%u nthreads=%u\n", ntasks,
01137             nthreads);
01138     fprintf (stderr,
01139             "optimize_genetic: nvariables=%u population=%u generations=%u\n",
01140             optimize->nvariables, optimize->nsimulations, optimize->
01141             niterations);
01142     fprintf (stderr,
01143             "optimize_genetic: mutation=%lg reproduction=%lg adaptation=%lg\n",
01144             optimize->mutation_ratio, optimize->reproduction_ratio,
01145             optimize->adaptation_ratio);
01146 #endif
01147     genetic_algorithm_default (optimize->nvariables,
01148                               optimize->genetic_variable,

```

```

01148             optimize->nsimulations,
01149             optimize->niterations,
01150             optimize->mutation_ratio,
01151             optimize->reproduction_ratio,
01152             optimize->adaptation_ratio,
01153             optimize->seed,
01154             optimize->threshold,
01155             &optimize_genetic_objective,
01156             &best_genome, &best_variable, &best_objective);
01157 #if DEBUG_OPTIMIZE
01158     fprintf (stderr, "optimize_genetic: the best\n");
01159 #endif
01160     optimize->error_old = (double *) g_malloc (sizeof (double));
01161     optimize->value_old
01162         = (double *) g_malloc (optimize->nvariables * sizeof (double));
01163     optimize->error_old[0] = best_objective;
01164     memcpy (optimize->value_old, best_variable,
01165             optimize->nvariables * sizeof (double));
01166     g_free (best_genome);
01167     g_free (best_variable);
01168     optimize_print ();
01169 #if DEBUG_OPTIMIZE
01170     fprintf (stderr, "optimize_genetic: end\n");
01171 #endif
01172 }
01173
01174 void
01175 optimize_save_old ()
01176 {
01177     unsigned int i, j;
01178 #if DEBUG_OPTIMIZE
01179     fprintf (stderr, "optimize_save_old: start\n");
01180     fprintf (stderr, "optimize_save_old: nsaveds=%u\n", optimize->nsaveds);
01181 #endif
01182     memcpy (optimize->error_old, optimize->error_best,
01183             optimize->nbest * sizeof (double));
01184     for (i = 0; i < optimize->nbest; ++i)
01185     {
01186         j = optimize->simulation_best[i];
01187 #if DEBUG_OPTIMIZE
01188         fprintf (stderr, "optimize_save_old: i=%u j=%u\n", i, j);
01189 #endif
01190         memcpy (optimize->value_old + i * optimize->nvariables,
01191                 optimize->value + j * optimize->nvariables,
01192                 optimize->nvariables * sizeof (double));
01193     }
01194 #if DEBUG_OPTIMIZE
01195     for (i = 0; i < optimize->nvariables; ++i)
01196         fprintf (stderr, "optimize_save_old: best variable %u=%lg\n",
01197                 i, optimize->value_old[i]);
01198     fprintf (stderr, "optimize_save_old: end\n");
01199 #endif
01200 }
01201
01202 void
01203 optimize_merge_old ()
01204 {
01205     unsigned int i, j, k;
01206     double v[optimize->nbest * optimize->nvariables], e[optimize->
01207         nbest],
01208         *enew, *eold;
01209 #if DEBUG_OPTIMIZE
01210     fprintf (stderr, "optimize_merge_old: start\n");
01211 #endif
01212     anew = optimize->error_best;
01213     eold = optimize->error_old;
01214     i = j = k = 0;
01215     do
01216     {
01217         if (*enew < *eold)
01218         {
01219             memcpy (v + k * optimize->nvariables,
01220                     optimize->value
01221                     + optimize->simulation_best[i] * optimize->
01222                     nvariables,
01223                     optimize->nvariables * sizeof (double));
01224             e[k] = *enew;
01225             ++k;
01226             ++enew;
01227             ++i;
01228         }
01229         else
01230         {
01231             memcpy (v + k * optimize->nvariables,
01232                     optimize->value_old + j * optimize->nvariables,
01233                     optimize->nvariables * sizeof (double));
01234             e[k] = *eold;

```



```

01242         ++k;
01243         ++eold;
01244         ++j;
01245     }
01246 }
01247 while (k < optimize->nbest);
01248 memcpy (optimize->value_old, v, k * optimize->nvariables * sizeof (double));
01249 memcpy (optimize->error_old, e, k * sizeof (double));
01250 #if DEBUG_OPTIMIZE
01251 fprintf (stderr, "optimize_merge_old: end\n");
01252 #endif
01253 }
01254
01255 void
01260 optimize_refine ()
01261 {
01262     unsigned int i, j;
01263     double d;
01264 #if HAVE_MPI
01265     MPI_Status mpi_stat;
01266 #endif
01267 #if DEBUG_OPTIMIZE
01268     fprintf (stderr, "optimize_refine: start\n");
01269 #endif
01270 #if HAVE_MPI
01271     if (!optimize->mpi_rank)
01272     {
01273         for (j = 0; j < optimize->nvariables; ++j)
01274         {
01275             optimize->rangemin[j] = optimize->rangemax[j]
01276             = optimize->value_old[j];
01277         }
01278         for (i = 0; ++i < optimize->nbest; )
01279         {
01280             for (j = 0; j < optimize->nvariables; ++j)
01281             {
01282                 optimize->rangemin[j]
01283                 = fmin (optimize->rangemin[j],
01284                     optimize->value_old[i * optimize->nvariables + j]);
01285                 optimize->rangemax[j]
01286                 = fmax (optimize->rangemax[j],
01287                     optimize->value_old[i * optimize->nvariables + j]);
01288             }
01289         }
01290     }
01291     for (j = 0; j < optimize->nvariables; ++j)
01292     {
01293         d = optimize->tolerance
01294         * (optimize->rangemax[j] - optimize->rangemin[j]);
01295         switch (optimize->algorithm)
01296         {
01297             case ALGORITHM_MONTE_CARLO:
01298                 d *= 0.5;
01299                 break;
01300             default:
01301                 if (optimize->nsweeps[j] > 1)
01302                     d /= optimize->nsweeps[j] - 1;
01303                 else
01304                     d = 0.;
01305         }
01306         optimize->rangemin[j] -= d;
01307         optimize->rangemin[j]
01308         = fmax (optimize->rangemin[j], optimize->rangeminabs[j]);
01309         optimize->rangemax[j] += d;
01310         optimize->rangemax[j]
01311         = fmin (optimize->rangemax[j], optimize->rangemaxabs[j]);
01312         printf ("%s min=%lg max=%lg\n", optimize->label[j],
01313             optimize->rangemin[j], optimize->rangemax[j]);
01314         fprintf (optimize->file_result, "%s min=%lg max=%lg\n",
01315             optimize->label[j], optimize->rangemin[j],
01316             optimize->rangemax[j]);
01317     }
01318 #if HAVE_MPI
01319     for (i = 1; i < ntasks; ++i)
01320     {
01321         MPI_Send (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, i,
01322             1, MPI_COMM_WORLD);
01323         MPI_Send (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, i,
01324             1, MPI_COMM_WORLD);
01325     }
01326 }
01327 }
01328 else
01329 {
01330     MPI_Recv (optimize->rangemin, optimize->nvariables, MPI_DOUBLE, 0, 1,
01331         MPI_COMM_WORLD, &mpi_stat);
01332     MPI_Recv (optimize->rangemax, optimize->nvariables, MPI_DOUBLE, 0, 1,
01333         MPI_COMM_WORLD, &mpi_stat);

```

```

01334     }
01335 #endif
01336 #if DEBUG_OPTIMIZE
01337     fprintf (stderr, "optimize_refine: end\n");
01338 #endif
01339 }
01340
01341 void
01342 optimize_step ()
01343 {
01344     #if DEBUG_OPTIMIZE
01345     fprintf (stderr, "optimize_step: start\n");
01346     #endif
01347     optimize_algorithm ();
01348     if (optimize->nsteps)
01349         optimize_direction ();
01350     #if DEBUG_OPTIMIZE
01351     fprintf (stderr, "optimize_step: end\n");
01352     #endif
01353 }
01354
01355 void
01356 optimize_iterate ()
01357 {
01358     unsigned int i;
01359     #if DEBUG_OPTIMIZE
01360     fprintf (stderr, "optimize_iterate: start\n");
01361     #endif
01362     optimize->error_old = (double *) g_malloc (optimize->nbest * sizeof (double));
01363     optimize->value_old = (double *)
01364         g_malloc (optimize->nbest * optimize->nvariables * sizeof (double));
01365     optimize_step ();
01366     optimize_save_old ();
01367     optimize_refine ();
01368     optimize_print ();
01369     for (i = 1; i < optimize->niterations && !optimize->stop; ++i)
01370     {
01371         optimize_step ();
01372         optimize_merge_old ();
01373         optimize_refine ();
01374         optimize_print ();
01375     }
01376     #if DEBUG_OPTIMIZE
01377     fprintf (stderr, "optimize_iterate: end\n");
01378     #endif
01379 }
01380
01381 void
01382 optimize_free ()
01383 {
01384     unsigned int i, j;
01385     #if DEBUG_OPTIMIZE
01386     fprintf (stderr, "optimize_free: start\n");
01387     #endif
01388     for (j = 0; j < optimize->ninputs; ++j)
01389     {
01390         for (i = 0; i < optimize->nexperiments; ++i)
01391             g_mapped_file_unref (optimize->file[j][i]);
01392         g_free (optimize->file[j]);
01393     }
01394     g_free (optimize->error_old);
01395     g_free (optimize->value_old);
01396     g_free (optimize->value);
01397     g_free (optimize->genetic_variable);
01398     #if DEBUG_OPTIMIZE
01399     fprintf (stderr, "optimize_free: end\n");
01400     #endif
01401 }
01402
01403 void
01404 optimize_open ()
01405 {
01406     GTimeZone *tz;
01407     GDateTime *t0, *t;
01408     unsigned int i, j;
01409
01410     #if DEBUG_OPTIMIZE
01411     char *buffer;
01412     fprintf (stderr, "optimize_open: start\n");
01413     #endif
01414
01415     // Getting initial time
01416     #if DEBUG_OPTIMIZE
01417     fprintf (stderr, "optimize_open: getting initial time\n");
01418     #endif
01419     tz = g_time_zone_new_utc ();
01420     t0 = g_date_time_new_now (tz);

```

```

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

```

```

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

```

```

01604     optimize_norm = optimize_norm_taxicab;
01605 }
01606
01607 // Allocating values
01608 #if DEBUG_OPTIMIZE
01609 fprintf (stderr, "optimize_open: allocating variables\n");
01610 fprintf (stderr, "optimize_open: nvariables=%u algorithm=%u\n",
01611         optimize->nvariables, optimize->algorithm);
01612 #endif
01613 optimize->genetic_variable = NULL;
01614 if (optimize->algorithm == ALGORITHM_GENETIC)
01615 {
01616     optimize->genetic_variable = (GeneticVariable *)
01617     g_malloc (optimize->nvariables * sizeof (GeneticVariable));
01618     for (i = 0; i < optimize->nvariables; ++i)
01619     {
01620 #if DEBUG_OPTIMIZE
01621         fprintf (stderr, "optimize_open: i=%u min=%lg max=%lg nbits=%u\n",
01622                 i, optimize->rangemin[i], optimize->rangemax[i],
01623                 optimize->nbits[i]);
01624 #endif
01625         optimize->genetic_variable[i].minimum = optimize->
01626         rangemin[i];
01627         optimize->genetic_variable[i].maximum = optimize->
01628         rangemax[i];
01629         optimize->genetic_variable[i].nbits = optimize->nbits[i];
01630     }
01631 #if DEBUG_OPTIMIZE
01632     fprintf (stderr, "optimize_open: nvariables=%u nsimulations=%u\n",
01633             optimize->nvariables, optimize->nsimulations);
01634 #endif
01635     optimize->value = (double *)
01636     g_malloc ((optimize->nsimulations
01637             + optimize->nestimates * optimize->nsteps)
01638             * optimize->nvariables * sizeof (double));
01639     // Calculating simulations to perform for each task
01640 #if HAVE_MPI
01641 #if DEBUG_OPTIMIZE
01642     fprintf (stderr, "optimize_open: rank=%u ntasks=%u\n",
01643             optimize->mpi_rank, ntasks);
01644 #endif
01645     optimize->nstart = optimize->mpi_rank * optimize->nsimulations /
01646     ntasks;
01647     optimize->nend = (1 + optimize->mpi_rank) * optimize->nsimulations /
01648     ntasks;
01649     if (optimize->nsteps)
01650     {
01651         optimize->nstart_direction
01652         = optimize->mpi_rank * optimize->nestimates / ntasks;
01653         optimize->nend_direction
01654         = (1 + optimize->mpi_rank) * optimize->nestimates /
01655         ntasks;
01656     }
01657 #else
01658     optimize->nstart = 0;
01659     optimize->nend = optimize->nsimulations;
01660     if (optimize->nsteps)
01661     {
01662         optimize->nstart_direction = 0;
01663         optimize->nend_direction = optimize->nestimates;
01664     }
01665 #endif
01666 #if DEBUG_OPTIMIZE
01667     fprintf (stderr, "optimize_open: nstart=%u nend=%u\n", optimize->nstart,
01668             optimize->nend);
01669 #endif
01670 // Calculating simulations to perform for each thread
01671 optimize->thread
01672 = (unsigned int *) alloca ((1 + nthreads) * sizeof (unsigned int));
01673 for (i = 0; i <= nthreads; ++i)
01674 {
01675     optimize->thread[i] = optimize->nstart
01676     + i * (optimize->nend - optimize->nstart) / nthreads;
01677 #if DEBUG_OPTIMIZE
01678     fprintf (stderr, "optimize_open: i=%u thread=%u\n", i,
01679             optimize->thread[i]);
01680 #endif
01681 }
01682 if (optimize->nsteps)
01683     optimize->thread_direction = (unsigned int *)
01684     alloca ((1 + nthreads_direction) * sizeof (unsigned int));
01685 // Opening result files
01686 optimize->file_result = g_fopen (optimize->result, "w");

```

```

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

```

5.19 optimize.h File Reference

Header file to define the optimization functions.

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

Data Structures

- struct [Optimize](#)
Struct to define the optimization ation data.
- struct [ParallelData](#)
Struct to pass to the GThreads parallelized function.

Functions

- void [optimize_input](#) (unsigned int simulation, char *[input](#), GMappedFile *[template](#))
Function to write the simulation input file.
- double [optimize_parse](#) (unsigned int simulation, unsigned int experiment)
Function to parse input files, simulating and calculating the \ objective function.
- double [optimize_norm_euclidian](#) (unsigned int simulation)
Function to calculate the Euclidian error norm.
- double [optimize_norm_maximum](#) (unsigned int simulation)
Function to calculate the maximum error norm.
- double [optimize_norm_p](#) (unsigned int simulation)
Function to calculate the P error norm.
- double [optimize_norm_taxicab](#) (unsigned int simulation)
Function to calculate the taxicab error norm.

- void [optimize_print](#) ()
Function to print the results.
- void [optimize_save_variables](#) (unsigned int simulation, double error)
Function to save in a file the variables and the error.
- void [optimize_best](#) (unsigned int simulation, double value)
Function to save the best simulations.
- void [optimize_sequential](#) ()
Function to optimize sequentially.
- void * [optimize_thread](#) (ParallelData *data)
Function to optimize on a thread.
- void [optimize_merge](#) (unsigned int nsaveds, unsigned int *simulation_best, double *error_best)
Function to merge the 2 optimization results.
- void [optimize_synchronise](#) ()
Function to synchronise the optimization results of MPI tasks.
- void [optimize_sweep](#) ()
Function to optimize with the sweep algorithm.
- void [optimize_MonteCarlo](#) ()
Function to optimize with the Monte-Carlo algorithm.
- void [optimize_best_direction](#) (unsigned int simulation, double value)
Function to save the best simulation in a direction search method.
- void [optimize_direction_sequential](#) (unsigned int simulation)
Function to estimate the direction search sequentially.
- void * [optimize_direction_thread](#) (ParallelData *data)
Function to estimate the direction search on a thread.
- double [optimize_estimate_direction_random](#) (unsigned int variable, unsigned int estimate)
Function to estimate a component of the direction search vector.
- double [optimize_estimate_direction_coordinates](#) (unsigned int variable, unsigned int estimate)
Function to estimate a component of the direction search vector.
- void [optimize_step_direction](#) (unsigned int simulation)
Function to do a step of the direction search method.
- void [optimize_direction](#) ()
Function to optimize with a direction search method.
- double [optimize_genetic_objective](#) (Entity *entity)
Function to calculate the objective function of an entity.
- void [optimize_genetic](#) ()
Function to optimize with the genetic algorithm.
- void [optimize_save_old](#) ()
Function to save the best results on iterative methods.
- void [optimize_merge_old](#) ()
Function to merge the best results with the previous step best results on iterative methods.
- void [optimize_refine](#) ()
Function to refine the search ranges of the variables in iterative algorithms.
- void [optimize_step](#) ()
Function to do a step of the iterative algorithm.
- void [optimize_iterate](#) ()
Function to iterate the algorithm.
- void [optimize_free](#) ()
Function to free the memory used by the [Optimize](#) struct.
- void [optimize_open](#) ()
Function to open and perform a optimization.

Variables

- int [ntasks](#)
Number of tasks.
- unsigned int [nthreads](#)
Number of threads.
- unsigned int [nthreads_direction](#)
Number of threads for the direction search method.
- GMutex [mutex](#) [1]
Mutex struct.
- void(* [optimize_algorithm](#))()
Pointer to the function to perform a optimization algorithm step.
- double(* [optimize_estimate_direction](#))(unsigned int variable, unsigned int estimate)
Pointer to the function to estimate the direction.
- double(* [optimize_norm](#))(unsigned int simulation)
Pointer to the error norm function.
- [Optimize optimize](#) [1]
Optimization data.

5.19.1 Detailed Description

Header file to define the optimization functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.19.2 Function Documentation

5.19.2.1 void [optimize_best](#) (unsigned int *simulation*, double *value*)

Function to save the best simulations.

Parameters

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

Definition at line [462](#) of file [optimize.c](#).

00463 {


```

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

```

5.19.2.2 void optimize_best_direction (unsigned int *simulation*, double *value*)

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

Parameters

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

Definition at line 787 of file [optimize.c](#).

```

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

```

5.19.2.3 void optimize_direction_sequential (unsigned int *simulation*)

Function to estimate the direction search sequentially.

Parameters

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

Definition at line 817 of file [optimize.c](#).

```

00818 {
00819     unsigned int i, j;
00820     double e;
00821     #if DEBUG_OPTIMIZE
00822     fprintf (stderr, "optimize_direction_sequential: start\n");
00823     fprintf (stderr, "optimize_direction_sequential: nstart_direction=%u "
00824             "nend_direction=%u\n",
00825             optimize->nstart_direction, optimize->
nend_direction);
00826     #endif
00827     for (i = optimize->nstart_direction; i < optimize->nend_direction; ++i)
00828     {
00829         j = simulation + i;
00830         e = optimize_norm (j);
00831         optimize_best_direction (j, e);
00832         optimize_save_variables (j, e);
00833         if (e < optimize->threshold)
00834         {
00835             optimize->stop = 1;
00836             break;
00837         }
00838         #if DEBUG_OPTIMIZE
00839         fprintf (stderr, "optimize_direction_sequential: i=%u e=%lg\n", i, e);
00840         #endif
00841     }
00842     #if DEBUG_OPTIMIZE
00843     fprintf (stderr, "optimize_direction_sequential: end\n");
00844     #endif
00845 }
```

Here is the call graph for this function:

5.19.2.4 void* optimize_direction_thread (ParallelData * data)

Function to estimate the direction search on a thread.

Parameters

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

Returns

NULL

Definition at line 855 of file [optimize.c](#).

```

00856 {
00857     unsigned int i, thread;
00858     double e;
00859     #if DEBUG_OPTIMIZE
00860     fprintf (stderr, "optimize_direction_thread: start\n");
00861     #endif
00862     thread = data->thread;
00863     #if DEBUG_OPTIMIZE
00864     fprintf (stderr, "optimize_direction_thread: thread=%u start=%u end=%u\n",
00865             thread,
00866             optimize->thread_direction[thread],
00867             optimize->thread_direction[thread + 1]);
00868     #endif
```

```

00869     for (i = optimize->thread_direction[thread];
00870          i < optimize->thread_direction[thread + 1]; ++i)
00871     {
00872         e = optimize_norm (i);
00873         g_mutex_lock (mutex);
00874         optimize_best_direction (i, e);
00875         optimize_save_variables (i, e);
00876         if (e < optimize->threshold)
00877             optimize->stop = 1;
00878         g_mutex_unlock (mutex);
00879         if (optimize->stop)
00880             break;
00881 #if DEBUG_OPTIMIZE
00882         fprintf (stderr, "optimize_direction_thread: i=%u e=%lg\n", i, e);
00883 #endif
00884     }
00885 #if DEBUG_OPTIMIZE
00886     fprintf (stderr, "optimize_direction_thread: end\n");
00887 #endif
00888     g_thread_exit (NULL);
00889     return NULL;
00890 }

```

Here is the call graph for this function:

5.19.2.5 double optimize_estimate_direction_coordinates (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 929 of file [optimize.c](#).

```

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

```

5.19.2.6 double optimize_estimate_direction_random (unsigned int *variable*, unsigned int *estimate*)

Function to estimate a component of the direction search vector.

Parameters

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

Definition at line 902 of file [optimize.c](#).

```

00904 {
00905     double x;
00906     #if DEBUG_OPTIMIZE
00907     fprintf (stderr, "optimize_estimate_direction_random: start\n");
00908     #endif
00909     x = optimize->direction[variable]
00910         + (1. - 2. * gsl_rng_uniform (optimize->rng)) * optimize->
00911         step[variable];
00912     #if DEBUG_OPTIMIZE
00913     fprintf (stderr, "optimize_estimate_direction_random: direction%u=%lg\n",
00914             variable, x);
00915     fprintf (stderr, "optimize_estimate_direction_random: end\n");
00916     #endif
00917     return x;
00918 }
```

5.19.2.7 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 1096 of file [optimize.c](#).

```

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

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

Function to write the simulation input file.

Parameters

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

Definition at line 103 of file [optimize.c](#).

```

00104 {
00105     unsigned int i;
00106     char buffer[32], value[32], *buffer2, *buffer3, *content;
00107     FILE *file;
00108     gsize length;
00109     GRegex *regex;
00110
00111     #if DEBUG_OPTIMIZE
00112     fprintf (stderr, "optimize_input: start\n");
00113     #endif
00114
00115     // Checking the file
00116     if (!template)
00117         goto optimize_input_end;
00118
00119     // Opening template
00120     content = g_mapped_file_get_contents (template);
00121     length = g_mapped_file_get_length (template);
00122     #if DEBUG_OPTIMIZE
00123     fprintf (stderr, "optimize_input: length=%lu\ncontent:\n%s", length, content);
00124     #endif
00125     file = g_fopen (input, "w");
00126
00127     // Parsing template
00128     for (i = 0; i < optimize->nvariables; ++i)
00129     {
00130         #if DEBUG_OPTIMIZE
00131         fprintf (stderr, "optimize_input: variable=%u\n", i);
00132         #endif
00133         snprintf (buffer, 32, "@variable%u@", i + 1);
00134         regex = g_regex_new (buffer, 0, 0, NULL);
00135         if (i == 0)
00136         {
00137             buffer2 = g_regex_replace_literal (regex, content, length, 0,
00138                                             optimize->label[i], 0, NULL);
00139             #if DEBUG_OPTIMIZE
00140             fprintf (stderr, "optimize_input: buffer2\n%s", buffer2);
00141             #endif
00142         }
00143         else
00144         {
00145             length = strlen (buffer3);
00146             buffer2 = g_regex_replace_literal (regex, buffer3, length, 0,
00147                                             optimize->label[i], 0, NULL);
00148             g_free (buffer3);
00149         }
00150         g_regex_unref (regex);
00151         length = strlen (buffer2);
00152         snprintf (buffer, 32, "@value%u@", i + 1);
00153         regex = g_regex_new (buffer, 0, 0, NULL);
00154         snprintf (value, 32, format[optimize->precision[i]],
00155                 optimize->value[simulation * optimize->
00156 nvariables + i]);
00157         #if DEBUG_OPTIMIZE
00158         fprintf (stderr, "optimize_input: value=%s\n", value);
00159         #endif
00160         buffer3 = g_regex_replace_literal (regex, buffer2, length, 0, value,
00161                                         0, NULL);
00162         g_free (buffer2);
00163         g_regex_unref (regex);
00164     }
00165
00166     // Saving input file
00167     fwrite (buffer3, strlen (buffer3), sizeof (char), file);
00168     g_free (buffer3);
00169     fclose (file);
00170
00171 optimize_input_end:
00172     #if DEBUG_OPTIMIZE
00173     fprintf (stderr, "optimize_input: end\n");
00174     #endif
00175     return;
00176 }

```

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

Function to merge the 2 optimization results.

Parameters

<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 585 of file [optimize.c](#).

```

00587 {
00588     unsigned int i, j, k, s[optimize->nbest];
00589     double e[optimize->nbest];
00590     #if DEBUG_OPTIMIZE
00591     fprintf (stderr, "optimize_merge: start\n");
00592     #endif
00593     i = j = k = 0;
00594     do
00595     {
00596         if (i == optimize->nsaveds)
00597         {
00598             s[k] = simulation_best[j];
00599             e[k] = error_best[j];
00600             ++j;
00601             ++k;
00602             if (j == nsaveds)
00603                 break;
00604         }
00605         else if (j == nsaveds)
00606         {
00607             s[k] = optimize->simulation_best[i];
00608             e[k] = optimize->error_best[i];
00609             ++i;
00610             ++k;
00611             if (i == optimize->nsaveds)
00612                 break;
00613         }
00614         else if (optimize->error_best[i] > error_best[j])
00615         {
00616             s[k] = simulation_best[j];
00617             e[k] = error_best[j];
00618             ++j;
00619             ++k;
00620         }
00621         else
00622         {
00623             s[k] = optimize->simulation_best[i];
00624             e[k] = optimize->error_best[i];
00625             ++i;
00626             ++k;
00627         }
00628     }
00629     while (k < optimize->nbest);
00630     optimize->nsaveds = k;
00631     memcpy (optimize->simulation_best, s, k * sizeof (unsigned int));
00632     memcpy (optimize->error_best, e, k * sizeof (double));
00633     #if DEBUG_OPTIMIZE
00634     fprintf (stderr, "optimize_merge: end\n");
00635     #endif
00636 }

```

5.19.2.10 double optimize_norm_euclidian (unsigned int *simulation*)

Function to calculate the Euclidian error norm.

Parameters

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

Returns

Euclidian error norm.

Definition at line 295 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.19.2.11 double optimize_norm_maximum (unsigned int *simulation*)

Function to calculate the maximum error norm.

Parameters

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

Returns

Maximum error norm.

Definition at line 324 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.19.2.12 double optimize_norm_p (unsigned int *simulation*)

Function to calculate the P error norm.

Parameters

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

Returns

P error norm.

Definition at line 352 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.19.2.13 double optimize_norm_taxicab (unsigned int *simulation*)

Function to calculate the taxicab error norm.

Parameters

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

Returns

Taxicab error norm.

Definition at line 381 of file [optimize.c](#).

```

00382 {
00383     double e;
00384     unsigned int i;
00385     #if DEBUG_OPTIMIZE
00386     fprintf (stderr, "optimize_norm_taxicab: start\n");
00387     #endif
00388     e = 0.;
00389     for (i = 0; i < optimize->nexperiments; ++i)
00390     {
00391         e += fabs (optimize_parse (simulation, i));
00392     }
00393     #if DEBUG_OPTIMIZE
00394     fprintf (stderr, "optimize_norm_taxicab: error=%lg\n", e);
00395     fprintf (stderr, "optimize_norm_taxicab: end\n");
00396     #endif
00397     return e;
00398 }
```

Here is the call graph for this function:

5.19.2.14 double optimize_parse (unsigned int *simulation*, unsigned int *experiment*)

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

Parameters

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

Returns

Objective function value.

Definition at line 189 of file [optimize.c](#).

```

00190 {
00191     unsigned int i;
00192     double e;
00193     char buffer[512], input[MAX_NINPUTS][32], output[32], result[32], *buffer2,
00194         *buffer3, *buffer4;
00195     FILE *file_result;
00196
00197     #if DEBUG_OPTIMIZE
00198         fprintf (stderr, "optimize_parse: start\n");
00199         fprintf (stderr, "optimize_parse: simulation=%u experiment=%u\n", simulation,
00200             experiment);
00201     #endif
00202
00203     // Opening input files
00204     for (i = 0; i < optimize->ninputs; ++i)
00205     {
00206         snprintf (&input[i][0], 32, "input-%u-%u-%u", i, simulation, experiment);
00207         #if DEBUG_OPTIMIZE
00208             fprintf (stderr, "optimize_parse: i=%u input=%s\n", i, &input[i][0]);
00209         #endif
00210         optimize_input (simulation, &input[i][0], optimize->
00211             file[i][experiment]);
00212     }
00213     for (; i < MAX_NINPUTS; ++i)
00214         strcpy (&input[i][0], "");
00215     #if DEBUG_OPTIMIZE
00216         fprintf (stderr, "optimize_parse: parsing end\n");
00217     #endif
00218
00219     // Performing the simulation
00220     snprintf (output, 32, "output-%u-%u", simulation, experiment);
00221     buffer2 = g_path_get_dirname (optimize->simulator);
00222     buffer3 = g_path_get_basename (optimize->simulator);
00223     buffer4 = g_build_filename (buffer2, buffer3, NULL);
00224     snprintf (buffer, 512, "\"%s\" %s %s %s %s %s %s %s %s %s",
00225         buffer4, input[0], input[1], input[2], input[3], input[4], input[5],
00226         input[6], input[7], output);
00227     g_free (buffer4);
00228     g_free (buffer3);
00229     g_free (buffer2);
00230     #if DEBUG_OPTIMIZE
00231         fprintf (stderr, "optimize_parse: %s\n", buffer);
00232     #endif
00233     system (buffer);
00234
00235     // Checking the objective value function
00236     if (optimize->evaluator)
00237     {
00238         snprintf (result, 32, "result-%u-%u", simulation, experiment);
00239         buffer2 = g_path_get_dirname (optimize->evaluator);
00240         buffer3 = g_path_get_basename (optimize->evaluator);
00241         buffer4 = g_build_filename (buffer2, buffer3, NULL);
00242         snprintf (buffer, 512, "\"%s\" %s %s %s",
00243             buffer4, output, optimize->experiment[experiment], result);
00244         g_free (buffer4);
00245         g_free (buffer3);
00246         g_free (buffer2);
00247         #if DEBUG_OPTIMIZE
00248             fprintf (stderr, "optimize_parse: %s\n", buffer);
00249         #endif
00250     }
00251 }
```

```

00249     system (buffer);
00250     file_result = g_fopen (result, "r");
00251     e = atof (fgets (buffer, 512, file_result));
00252     fclose (file_result);
00253 }
00254 else
00255 {
00256     strcpy (result, "");
00257     file_result = g_fopen (output, "r");
00258     e = atof (fgets (buffer, 512, file_result));
00259     fclose (file_result);
00260 }
00261
00262 // Removing files
00263 #if !DEBUG_OPTIMIZE
00264 for (i = 0; i < optimize->ninputs; ++i)
00265 {
00266     if (optimize->file[i][0])
00267     {
00268         snprintf (buffer, 512, RM " %s", &input[i][0]);
00269         system (buffer);
00270     }
00271 }
00272 snprintf (buffer, 512, RM " %s %s", output, result);
00273 system (buffer);
00274 #endif
00275
00276 // Processing pending events
00277 show_pending ();
00278
00279 #if DEBUG_OPTIMIZE
00280 fprintf (stderr, "optimize_parse: end\n");
00281 #endif
00282
00283 // Returning the objective function
00284 return e * optimize->weight[experiment];
00285 }

```

Here is the call graph for this function:

5.19.2.15 void optimize_save_variables (unsigned int *simulation*, double *error*)

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

Parameters

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

Definition at line 434 of file [optimize.c](#).

```

00435 {
00436     unsigned int i;
00437     char buffer[64];
00438     #if DEBUG_OPTIMIZE
00439     fprintf (stderr, "optimize_save_variables: start\n");
00440     #endif
00441     for (i = 0; i < optimize->nvariables; ++i)
00442     {
00443         snprintf (buffer, 64, "%s ", format[optimize->precision[i]]);
00444         fprintf (optimize->file_variables, buffer,
00445             optimize->value[simulation * optimize->
00446                 nvariables + i]);
00447         fprintf (optimize->file_variables, "%.14le\n", error);
00448     }
00449     #if DEBUG_OPTIMIZE
00450     fprintf (stderr, "optimize_save_variables: end\n");
00451     #endif
00452 }

```

5.19.2.16 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 960 of file [optimize.c](#).

```

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

```

Here is the call graph for this function:

5.19.2.17 void* optimize_thread (ParallelData * data)

Function to optimize on a thread.

Parameters

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

Returns

NULL

Definition at line 539 of file [optimize.c](#).

```

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

Here is the call graph for this function:

5.20 optimize.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier 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;
00058     double *error_old;
00059     unsigned int *precision;
00060     unsigned int *nsweeps;
00061     unsigned int *nbits;
00062     unsigned int *thread;
00063     unsigned int *thread_direction;
00064     unsigned int *simulation_best;
00065     double tolerance;
00066     double mutation_ratio;
00067     double reproduction_ratio;
00068     double adaptation_ratio;
00069     double relaxation;
00070     double calculation_time;
00071     double p;
00072     double threshold;
00073     unsigned long int seed;
00074     unsigned int nvariables;
00075     unsigned int nexperiments;
00076     unsigned int ninputs;
00077     unsigned int nsimulations;
00078     unsigned int nsteps;
00079     unsigned int nestimates;
00080     unsigned int algorithm;
00081     unsigned int nstart;
00082     unsigned int nend;
00083     unsigned int nstart_direction;
00084     unsigned int nend_direction;
00085     unsigned int niterations;
00086     unsigned int nbest;
00087     unsigned int nsaveds;
00088     unsigned int stop;
00089 #if HAVE_MPI
00090     int mpi_rank;
00091 #endif
00092 } Optimize;
00093
00094 typedef struct
00095 {
00096     unsigned int thread;
00097 } ParallelData;
00098
00099 // Global variables
00100 extern int ntasks;
00101 extern unsigned int nthreads;
00102 extern unsigned int nthreads_direction;
00103 extern GMutex mutex[1];
00104 extern void (*optimize_algorithm) ();

```

```

00133 extern double (*optimize_estimate_direction) (unsigned int variable,
00134                                                unsigned int estimate);
00135 extern double (*optimize_norm) (unsigned int simulation);
00136 extern Optimize optimize[1];
00137
00138 // Public functions
00139 void optimize_input (unsigned int simulation, char *input,
00140                    GMappedFile * template);
00141 double optimize_parse (unsigned int simulation, unsigned int experiment);
00142 double optimize_norm_euclidian (unsigned int simulation);
00143 double optimize_norm_maximum (unsigned int simulation);
00144 double optimize_norm_p (unsigned int simulation);
00145 double optimize_norm_taxicab (unsigned int simulation);
00146 void optimize_print ();
00147 void optimize_save_variables (unsigned int simulation, double error);
00148 void optimize_best (unsigned int simulation, double value);
00149 void optimize_sequential ();
00150 void *optimize_thread (ParallelData * data);
00151 void optimize_merge (unsigned int nsaveds, unsigned int *simulation_best,
00152                    double *error_best);
00153 #if HAVE_MPI
00154 void optimize_synchronise ();
00155 #endif
00156 void optimize_sweep ();
00157 void optimize_MonteCarlo ();
00158 void optimize_best_direction (unsigned int simulation, double value);
00159 void optimize_direction_sequential (unsigned int simulation);
00160 void *optimize_direction_thread (ParallelData * data);
00161 double optimize_estimate_direction_random (unsigned int variable,
00162                                           unsigned int estimate);
00163 double optimize_estimate_direction_coordinates (unsigned int
variable,
00164                                                unsigned int estimate);
00165 void optimize_step_direction (unsigned int simulation);
00166 void optimize_direction ();
00167 double optimize_genetic_objective (Entity * entity);
00168 void optimize_genetic ();
00169 void optimize_save_old ();
00170 void optimize_merge_old ();
00171 void optimize_refine ();
00172 void optimize_step ();
00173 void optimize_iterate ();
00174 void optimize_free ();
00175 void optimize_open ();
00176
00177 #endif

```

5.21 utils.c File Reference

Source file to define some useful functions.

```

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

```

Include dependency graph for utils.c:

Functions

- void [show_pending](#) ()
Function to show events on long computation.
- void [show_message](#) (char *title, char *msg, int type)
Function to show a dialog with a message.
- void [show_error](#) (char *msg)

- Function to show a dialog with an error message.*

 - int [xml_node_get_int](#) (xmlNode *node, const xmlChar *prop, int *error_code)

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

 - unsigned int [xml_node_get_uint](#) (xmlNode *node, const xmlChar *prop, int *error_code)

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

 - unsigned int [xml_node_get_uint_with_default](#) (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)

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

 - double [xml_node_get_float](#) (xmlNode *node, const xmlChar *prop, int *error_code)

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

 - double [xml_node_get_float_with_default](#) (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)

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

 - void [xml_node_set_int](#) (xmlNode *node, const xmlChar *prop, int value)

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

 - void [xml_node_set_uint](#) (xmlNode *node, const xmlChar *prop, unsigned int value)

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

 - void [xml_node_set_float](#) (xmlNode *node, const xmlChar *prop, double value)

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

 - int [json_object_get_int](#) (JsonObject *object, const char *prop, int *error_code)

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

 - unsigned int [json_object_get_uint](#) (JsonObject *object, const char *prop, int *error_code)

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

 - unsigned int [json_object_get_uint_with_default](#) (JsonObject *object, const char *prop, unsigned int default_value, int *error_code)

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

 - double [json_object_get_float](#) (JsonObject *object, const char *prop, int *error_code)

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

 - double [json_object_get_float_with_default](#) (JsonObject *object, const char *prop, double default_value, int *error_code)

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

 - void [json_object_set_int](#) (JsonObject *object, const char *prop, int value)

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

 - void [json_object_set_uint](#) (JsonObject *object, const char *prop, unsigned int value)

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

 - void [json_object_set_float](#) (JsonObject *object, const char *prop, double value)

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

 - int [cores_number](#) ()

Function to obtain the cores number.

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

Function to get the active GtkRadioButton.

Variables

- GtkWidget * [main_window](#)
- Main GtkWidget.*
- char * [error_message](#)
- Error message.*

5.21.1 Detailed Description

Source file to define some useful functions.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.21.2 Function Documentation

5.21.2.1 int cores_number ()

Function to obtain the cores number.

Returns

Cores number.

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

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

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

Function to get the active GtkWidget.

Parameters

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

Returns

Active GtkWidget.

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

```

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

```

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

```

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

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

Parameters

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

Returns

Floating point number value.

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

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

Here is the call graph for this function:

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

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

Parameters

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

Returns

Integer number value.

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

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

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

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

Parameters

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

Returns

Unsigned integer number value.

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

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

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

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

Parameters

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

Returns

Unsigned integer number value.

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

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

Here is the call graph for this function:

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

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

Parameters

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

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

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

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

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

Parameters

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

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

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

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

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

Parameters

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

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

```
00510 {
```

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

5.21.2.11 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

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

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

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

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

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

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

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

Here is the call graph for this function:

5.21.2.15 int xml_node_get_int (xmlDoc * node, const xmlChar * prop, int * error_code)

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

Parameters

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

Returns

Integer number value.

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

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

5.21.2.16 int xml_node_get_uint (xmlDoc * node, const xmlChar * prop, int * error_code)

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

Parameters

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

Returns

Unsigned integer number value.

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

```
00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
```

```

00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }

```

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

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

Parameters

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

Returns

Unsigned integer number value.

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

```

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

```

Here is the call graph for this function:

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

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

Parameters

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

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

```

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

```

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

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

Parameters

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

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

```

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

```

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

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

Parameters

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

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

```

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

```

5.22 utils.c

```

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

```

```

00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <unistd.h>
00036 #include <libxml/parser.h>
00037 #include <libintl.h>
00038 #include <glib.h>
00039 #include <json-glib/json-glib.h>
00040 #ifdef G_OS_WIN32
00041 #include <windows.h>
00042 #endif
00043 #if HAVE_GTK
00044 #include <gtk/gtk.h>
00045 #endif
00046 #include "utils.h"
00047
00048 #if HAVE_GTK
00049 GtkWidget *main_window;
00050 #endif
00051
00052 char *error_message;
00053
00054 void
00055 show_pending ()
00056 {
00057     #if HAVE_GTK
00058     while (gtk_events_pending ())
00059         gtk_main_iteration ();
00060     #endif
00061 }
00062
00063 void
00064 show_message (char *title, char *msg, int type)
00065 {
00066     #if HAVE_GTK
00067     GtkMessageDialog *dlg;
00068
00069     // Creating the dialog
00070     dlg = (GtkMessageDialog *) gtk_message_dialog_new
00071         (main_window, GTK_DIALOG_MODAL, type, GTK_BUTTONS_OK, "%s", msg);
00072
00073     // Setting the dialog title
00074     gtk_window_set_title (GTK_WINDOW (dlg), title);
00075
00076     // Showing the dialog and waiting response
00077     gtk_dialog_run (GTK_DIALOG (dlg));
00078
00079     // Closing and freeing memory
00080     gtk_widget_destroy (GTK_WIDGET (dlg));
00081
00082 #else
00083     printf ("%s: %s\n", title, msg);
00084 #endif
00085 }
00086
00087 void
00088 show_error (char *msg)
00089 {
00090     show_message (gettext ("ERROR!"), msg, ERROR_TYPE);
00091 }
00092
00093 int
00094 xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code)
00095 {
00096     int i = 0;

```

```

00135     xmlChar *buffer;
00136     buffer = xmlGetProp (node, prop);
00137     if (!buffer)
00138         *error_code = 1;
00139     else
00140     {
00141         if (sscanf ((char *) buffer, "%d", &i) != 1)
00142             *error_code = 2;
00143         else
00144             *error_code = 0;
00145         xmlFree (buffer);
00146     }
00147     return i;
00148 }
00149
00162 unsigned int
00163 xml_node_get_uint (xmlNode * node, const xmlChar * prop, int *error_code)
00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }
00180
00196 unsigned int
00197 xml_node_get_uint_with_default (xmlNode * node, const xmlChar * prop,
00198                                unsigned int default_value, int *error_code)
00199 {
00200     unsigned int i;
00201     if (xmlHasProp (node, prop))
00202         i = xml_node_get_uint (node, prop, error_code);
00203     else
00204     {
00205         i = default_value;
00206         *error_code = 0;
00207     }
00208     return i;
00209 }
00210
00223 double
00224 xml_node_get_float (xmlNode * node, const xmlChar * prop, int *error_code)
00225 {
00226     double x = 0.;
00227     xmlChar *buffer;
00228     buffer = xmlGetProp (node, prop);
00229     if (!buffer)
00230         *error_code = 1;
00231     else
00232     {
00233         if (sscanf ((char *) buffer, "%lf", &x) != 1)
00234             *error_code = 2;
00235         else
00236             *error_code = 0;
00237         xmlFree (buffer);
00238     }
00239     return x;
00240 }
00241
00257 double
00258 xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00259                                 double default_value, int *error_code)
00260 {
00261     double x;
00262     if (xmlHasProp (node, prop))
00263         x = xml_node_get_float (node, prop, error_code);
00264     else
00265     {
00266         x = default_value;
00267         *error_code = 0;
00268     }
00269     return x;
00270 }
00271
00282 void
00283 xml_node_set_int (xmlNode * node, const xmlChar * prop, int value)
00284 {
00285     xmlChar buffer[64];

```

```
00286     snprintf ((char *) buffer, 64, "%d", value);
00287     xmlSetProp (node, prop, buffer);
00288 }
00289
00301 void
00302 xml_node_set_uint (xmlNode * node, const xmlChar * prop, unsigned int value)
00303 {
00304     xmlChar buffer[64];
00305     snprintf ((char *) buffer, 64, "%u", value);
00306     xmlSetProp (node, prop, buffer);
00307 }
00308
00320 void
00321 xml_node_set_float (xmlNode * node, const xmlChar * prop, double value)
00322 {
00323     xmlChar buffer[64];
00324     snprintf ((char *) buffer, 64, "%.14lg", value);
00325     xmlSetProp (node, prop, buffer);
00326 }
00327
00340 int
00341 json_object_get_int (JsonObject * object, const char *prop, int *error_code)
00342 {
00343     const char *buffer;
00344     int i = 0;
00345     buffer = json_object_get_string_member (object, prop);
00346     if (!buffer)
00347         *error_code = 1;
00348     else
00349     {
00350         if (sscanf (buffer, "%d", &i) != 1)
00351             *error_code = 2;
00352         else
00353             *error_code = 0;
00354     }
00355     return i;
00356 }
00357
00370 unsigned int
00371 json_object_get_uint (JsonObject * object, const char *prop, int *error_code)
00372 {
00373     const char *buffer;
00374     unsigned int i = 0;
00375     buffer = json_object_get_string_member (object, prop);
00376     if (!buffer)
00377         *error_code = 1;
00378     else
00379     {
00380         if (sscanf (buffer, "%u", &i) != 1)
00381             *error_code = 2;
00382         else
00383             *error_code = 0;
00384     }
00385     return i;
00386 }
00387
00403 unsigned int
00404 json_object_get_uint_with_default (JsonObject * object, const char *prop,
00405                                   unsigned int default_value, int *error_code)
00406 {
00407     unsigned int i;
00408     if (json_object_get_member (object, prop))
00409         i = json_object_get_uint (object, prop, error_code);
00410     else
00411     {
00412         i = default_value;
00413         *error_code = 0;
00414     }
00415     return i;
00416 }
00417
00430 double
00431 json_object_get_float (JsonObject * object, const char *prop, int *error_code)
00432 {
00433     const char *buffer;
00434     double x = 0.;
00435     buffer = json_object_get_string_member (object, prop);
00436     if (!buffer)
00437         *error_code = 1;
00438     else
00439     {
00440         if (sscanf (buffer, "%lf", &x) != 1)
00441             *error_code = 2;
00442         else
00443             *error_code = 0;
00444     }
00445     return x;
}
```

```

00446 }
00447
00463 double
00464 json_object_get_float_with_default (JsonObject * object, const char *prop
00465                                     double default_value, int *error_code)
00466 {
00467     double x;
00468     if (json_object_get_member (object, prop))
00469         x = json_object_get_float (object, prop, error_code);
00470     else
00471     {
00472         x = default_value;
00473         *error_code = 0;
00474     }
00475     return x;
00476 }
00477
00489 void
00490 json_object_set_int (JsonObject * object, const char *prop, int value)
00491 {
00492     char buffer[64];
00493     snprintf (buffer, 64, "%d", value);
00494     json_object_set_string_member (object, prop, buffer);
00495 }
00496
00508 void
00509 json_object_set_uint (JsonObject * object, const char *prop, unsigned int value)
00510 {
00511     char buffer[64];
00512     snprintf (buffer, 64, "%u", value);
00513     json_object_set_string_member (object, prop, buffer);
00514 }
00515
00527 void
00528 json_object_set_float (JsonObject * object, const char *prop, double value)
00529 {
00530     char buffer[64];
00531     snprintf (buffer, 64, "%.14lg", value);
00532     json_object_set_string_member (object, prop, buffer);
00533 }
00534
00540 int
00541 cores_number ()
00542 {
00543     #ifdef G_OS_WIN32
00544         SYSTEM_INFO sysinfo;
00545         GetSystemInfo (&sysinfo);
00546         return sysinfo.dwNumberOfProcessors;
00547     #else
00548         return (int) sysconf (_SC_NPROCESSORS_ONLN);
00549     #endif
00550 }
00551
00552 #if HAVE_GTK
00553
00564 unsigned int
00565 gtk_array_get_active (GtkRadioButton * array[], unsigned int n)
00566 {
00567     unsigned int i;
00568     for (i = 0; i < n; ++i)
00569         if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (array[i])))
00570             break;
00571     return i;
00572 }
00573
00574 #endif

```

5.23 utils.h File Reference

Header file to define some useful functions.

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

Macros

- #define `ERROR_TYPE` `GTK_MESSAGE_ERROR`

Macro to define the error message type.

- `#define INFO_TYPE GTK_MESSAGE_INFO`

Macro to define the information message type.

Functions

- void `show_pending` ()
Function to show events on long computation.
- void `show_message` (char *title, char *msg, int type)
Function to show a dialog with a message.
- void `show_error` (char *msg)
Function to show a dialog with an error message.
- int `xml_node_get_int` (xmlNode *node, const xmlChar *prop, int *error_code)
Function to get an integer number of a XML node property.
- unsigned int `xml_node_get_uint` (xmlNode *node, const xmlChar *prop, int *error_code)
Function to get an unsigned integer number of a XML node property.
- unsigned int `xml_node_get_uint_with_default` (xmlNode *node, const xmlChar *prop, unsigned int default_value, int *error_code)
Function to get an unsigned integer number of a XML node property with a default value.
- double `xml_node_get_float` (xmlNode *node, const xmlChar *prop, int *error_code)
Function to get a floating point number of a XML node property.
- double `xml_node_get_float_with_default` (xmlNode *node, const xmlChar *prop, double default_value, int *error_code)
Function to get a floating point number of a XML node property with a default value.
- void `xml_node_set_int` (xmlNode *node, const xmlChar *prop, int value)
Function to set an integer number in a XML node property.
- void `xml_node_set_uint` (xmlNode *node, const xmlChar *prop, unsigned int value)
Function to set an unsigned integer number in a XML node property.
- void `xml_node_set_float` (xmlNode *node, const xmlChar *prop, double value)
Function to set a floating point number in a XML node property.
- int `json_object_get_int` (JsonObject *object, const char *prop, int *error_code)
Function to get an integer number of a JSON object property.
- unsigned int `json_object_get_uint` (JsonObject *object, const char *prop, int *error_code)
Function to get an unsigned integer number of a JSON object property.
- unsigned int `json_object_get_uint_with_default` (JsonObject *object, const char *prop, unsigned int default_value, int *error_code)
Function to get an unsigned integer number of a JSON object property with a default value.
- double `json_object_get_float` (JsonObject *object, const char *prop, int *error_code)
Function to get a floating point number of a JSON object property.
- double `json_object_get_float_with_default` (JsonObject *object, const char *prop, double default_value, int *error_code)
Function to get a floating point number of a JSON object property with a default value.
- void `json_object_set_int` (JsonObject *object, const char *prop, int value)
Function to set an integer number in a JSON object property.
- void `json_object_set_uint` (JsonObject *object, const char *prop, unsigned int value)
Function to set an unsigned integer number in a JSON object property.
- void `json_object_set_float` (JsonObject *object, const char *prop, double value)
Function to set a floating point number in a JSON object property.
- int `cores_number` ()
Function to obtain the cores number.
- unsigned int `gtk_array_get_active` (GtkRadioButton *array[], unsigned int n)
Function to get the active GtkRadioButton.

Variables

- GtkWidget * [main_window](#)
Main GtkWidget.
- char * [error_message](#)
Error message.

5.23.1 Detailed Description

Header file to define some useful functions.

Authors

Javier Burguete.

Copyright

Copyright 2012-2016, all rights reserved.

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

5.23.2 Function Documentation

5.23.2.1 int cores_number ()

Function to obtain the cores number.

Returns

Cores number.

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

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

5.23.2.2 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 565 of file [utils.c](#).

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

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

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

Parameters

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

Returns

Floating point number value.

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

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

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

Here is the call graph for this function:

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

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

Parameters

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

Returns

Integer number value.

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

```

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

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

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

Parameters

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

Returns

Unsigned integer number value.

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

```

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

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

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

Parameters

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

Returns

Unsigned integer number value.

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

```

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

Here is the call graph for this function:

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

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

Parameters

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

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

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

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

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

Parameters

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

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

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

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

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

Parameters

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

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

```
00510 {
```

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

5.23.2.11 void show_error (char * *msg*)

Function to show a dialog with an error message.

Parameters

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

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

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

Here is the call graph for this function:

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

Function to show a dialog with a message.

Parameters

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

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

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

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

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

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

Parameters

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

Returns

Floating point number value.

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

```

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

Here is the call graph for this function:

5.23.2.15 int xml_node_get_int (xmlDoc * node, const xmlChar * prop, int * error_code)

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

Parameters

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

Returns

Integer number value.

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

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

5.23.2.16 unsigned int xml_node_get_uint (xmlDoc * node, const xmlChar * prop, int * error_code)

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

Parameters

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

Returns

Unsigned integer number value.

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

```
00164 {
00165     unsigned int i = 0;
00166     xmlChar *buffer;
00167     buffer = xmlGetProp (node, prop);
00168     if (!buffer)
00169         *error_code = 1;
00170     else
```

```

00171     {
00172         if (sscanf ((char *) buffer, "%u", &i) != 1)
00173             *error_code = 2;
00174         else
00175             *error_code = 0;
00176         xmlFree (buffer);
00177     }
00178     return i;
00179 }

```

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

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

Parameters

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

Returns

Unsigned integer number value.

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

```

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

```

Here is the call graph for this function:

5.23.2.18 void xml_node_set_float (xmlDoc * node, const xmlChar * prop, double value)

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

Parameters

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

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

```

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

```

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

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

Parameters

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

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

```

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

```

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

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

Parameters

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

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

```

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

```

5.24 utils.h

```

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

```

```

00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014         this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017         this list of conditions and the following disclaimer in the
00018         documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef UTILS__H
00033 #define UTILS__H 1
00034
00035 #if HAVE_GTK
00036 #define ERROR_TYPE GTK_MESSAGE_ERROR
00037 #define INFO_TYPE GTK_MESSAGE_INFO
00038 extern GtkWidget *main_window;
00039 #else
00040 #define ERROR_TYPE 0
00041 #define INFO_TYPE 0
00042 #endif
00043
00044 extern char *error_message;
00045
00046 // Public functions
00047 void show_pending ();
00048 void show_message (char *title, char *msg, int type);
00049 void show_error (char *msg);
00050 int xml_node_get_int (xmlNode * node, const xmlChar * prop, int *error_code);
00051 unsigned int xml_node_get_uint (xmlNode * node, const xmlChar * prop,
00052                                int *error_code);
00053 unsigned int xml_node_get_uint_with_default (xmlNode * node,
00054                                             const xmlChar * prop,
00055                                             unsigned int default_value,
00056                                             int *error_code);
00057 double xml_node_get_float (xmlNode * node, const xmlChar * prop,
00058                           int *error_code);
00059 double xml_node_get_float_with_default (xmlNode * node, const xmlChar * prop,
00060                                        double default_value, int *error_code);
00061 void xml_node_set_int (xmlNode * node, const xmlChar * prop, int value);
00062 void xml_node_set_uint (xmlNode * node, const xmlChar * prop,
00063                        unsigned int value);
00064 void xml_node_set_float (xmlNode * node, const xmlChar * prop, double value);
00065 int json_object_get_int (JsonObject * object, const char *prop,
00066                        int *error_code);
00067 unsigned int json_object_get_uint (JsonObject * object, const char *prop,
00068                                   int *error_code);
00069 unsigned int json_object_get_uint_with_default (JsonObject * object,
00070                                                const char *prop,
00071                                                unsigned int default_value,
00072                                                int *error_code);
00073 double json_object_get_float (JsonObject * object, const char *prop,
00074                              int *error_code);
00075 double json_object_get_float_with_default (JsonObject * object,
00076                                           const char *prop,
00077                                           double default_value,
00078                                           int *error_code);
00079 void json_object_set_int (JsonObject * object, const char *prop, int value);
00080 void json_object_set_uint (JsonObject * object, const char *prop,
00081                          unsigned int value);
00082 void json_object_set_float (JsonObject * object, const char *prop,
00083                          double value);
00084 int cores_number ();
00085 #if HAVE_GTK
00086 unsigned int gtk_array_get_active (GtkRadioButton * array[], unsigned int n);
00087 #endif
00088 #endif

```

5.25 variable.c File Reference

Source file to define the variable data.

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

Include dependency graph for variable.c:

Macros

- `#define _GNU_SOURCE`
- `#define DEBUG_VARIABLE 0`
Macro to debug variable functions.

Functions

- void `variable_new` (`Variable *variable`)
Function to create a new `Variable` struct.
- void `variable_free` (`Variable *variable`, unsigned int type)
Function to free the memory of a `Variable` struct.
- void `variable_error` (`Variable *variable`, char *message)
Function to print a message error opening an `Variable` struct.
- int `variable_open_xml` (`Variable *variable`, xmlNode *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.
- int `variable_open_json` (`Variable *variable`, JsonNode *node, unsigned int algorithm, unsigned int nsteps)
Function to open the variable file.

Variables

- const char * `format` [`NPRECISIONS`]
Array of C-strings with variable formats.
- const double `precision` [`NPRECISIONS`]
Array of variable precisions.

5.25.1 Detailed Description

Source file to define the variable data.

Authors

Javier Burguete and Borja Latorre.

Copyright

Copyright 2012-2016, all rights reserved.

Definition in file `variable.c`.

5.25.2 Function Documentation

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

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

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	Error message.

Definition at line 110 of file [variable.c](#).

```

00111 {
00112     char buffer[64];
00113     if (!variable->name)
00114         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117                 message);
00118     error_message = g_strdup (buffer);
00119 }
```

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

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

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	Type of input file.

Definition at line 87 of file [variable.c](#).

```

00088 {
00089     #if DEBUG_VARIABLE
00090         fprintf (stderr, "variable_free: start\n");
00091     #endif
00092     if (type == INPUT_TYPE_XML)
00093         xmlFree (variable->name);
00094     else
00095         g_free (variable->name);
00096     #if DEBUG_VARIABLE
00097         fprintf (stderr, "variable_free: end\n");
00098     #endif
00099 }
```

5.25.2.3 void variable_new (Variable * variable)

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

Definition at line 67 of file [variable.c](#).

```

00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_new: start\n");
00071     #endif
00072     variable->name = NULL;
00073     #if DEBUG_VARIABLE
00074         fprintf (stderr, "variable_new: end\n");
00075     #endif
00076 }
```

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

Function to open the variable file.

Parameters

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

Returns

1 on success, 0 on error.

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

```

00304 {
00305     JsonObject *object;
00306     const char *label;
00307     int error_code;
00308     #if DEBUG_VARIABLE
00309         fprintf (stderr, "variable_open_json: start\n");
00310     #endif
00311     object = json_node_get_object (node);
00312     label = json_object_get_string_member (object, LABEL_NAME);
00313     if (!label)
00314     {
00315         variable_error (variable, gettext ("no name"));
00316         goto exit_on_error;
00317     }
00318     variable->name = g_strdup (label);
00319     if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321         variable->rangemin
00322             = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323         if (error_code)
00324         {
00325             variable_error (variable, gettext ("bad minimum"));
00326             goto exit_on_error;
00327         }
00328         variable->rangeminabs
00329             = json_object_get_float_with_default (object,
00330             LABEL_ABSOLUTE_MINIMUM,
00331             -G_MAXDOUBLE, &error_code);
00332         if (error_code)
00333         {
00334             variable_error (variable, gettext ("bad absolute minimum"));
00335             goto exit_on_error;
00336         }
00337         if (variable->rangemin < variable->rangeminabs)
00338         {
00339             variable_error (variable, gettext ("minimum range not allowed"));
00340             goto exit_on_error;
00341         }
00342     }
```

```

00341     }
00342     else
00343     {
00344         variable_error (variable, gettext ("no minimum range"));
00345         goto exit_on_error;
00346     }
00347     if (json_object_get_member (object, LABEL_MAXIMUM))
00348     {
00349         variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351         if (error_code)
00352         {
00353             variable_error (variable, gettext ("bad maximum"));
00354             goto exit_on_error;
00355         }
00356         variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
00358 LABEL_ABSOLUTE_MAXIMUM,
00359                                             G_MAXDOUBLE, &error_code);
00360         if (error_code)
00361         {
00362             variable_error (variable, gettext ("bad absolute maximum"));
00363             goto exit_on_error;
00364         }
00365         if (variable->rangemax > variable->rangemaxabs)
00366         {
00367             variable_error (variable, gettext ("maximum range not allowed"));
00368             goto exit_on_error;
00369         }
00370         if (variable->rangemax < variable->rangemin)
00371         {
00372             variable_error (variable, gettext ("bad range"));
00373             goto exit_on_error;
00374         }
00375     }
00376     else
00377     {
00378         variable_error (variable, gettext ("no maximum range"));
00379         goto exit_on_error;
00380     }
00381     variable->precision
00382     = json_object_get_uint_with_default (object,
00383 LABEL_PRECISION,
00384                                         DEFAULT_PRECISION, &error_code);
00385     if (error_code || variable->precision >= NPRECISIONS)
00386     {
00387         variable_error (variable, gettext ("bad precision"));
00388         goto exit_on_error;
00389     }
00390     if (algorithm == ALGORITHM_SWEEP)
00391     {
00392         if (json_object_get_member (object, LABEL_NSWEEPS))
00393         {
00394             variable->nsweeps
00395             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00396             if (error_code || !variable->nsweeps)
00397             {
00398                 variable_error (variable, gettext ("bad sweeps"));
00399                 goto exit_on_error;
00400             }
00401         }
00402         else
00403         {
00404             variable_error (variable, gettext ("no sweeps number"));
00405             goto exit_on_error;
00406         }
00407     }
00408     #if DEBUG_VARIABLE
00409     fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00410     #endif
00411     if (algorithm == ALGORITHM_GENETIC)
00412     {
00413         // Obtaining bits representing each variable
00414         if (json_object_get_member (object, LABEL_NBITS))
00415         {
00416             variable->nbits
00417             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00418             if (error_code || !variable->nbits)
00419             {
00420                 variable_error (variable, gettext ("invalid bits number"));
00421                 goto exit_on_error;
00422             }
00423         }
00424         else
00425         {
00426             variable_error (variable, gettext ("no bits number"));
00427             goto exit_on_error;
00428         }
00429     }

```



```

00426     }
00427 }
00428 else if (nsteps)
00429 {
00430     variable->step = json_object_get_float (object,
00431 LABEL_STEP, &error_code);
00432     if (error_code || variable->step < 0.)
00433     {
00434         variable_error (variable, gettext ("bad step size"));
00435         goto exit_on_error;
00436     }
00437
00438 #if DEBUG_VARIABLE
00439     fprintf (stderr, "variable_open_json: end\n");
00440 #endif
00441     return 1;
00442 exit_on_error:
00443     variable_free (variable, INPUT_TYPE_JSON);
00444 #if DEBUG_VARIABLE
00445     fprintf (stderr, "variable_open_json: end\n");
00446 #endif
00447     return 0;
00448 }

```

Here is the call graph for this function:

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

Function to open the variable file.

Parameters

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

Returns

1 on success, 0 on error.

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

```

00138 {
00139     int error_code;
00140
00141 #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143 #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, gettext ("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154             = xml_node_get_float (node, (const xmlChar *)
00155 LABEL_MINIMUM,
00156                                     &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad minimum"));
00160             goto exit_on_error;
00161         }
00162     }
00163 }

```

```

00161     variable->rangeminabs = xml_node_get_float_with_default
00162         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00163         &error_code);
00164     if (error_code)
00165     {
00166         variable_error (variable, gettext ("bad absolute minimum"));
00167         goto exit_on_error;
00168     }
00169     if (variable->rangemin < variable->rangeminabs)
00170     {
00171         variable_error (variable, gettext ("minimum range not allowed"));
00172         goto exit_on_error;
00173     }
00174 }
00175 else
00176 {
00177     variable_error (variable, gettext ("no minimum range"));
00178     goto exit_on_error;
00179 }
00180 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181 {
00182     variable->rangemax
00183     = xml_node_get_float (node, (const xmlChar *)
00184 LABEL_MAXIMUM,
00185                             &error_code);
00186     if (error_code)
00187     {
00188         variable_error (variable, gettext ("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, gettext ("bad absolute maximum"));
00197         goto exit_on_error;
00198     }
00199     if (variable->rangemax > variable->rangemaxabs)
00200     {
00201         variable_error (variable, gettext ("maximum range not allowed"));
00202         goto exit_on_error;
00203     }
00204     if (variable->rangemax < variable->rangemin)
00205     {
00206         variable_error (variable, gettext ("bad range"));
00207         goto exit_on_error;
00208     }
00209 }
00210 else
00211 {
00212     variable_error (variable, gettext ("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, gettext ("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, gettext ("bad sweeps"));
00235             goto exit_on_error;
00236         }
00237     }
00238 }
00239 else
00240 {
00241     variable_error (variable, gettext ("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, gettext ("invalid bits number"));
00256                 goto exit_on_error;
00257             }
00258             else
00259             {
00260                 variable_error (variable, gettext ("no bits number"));
00261                 goto exit_on_error;
00262             }
00263         }
00264         else if (nsteps)
00265         {
00266             variable->step
00267             = xml_node_get_float (node, (const xmlChar *)
00268             LABEL_STEP, &error_code);
00269             if (error_code || variable->step < 0.)
00270             {
00271                 variable_error (variable, gettext ("bad step size"));
00272                 goto exit_on_error;
00273             }
00274         }
00275         #if DEBUG_VARIABLE
00276         fprintf (stderr, "variable_open_xml: end\n");
00277         #endif
00278         return 1;
00279     exit_on_error:
00280         variable_free (variable, INPUT_TYPE_XML);
00281         #if DEBUG_VARIABLE
00282         fprintf (stderr, "variable_open_xml: end\n");
00283         #endif
00284         return 0;
00285     }

```

Here is the call graph for this function:

5.25.3 Variable Documentation

5.25.3.1 const char* format[NPRECISIONS]

Initial value:

```

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

```

Array of C-strings with variable formats.

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

5.25.3.2 const double precision[NPRECISIONS]

Initial value:

```

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

```

Array of variable precisions.

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

5.26 variable.c

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #define _GNU_SOURCE
00033 #include "config.h"
00034 #include <stdio.h>
00035 #include <libxml/parser.h>
00036 #include <libintl.h>
00037 #include <glib.h>
00038 #include <json-glib/json-glib.h>
00039 #include "utils.h"
00040 #include "variable.h"
00041
00042 #define DEBUG_VARIABLE 0
00043
00044 const char *format[NPRECISIONS] = {
00045     "%.0lf", "%.1lf", "%.2lf", "%.3lf", "%.4lf", "%.5lf", "%.6lf", "%.7lf",
00046     "%.8lf", "%.9lf", "%.10lf", "%.11lf", "%.12lf", "%.13lf", "%.14lf"
00047 };
00048
00049 const double precision[NPRECISIONS] = {
00050     1., 0.1, 0.01, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12,
00051     1e-13, 1e-14
00052 };
00053
00054 void
00055 variable_new (Variable * variable)
00056 {
00057     #if DEBUG_VARIABLE
00058         fprintf (stderr, "variable_new: start\n");
00059     #endif
00060     variable->name = NULL;
00061     #if DEBUG_VARIABLE
00062         fprintf (stderr, "variable_new: end\n");
00063     #endif
00064 }
00065
00066 void
00067 variable_free (Variable * variable, unsigned int type)
00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_free: start\n");
00071     #endif
00072     if (type == INPUT_TYPE_XML)
00073         xmlFree (variable->name);
00074     else
00075         g_free (variable->name);
00076     #if DEBUG_VARIABLE
00077         fprintf (stderr, "variable_free: end\n");
00078     #endif
00079 }
00080
00081 void
00082 variable_error (Variable * variable, char *message)
00083 {
00084     char buffer[64];

```

```

00113     if (!variable->name)
00114         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117             message);
00118     error_message = g_strdup (buffer);
00119 }
00120
00135 int
00136 variable_open_xml (Variable * variable, xmlNode * node, unsigned int algorithm,
00137     unsigned int nsteps)
00138 {
00139     int error_code;
00140
00141     #if DEBUG_VARIABLE
00142         fprintf (stderr, "variable_open_xml: start\n");
00143     #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, gettext ("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154             = xml_node_get_float (node, (const xmlChar *)
00155             LABEL_MINIMUM,
00156                 &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad minimum"));
00160             goto exit_on_error;
00161         }
00162         variable->rangeminabs = xml_node_get_float_with_default
00163             (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00164                 &error_code);
00165         if (error_code)
00166         {
00167             variable_error (variable, gettext ("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170         if (variable->rangemin < variable->rangeminabs)
00171         {
00172             variable_error (variable, gettext ("minimum range not allowed"));
00173             goto exit_on_error;
00174         }
00175     }
00176     else
00177     {
00178         variable_error (variable, gettext ("no minimum range"));
00179         goto exit_on_error;
00180     }
00181     if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00182     {
00183         variable->rangemax
00184             = xml_node_get_float (node, (const xmlChar *)
00185             LABEL_MAXIMUM,
00186                 &error_code);
00187         if (error_code)
00188         {
00189             variable_error (variable, gettext ("bad maximum"));
00190             goto exit_on_error;
00191         }
00192         variable->rangemaxabs = xml_node_get_float_with_default
00193             (node, (const xmlChar *) LABEL_ABSOLUTE_MAXIMUM, G_MAXDOUBLE,
00194                 &error_code);
00195         if (error_code)
00196         {
00197             variable_error (variable, gettext ("bad absolute maximum"));
00198             goto exit_on_error;
00199         }
00200         if (variable->rangemax > variable->rangemaxabs)
00201         {
00202             variable_error (variable, gettext ("maximum range not allowed"));
00203             goto exit_on_error;
00204         }
00205         if (variable->rangemax < variable->rangemin)
00206         {
00207             variable_error (variable, gettext ("bad range"));
00208             goto exit_on_error;
00209         }
00210     }
00211     else
00212     {
00213         variable_error (variable, gettext ("no maximum range"));
00214     }

```

```

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

```

```

00309     fprintf (stderr, "variable_open_json: start\n");
00310 #endif
00311     object = json_node_get_object (node);
00312     label = json_object_get_string_member (object, LABEL_NAME);
00313     if (!label)
00314     {
00315         variable_error (variable, gettext ("no name"));
00316         goto exit_on_error;
00317     }
00318     variable->name = g_strdup (label);
00319     if (json_object_get_member (object, LABEL_MINIMUM))
00320     {
00321         variable->rangemin
00322         = json_object_get_float (object, LABEL_MINIMUM, &error_code);
00323         if (error_code)
00324         {
00325             variable_error (variable, gettext ("bad minimum"));
00326             goto exit_on_error;
00327         }
00328         variable->rangeminabs
00329         = json_object_get_float_with_default (object,
00330 LABEL_ABSOLUTE_MINIMUM,
00331                                             -G_MAXDOUBLE, &error_code);
00332         if (error_code)
00333         {
00334             variable_error (variable, gettext ("bad absolute minimum"));
00335             goto exit_on_error;
00336         }
00337         if (variable->rangemin < variable->rangeminabs)
00338         {
00339             variable_error (variable, gettext ("minimum range not allowed"));
00340             goto exit_on_error;
00341         }
00342     }
00343     else
00344     {
00345         variable_error (variable, gettext ("no minimum range"));
00346         goto exit_on_error;
00347     }
00348     if (json_object_get_member (object, LABEL_MAXIMUM))
00349     {
00350         variable->rangemax
00351         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00352         if (error_code)
00353         {
00354             variable_error (variable, gettext ("bad maximum"));
00355             goto exit_on_error;
00356         }
00357         variable->rangemaxabs
00358         = json_object_get_float_with_default (object,
00359 LABEL_ABSOLUTE_MAXIMUM,
00360                                             G_MAXDOUBLE, &error_code);
00361         if (error_code)
00362         {
00363             variable_error (variable, gettext ("bad absolute maximum"));
00364             goto exit_on_error;
00365         }
00366         if (variable->rangemax > variable->rangemaxabs)
00367         {
00368             variable_error (variable, gettext ("maximum range not allowed"));
00369             goto exit_on_error;
00370         }
00371         if (variable->rangemax < variable->rangemin)
00372         {
00373             variable_error (variable, gettext ("bad range"));
00374             goto exit_on_error;
00375         }
00376     }
00377     else
00378     {
00379         variable_error (variable, gettext ("no maximum range"));
00380         goto exit_on_error;
00381     }
00382     variable->precision
00383     = json_object_get_uint_with_default (object,
00384 LABEL_PRECISION,
00385                                         DEFAULT_PRECISION, &error_code);
00386     if (error_code || variable->precision >= NPRECISIONS)
00387     {
00388         variable_error (variable, gettext ("bad precision"));
00389         goto exit_on_error;
00390     }
00391     if (algorithm == ALGORITHM_SWEEP)
00392     {
00393         if (json_object_get_member (object, LABEL_NSWEEPS))
00394         {
00395             variable->nsweeps

```

```

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

```

5.27 variable.h File Reference

Header file to define the variable data.

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

Data Structures

- struct [Variable](#)
Struct to define the variable data.

Enumerations

- enum [Algorithm](#) { [ALGORITHM_MONTE_CARLO](#) = 0, [ALGORITHM_SWEEP](#) = 1, [ALGORITHM_GENETIC](#) = 2 }
Enum to define the algorithms.

Functions

- void [variable_new](#) ([Variable](#) *variable)
Function to create a new [Variable](#) struct.
- void [variable_free](#) ([Variable](#) *variable, unsigned int type)
Function to free the memory of a [Variable](#) struct.
- void [variable_error](#) ([Variable](#) *variable, char *message)
Function to print a message error opening an [Variable](#) struct.
- int [variable_open_xml](#) ([Variable](#) *variable, 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 enum Algorithm

Enum to define the algorithms.

Enumerator

ALGORITHM_MONTE_CARLO Monte-Carlo algorithm.

ALGORITHM_SWEEP Sweep algorithm.

ALGORITHM_GENETIC Genetic algorithm.

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

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

5.27.3 Function Documentation

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

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

Parameters

<i>variable</i>	Variable struct.
<i>message</i>	Error message.

Definition at line 110 of file [variable.c](#).

```

00111 {
00112     char buffer[64];
00113     if (!variable->name)
00114         snprintf (buffer, 64, "%s: %s", gettext ("Variable"), message);
00115     else
00116         snprintf (buffer, 64, "%s %s: %s", gettext ("Variable"), variable->name,
00117                 message);
00118     error_message = g_strdup (buffer);
00119 }
```

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

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

Parameters

<i>variable</i>	Variable struct.
<i>type</i>	Type of input file.

Definition at line 87 of file [variable.c](#).

```

00088 {
00089     #if DEBUG_VARIABLE
00090         fprintf (stderr, "variable_free: start\n");
00091     #endif
00092     if (type == INPUT_TYPE_XML)
00093         xmlFree (variable->name);
00094     else
00095         g_free (variable->name);
00096     #if DEBUG_VARIABLE
00097         fprintf (stderr, "variable_free: end\n");
00098     #endif
00099 }
```

5.27.3.3 void variable_new (Variable * variable)

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

Parameters

<i>variable</i>	Variable struct.
-----------------	----------------------------------

Definition at line 67 of file [variable.c](#).

```

00068 {
00069     #if DEBUG_VARIABLE
00070         fprintf (stderr, "variable_new: start\n");
```

```

00071 #endif
00072     variable->name = NULL;
00073 #if DEBUG_VARIABLE
00074     fprintf (stderr, "variable_new: end\n");
00075 #endif
00076 }

```

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

Function to open the variable file.

Parameters

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

Returns

1 on success, 0 on error.

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

```

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

```

```

00348     {
00349         variable->rangemax
00350         = json_object_get_float (object, LABEL_MAXIMUM, &error_code);
00351         if (error_code)
00352         {
00353             variable_error (variable, gettext ("bad maximum"));
00354             goto exit_on_error;
00355         }
00356         variable->rangemaxabs
00357         = json_object_get_float_with_default (object,
00358 LABEL_ABSOLUTE_MAXIMUM,
00359                                             G_MAXDOUBLE, &error_code);
00359         if (error_code)
00360         {
00361             variable_error (variable, gettext ("bad absolute maximum"));
00362             goto exit_on_error;
00363         }
00364         if (variable->rangemax > variable->rangemaxabs)
00365         {
00366             variable_error (variable, gettext ("maximum range not allowed"));
00367             goto exit_on_error;
00368         }
00369         if (variable->rangemax < variable->rangemin)
00370         {
00371             variable_error (variable, gettext ("bad range"));
00372             goto exit_on_error;
00373         }
00374     }
00375     else
00376     {
00377         variable_error (variable, gettext ("no maximum range"));
00378         goto exit_on_error;
00379     }
00380     variable->precision
00381     = json_object_get_uint_with_default (object,
00382 LABEL_PRECISION,
00383                                         DEFAULT_PRECISION, &error_code);
00383     if (error_code || variable->precision >= NPRECISIONS)
00384     {
00385         variable_error (variable, gettext ("bad precision"));
00386         goto exit_on_error;
00387     }
00388     if (algorithm == ALGORITHM_SWEEP)
00389     {
00390         if (json_object_get_member (object, LABEL_NSWEEPS))
00391         {
00392             variable->nsweeps
00393             = json_object_get_uint (object, LABEL_NSWEEPS, &error_code);
00394             if (error_code || !variable->nsweeps)
00395             {
00396                 variable_error (variable, gettext ("bad sweeps"));
00397                 goto exit_on_error;
00398             }
00399         }
00400         else
00401         {
00402             variable_error (variable, gettext ("no sweeps number"));
00403             goto exit_on_error;
00404         }
00405         #if DEBUG_VARIABLE
00406         fprintf (stderr, "variable_open_json: nsweeps=%u\n", variable->nsweeps);
00407         #endif
00408     }
00409     if (algorithm == ALGORITHM_GENETIC)
00410     {
00411         // Obtaining bits representing each variable
00412         if (json_object_get_member (object, LABEL_NBITS))
00413         {
00414             variable->nbits
00415             = json_object_get_uint (object, LABEL_NBITS, &error_code);
00416             if (error_code || !variable->nbits)
00417             {
00418                 variable_error (variable, gettext ("invalid bits number"));
00419                 goto exit_on_error;
00420             }
00421         }
00422         else
00423         {
00424             variable_error (variable, gettext ("no bits number"));
00425             goto exit_on_error;
00426         }
00427     }
00428     else if (nsteps)
00429     {
00430         variable->step = json_object_get_float (object,
00431 LABEL_STEP, &error_code);
00431         if (error_code || variable->step < 0.)

```

```

00432     {
00433         variable_error (variable, gettext ("bad step size"));
00434         goto exit_on_error;
00435     }
00436 }
00437
00438 #if DEBUG_VARIABLE
00439 fprintf (stderr, "variable_open_json: end\n");
00440 #endif
00441 return 1;
00442 exit_on_error:
00443     variable_free (variable, INPUT_TYPE_JSON);
00444 #if DEBUG_VARIABLE
00445 fprintf (stderr, "variable_open_json: end\n");
00446 #endif
00447 return 0;
00448 }

```

Here is the call graph for this function:

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

Function to open the variable file.

Parameters

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

Returns

1 on success, 0 on error.

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

```

00138 {
00139     int error_code;
00140
00141     #if DEBUG_VARIABLE
00142     fprintf (stderr, "variable_open_xml: start\n");
00143     #endif
00144
00145     variable->name = (char *) xmlGetProp (node, (const xmlChar *) LABEL_NAME);
00146     if (!variable->name)
00147     {
00148         variable_error (variable, gettext ("no name"));
00149         goto exit_on_error;
00150     }
00151     if (xmlHasProp (node, (const xmlChar *) LABEL_MINIMUM))
00152     {
00153         variable->rangemin
00154         = xml_node_get_float (node, (const xmlChar *)
00155         LABEL_MINIMUM,
00156         &error_code);
00157         if (error_code)
00158         {
00159             variable_error (variable, gettext ("bad minimum"));
00160             goto exit_on_error;
00161         }
00162         variable->rangeminabs = xml_node_get_float_with_default
00163         (node, (const xmlChar *) LABEL_ABSOLUTE_MINIMUM, -G_MAXDOUBLE,
00164         &error_code);
00165         if (error_code)
00166         {
00167             variable_error (variable, gettext ("bad absolute minimum"));
00168             goto exit_on_error;
00169         }
00170     }
00171 }

```

```

00168     }
00169     if (variable->rangemin < variable->rangeminabs)
00170     {
00171         variable_error (variable, gettext ("minimum range not allowed"));
00172         goto exit_on_error;
00173     }
00174 }
00175 else
00176 {
00177     variable_error (variable, gettext ("no minimum range"));
00178     goto exit_on_error;
00179 }
00180 if (xmlHasProp (node, (const xmlChar *) LABEL_MAXIMUM))
00181 {
00182     variable->rangemax
00183     = xml_node_get_float (node, (const xmlChar *)
00184 LABEL_MAXIMUM,
00185                             &error_code);
00186     if (error_code)
00187     {
00188         variable_error (variable, gettext ("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, gettext ("bad absolute maximum"));
00197         goto exit_on_error;
00198     }
00199     if (variable->rangemax > variable->rangemaxabs)
00200     {
00201         variable_error (variable, gettext ("maximum range not allowed"));
00202         goto exit_on_error;
00203     }
00204     if (variable->rangemax < variable->rangemin)
00205     {
00206         variable_error (variable, gettext ("bad range"));
00207         goto exit_on_error;
00208     }
00209 }
00210 else
00211 {
00212     variable_error (variable, gettext ("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, gettext ("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, gettext ("bad sweeps"));
00235             goto exit_on_error;
00236         }
00237     }
00238 }
00239 else
00240 {
00241     variable_error (variable, gettext ("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)
00249 {
00250     // Obtaining bits representing each variable
00251     if (xmlHasProp (node, (const xmlChar *) LABEL_NBITS))
00252     {
00253         variable->nbits
00254         = xml_node_get_uint (node, (const xmlChar *)
00255 LABEL_NBITS,

```

```

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

```

Here is the call graph for this function:

5.28 variable.h

```

00001 /*
00002 MPCOTool:
00003 The Multi-Purposes Calibration and Optimization Tool. A software to perform
00004 calibrations or optimizations of empirical parameters.
00005
00006 AUTHORS: Javier Burguete and Borja Latorre.
00007
00008 Copyright 2012-2016, AUTHORS.
00009
00010 Redistribution and use in source and binary forms, with or without modification,
00011 are permitted provided that the following conditions are met:
00012
00013     1. Redistributions of source code must retain the above copyright notice,
00014        this list of conditions and the following disclaimer.
00015
00016     2. Redistributions in binary form must reproduce the above copyright notice,
00017        this list of conditions and the following disclaimer in the
00018        documentation and/or other materials provided with the distribution.
00019
00020 THIS SOFTWARE IS PROVIDED BY AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED
00021 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
00022 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
00023 SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00025 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
00026 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
00027 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
00028 IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
00029 OF SUCH DAMAGE.
00030 */
00031
00032 #ifndef VARIABLE__H
00033 #define VARIABLE__H 1
00034
00035 enum Algorithm
00036 {
00037     ALGORITHM_MONTE_CARLO = 0,
00038     ALGORITHM_SWEEP = 1,
00039     ALGORITHM_GENETIC = 2

```

```
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_GENETIC
 variable.h, 239

ALGORITHM_MONTE_CARLO
 variable.h, 239

ALGORITHM_SWEEP
 variable.h, 239

Algorithm
 variable.h, 239

config.h, 29

cores_number
 utils.c, 199
 utils.h, 215

DIRECTION_METHOD_COORDINATES
 input.h, 71

DIRECTION_METHOD_RANDOM
 input.h, 71

DirectionMethod
 input.h, 71

ERROR_NORM_EUCLIDIAN
 input.h, 72

ERROR_NORM_MAXIMUM
 input.h, 72

ERROR_NORM_TAXICAB
 input.h, 72

ERROR_NORM_P
 input.h, 72

ErrorNorm
 input.h, 71

Experiment, 15

experiment.c, 30
 experiment_error, 31
 experiment_free, 32
 experiment_new, 32
 experiment_open_json, 33
 experiment_open_xml, 34
 template, 36

experiment.h, 40
 experiment_error, 40
 experiment_free, 41
 experiment_new, 41
 experiment_open_json, 42
 experiment_open_xml, 43

experiment_error
 experiment.c, 31
 experiment.h, 40

experiment_free
 experiment.c, 32
 experiment.h, 41

experiment_new
 experiment.c, 32
 experiment.h, 41

experiment_open_json
 experiment.c, 33
 experiment.h, 42

experiment_open_xml
 experiment.c, 34
 experiment.h, 43

format
 variable.c, 233

gtk_array_get_active
 interface.h, 133
 utils.c, 199
 utils.h, 215

Input, 15

input.c, 46
 input_error, 47
 input_open, 47
 input_open_json, 48
 input_open_xml, 53

input.h, 70
 DIRECTION_METHOD_COORDINATES, 71
 DIRECTION_METHOD_RANDOM, 71
 DirectionMethod, 71
 ERROR_NORM_EUCLIDIAN, 72
 ERROR_NORM_MAXIMUM, 72
 ERROR_NORM_TAXICAB, 72
 ERROR_NORM_P, 72
 ErrorNorm, 71
 input_error, 72
 input_open, 72
 input_open_json, 73
 input_open_xml, 78

input_error
 input.c, 47
 input.h, 72

input_open
 input.c, 47
 input.h, 72

input_open_json
 input.c, 48
 input.h, 73

input_open_xml
 input.c, 53
 input.h, 78

- input_save
 - interface.c, 87
 - interface.h, 134
- input_save_direction_json
 - interface.c, 88
- input_save_direction_xml
 - interface.c, 89
- input_save_json
 - interface.c, 89
- input_save_xml
 - interface.c, 92
- interface.c, 85
 - input_save, 87
 - input_save_direction_json, 88
 - input_save_direction_xml, 89
 - input_save_json, 89
 - input_save_xml, 92
 - window_get_algorithm, 94
 - window_get_direction, 94
 - window_get_norm, 95
 - window_read, 95
 - window_save, 97
 - window_template_experiment, 99
- interface.h, 131
 - gtk_array_get_active, 133
 - input_save, 134
 - window_get_algorithm, 135
 - window_get_direction, 135
 - window_get_norm, 135
 - window_read, 136
 - window_save, 138
 - window_template_experiment, 140
- json_object_get_float
 - utils.c, 200
 - utils.h, 216
- json_object_get_float_with_default
 - utils.c, 200
 - utils.h, 216
- json_object_get_int
 - utils.c, 201
 - utils.h, 217
- json_object_get_uint
 - utils.c, 201
 - utils.h, 217
- json_object_get_uint_with_default
 - utils.c, 202
 - utils.h, 218
- json_object_set_float
 - utils.c, 202
 - utils.h, 218
- json_object_set_int
 - utils.c, 203
 - utils.h, 219
- json_object_set_uint
 - utils.c, 203
 - utils.h, 219
- main.c, 143
- Optimize, 17
 - thread_direction, 19
- optimize.c, 147
 - optimize_best, 149
 - optimize_best_direction, 150
 - optimize_direction_sequential, 150
 - optimize_direction_thread, 152
 - optimize_estimate_direction_coordinates, 153
 - optimize_estimate_direction_random, 153
 - optimize_genetic_objective, 154
 - optimize_input, 154
 - optimize_merge, 155
 - optimize_norm_euclidian, 156
 - optimize_norm_maximum, 157
 - optimize_norm_p, 157
 - optimize_norm_taxicab, 158
 - optimize_parse, 158
 - optimize_save_variables, 160
 - optimize_step_direction, 160
 - optimize_thread, 161
- optimize.h, 180
 - optimize_best, 182
 - optimize_best_direction, 183
 - optimize_direction_sequential, 183
 - optimize_direction_thread, 185
 - optimize_estimate_direction_coordinates, 186
 - optimize_estimate_direction_random, 186
 - optimize_genetic_objective, 187
 - optimize_input, 187
 - optimize_merge, 188
 - optimize_norm_euclidian, 189
 - optimize_norm_maximum, 190
 - optimize_norm_p, 190
 - optimize_norm_taxicab, 191
 - optimize_parse, 191
 - optimize_save_variables, 193
 - optimize_step_direction, 193
 - optimize_thread, 194
- optimize_best
 - optimize.c, 149
 - optimize.h, 182
- optimize_best_direction
 - optimize.c, 150
 - optimize.h, 183
- optimize_direction_sequential
 - optimize.c, 150
 - optimize.h, 183
- optimize_direction_thread
 - optimize.c, 152
 - optimize.h, 185
- optimize_estimate_direction_coordinates
 - optimize.c, 153
 - optimize.h, 186
- optimize_estimate_direction_random
 - optimize.c, 153
 - optimize.h, 186
- optimize_genetic_objective
 - optimize.c, 154

- optimize.h, 187
- optimize_input
 - optimize.c, 154
 - optimize.h, 187
- optimize_merge
 - optimize.c, 155
 - optimize.h, 188
- optimize_norm_euclidian
 - optimize.c, 156
 - optimize.h, 189
- optimize_norm_maximum
 - optimize.c, 157
 - optimize.h, 190
- optimize_norm_p
 - optimize.c, 157
 - optimize.h, 190
- optimize_norm_taxicab
 - optimize.c, 158
 - optimize.h, 191
- optimize_parse
 - optimize.c, 158
 - optimize.h, 191
- optimize_save_variables
 - optimize.c, 160
 - optimize.h, 193
- optimize_step_direction
 - optimize.c, 160
 - optimize.h, 193
- optimize_thread
 - optimize.c, 161
 - optimize.h, 194
- Options, 20
- ParallelData, 20
- precision
 - variable.c, 233
- Running, 21
- show_error
 - utils.c, 204
 - utils.h, 220
- show_message
 - utils.c, 204
 - utils.h, 220
- template
 - experiment.c, 36
- thread_direction
 - Optimize, 19
- utils.c, 197
 - cores_number, 199
 - gtk_array_get_active, 199
 - json_object_get_float, 200
 - json_object_get_float_with_default, 200
 - json_object_get_int, 201
 - json_object_get_uint, 201
 - json_object_get_uint_with_default, 202
 - json_object_set_float, 202
 - json_object_set_int, 203
 - json_object_set_uint, 203
 - show_error, 204
 - show_message, 204
 - xml_node_get_float, 204
 - xml_node_get_float_with_default, 206
 - xml_node_get_int, 206
 - xml_node_get_uint, 207
 - xml_node_get_uint_with_default, 208
 - xml_node_set_float, 208
 - xml_node_set_int, 209
 - xml_node_set_uint, 209
- utils.h, 213
 - cores_number, 215
 - gtk_array_get_active, 215
 - json_object_get_float, 216
 - json_object_get_float_with_default, 216
 - json_object_get_int, 217
 - json_object_get_uint, 217
 - json_object_get_uint_with_default, 218
 - json_object_set_float, 218
 - json_object_set_int, 219
 - json_object_set_uint, 219
 - show_error, 220
 - show_message, 220
 - xml_node_get_float, 220
 - xml_node_get_float_with_default, 222
 - xml_node_get_int, 222
 - xml_node_get_uint, 223
 - xml_node_get_uint_with_default, 224
 - xml_node_set_float, 224
 - xml_node_set_int, 225
 - xml_node_set_uint, 225
- Variable, 21
- variable.c, 227
 - format, 233
 - precision, 233
 - variable_error, 228
 - variable_free, 228
 - variable_new, 228
 - variable_open_json, 229
 - variable_open_xml, 231
- variable.h, 238
 - ALGORITHM_GENETIC, 239
 - ALGORITHM_MONTE_CARLO, 239
 - ALGORITHM_SWEEP, 239
 - Algorithm, 239
 - variable_error, 239
 - variable_free, 240
 - variable_new, 240
 - variable_open_json, 241
 - variable_open_xml, 243
- variable_error
 - variable.c, 228
 - variable.h, 239
- variable_free
 - variable.c, 228

- variable.h, [240](#)
- variable_new
 - variable.c, [228](#)
 - variable.h, [240](#)
- variable_open_json
 - variable.c, [229](#)
 - variable.h, [241](#)
- variable_open_xml
 - variable.c, [231](#)
 - variable.h, [243](#)
- Window, [22](#)
- window_get_algorithm
 - interface.c, [94](#)
 - interface.h, [135](#)
- window_get_direction
 - interface.c, [94](#)
 - interface.h, [135](#)
- window_get_norm
 - interface.c, [95](#)
 - interface.h, [135](#)
- window_read
 - interface.c, [95](#)
 - interface.h, [136](#)
- window_save
 - interface.c, [97](#)
 - interface.h, [138](#)
- window_template_experiment
 - interface.c, [99](#)
 - interface.h, [140](#)
- xml_node_get_float
 - utils.c, [204](#)
 - utils.h, [220](#)
- xml_node_get_float_with_default
 - utils.c, [206](#)
 - utils.h, [222](#)
- xml_node_get_int
 - utils.c, [206](#)
 - utils.h, [222](#)
- xml_node_get_uint
 - utils.c, [207](#)
 - utils.h, [223](#)
- xml_node_get_uint_with_default
 - utils.c, [208](#)
 - utils.h, [224](#)
- xml_node_set_float
 - utils.c, [208](#)
 - utils.h, [224](#)
- xml_node_set_int
 - utils.c, [209](#)
 - utils.h, [225](#)
- xml_node_set_uint
 - utils.c, [209](#)
 - utils.h, [225](#)