LSD Input GUI documentation

Laboratoire de simulation du depistage

February 22, 2011

Contents

Chapter 1

Introduction

LSD's input GUI has been developed to create XML configuration files for SCHNAPS, LSD's simulator. The GUI displays the XML file in a way that allows non-programmers to easily interprete the data. It also contains multiple tools and buttons to modify the file. It is written in python, and uses the python bindings of the Qt library to display the graphics.

This document aims at presenting the functionalities of the GUI. It does not contain a deep explanation of the code underlying SCHNAPS nor the GUI itself.

Chapter 2

SCHNAPS : A short introduction

SCHNAPS is a generic simulator designed for health care modelling and simulations, parametrizable by configuration files and usable by non-programmers such as public health specialists. SCHNAPS stands for *SynCHroNous Agent and Population-based Simulator*. Before starting to use the GUI, a small knowledge of how SCHNAPS behaves when running simulations is required. Mainly, two different steps occur during simulations.

2.1 SCHNAPS : Generator

At first, SCHNAPS generates virtual individuals. Each individual has its own variables. When supplying a configuration file to SCHNAPS, one has to ensure that these variables are correctly named, initialized, and that the information on how the variables are distributed in the virtual population is correct. Such subjects will be later discussed.

Individuals are generated at the start of a simulation. Also, the user can specify additional generations during the simulation.

2.2 SCHNAPS : Simulator

Once individuals have been generated, the simulator make sure every individual evolves during the simulation. Such evolution is done by "routing" the individuals in trees that have been previously created by the user. Probabilities, either variables or fixed in the tree, affect the path taken by the virtual individuals. A path is often configured to modify variables, hence causing the so-called evolution.

Chapter 3

LSD input GUI

The GUI itself is quite simple, consisting of a menu and multiple tabs. Basic tabs are always shown in the GUI while advanced tabs can be hidden/shown by the user.

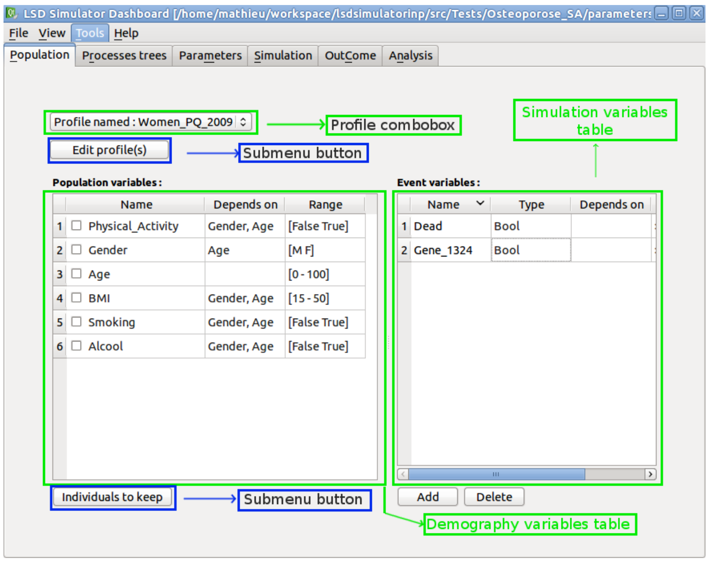
3.1 Population Tab

The population tab is where you define the characteristics of the individuals that are going to be created during a simulation. In a SCHNAPS perspective, the virtual population is defined by demography variables and simulation variables, respectively named population variables and event variables in the GUI. Demography variables are variables that are predefined in files. Simulation variables are variables that are specific to a problematic/simulation. To make it simple, simulation variables live in a configuration file, while demography variables live beneath the GUI subsystem and are accessible in all configurations. The population tab is considered as a basic tab.

Moreover, it is possible for the user to specify multiple profiles. Each profile has its demography variables and its simulation variables.

3.1.1 Main Tab

Figure 3.1 shows the different elements composing the population tab. One combobox, two tables and two submenu buttons are the principal elements of the tab. The *Add* and *Delete* buttons can be seen as part of the event variables table.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose_SA/parameters
File View Tools Help
Population
Processes trees
Parameters
Simulation
OutCome
Analysis
Profile named : Women_PQ_2009
Profile combobox
Edit profile(s)
Submenu button
Simulation variables table
Individuals to keep
Submenu button
Add
Delete
Demography variables table

Population variables :

	Name	Depends on	Range
1	☐ Physical_Activity	Gender, Age	[False True]
2	☐ Gender	Age	[M F]
3	☐ Age		[0 - 100]
4	☐ BMI	Gender, Age	[15 - 50]
5	☐ Smoking	Gender, Age	[False True]
6	☐ Alcool	Gender, Age	[False True]

Event variables :

	Name	Type	Depends on
1	Dead	Bool	
2	Gene_1324	Bool	

Figure 3.1: Population tab

3.1.1.1 Profile ComboBox

Choosing a profile refreshes the two tables with the population and event variables data associated with the selected profile. nstructions on how to create a profile will be later discussed.

3.1.1.2 Population variables table

The population variables table lists information about the data found in the demography that is linked with the selected profile. Each line is a variable found in the file. As for the columns:

1. Name : Name of the variable, as found in the demography file
2. Depends on : Variable dependencies
3. Range : Variable possible values

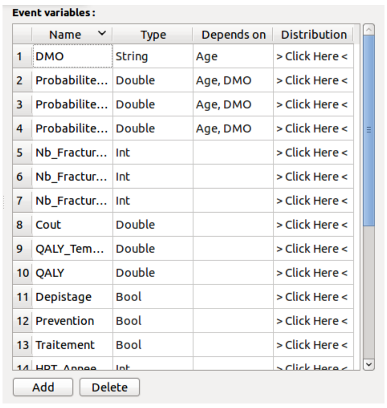
Note the checkbox located left of the name of the variables. The table lists all variables found in the demography. However, users might want to discard variables when generating the virtual population. Thus, only checked variables are kept.[1]

3.1.1.3 Simulation variables table

The event variables table lists information that the user has previously entered in the table. Figure 3.2 shows the table and its buttons. Each line is a variable found in the configuration file for the currently selected profile. As for the columns:

[1]When checking variables that have dependencies, ensure the dependencies are also checked. Otherwise, it will result in a simulator crash at runtime.

1. Name : Name of the variable, as found in the configuration file
2. Type : Variable type
3. Depends on : Variable dependencies
4. Distribution : Allows the user to edit variable's tree.



Event variables:

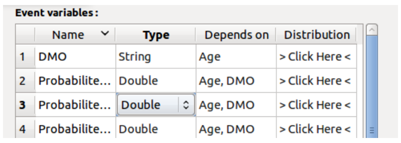
	Name	Type	Depends on	Distribution
1	DMO	String	Age	> Click Here <
2	Probabilite...	Double	Age, DMO	> Click Here <
3	Probabilite...	Double	Age, DMO	> Click Here <
4	Probabilite...	Double	Age, DMO	> Click Here <
5	Nb_Fractur...	Int		> Click Here <
6	Nb_Fractur...	Int		> Click Here <
7	Nb_Fractur...	Int		> Click Here <
8	Cout	Double		> Click Here <
9	QALY_Tem...	Double		> Click Here <
10	QALY	Double		> Click Here <
11	Depistage	Bool		> Click Here <
12	Prevention	Bool		> Click Here <
13	Traitement	Bool		> Click Here <
14	HPT_Annee	Int		> Click Here <

Add Delete

Figure 3.2: Event variables table

Unlike the population variables table, all columns of the event variables table can be edited, except for the *Depends on* column. Double-click on the *Name* column to edit variable's name.[2]

Variable type is a more obscure concept that can be hard to understand by non-programers. Unfortunately, typing is essential for SCHNAPS. Double-click on the *Type* column to edit variable's type.[3] As seen in Figure 3.3, a combobox will appear, allowing the user to choose between predefined types. More on types can be seen in Chapter 4.



Event variables:

	Name	Type	Depends on	Distribution
1	DMO	String	Age	> Click Here <
2	Probabilite...	Double	Age, DMO	> Click Here <
3	Probabilite...	Double	Age, DMO	> Click Here <
4	Probabilite...	Double	Age, DMO	> Click Here <

Figure 3.3: Type's edit combobox

Double-clicking on the *Distribution* column opens the tree editor. Here, the tree editor is used to set variable values, either in a fixed way or by statistical means. See Chapter 4 for more details on the tree editor.

The *Add* and *Delete* buttons are used to add and delete variables from the event variables table. More on adding and deleting can be found in Section 3.9.1.

[2]If a variable name is edited and the edited variable is a dependency of another variable, ensure the modifications are also made in the dependent variable's tree. Otherwise, it will result in a simulator crash at runtime.

[3]Again, modifying a variable's type might cause problems if this variable is referenced somewhere in the trees. Be careful before doing so.

3.1.2 Submenus

Two submenus can be opened pressing the two submenu buttons seen in Figure 3.1.

3.1.2.1 Profile manager submenu

The *Edit profile(s)* button opens the Profile Manager, as seen in Figure 3.4. The *Available profiles* list shows the currently defined profiles of the project.

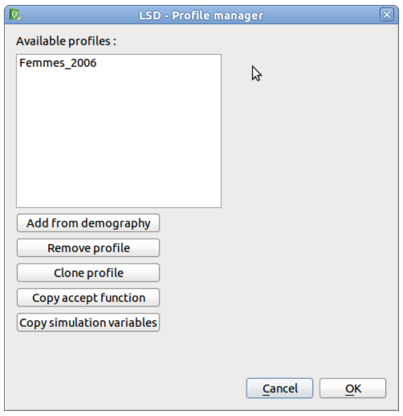
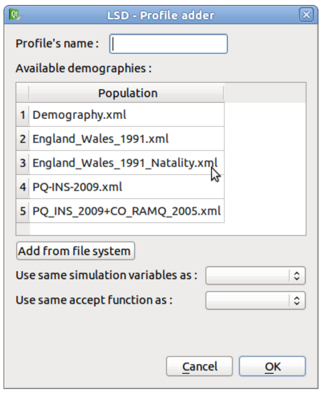


Figure 3.4: Profile manager

Adding a profile Adding a profile can be achieved by pressing the *Add from demography* button. When pressed, the Profile Adder is opened(see Figure 3.5).



	Population
1	Demography.xml
2	England_Wales_1991.xml
3	England_Wales_1991_Natality.xml
4	PQ-INS-2009.xml
5	PQ_INS_2009+CO_RAMQ_2005.xml

Figure 3.5: Profile adder

The line edit labeled *Profile's name* is where you enter the name of your new profile. The *Available demographies* list displays the demography files found in the Demography folder of the GUI application. For custom demography files to be listed as well, use the *Add from file system* button to specify where other demographies rest in the file system. Once the demography has been chosen, you can add your new profile by pressing the *OK* button. This will add a new profile, with empty event variables and an empty accept function. If you want one or both of the last two items to be copied from an existing profile, specify it using the appropriate combobox. More on the accept function can be found in section 3.1.2.2.

Removing a profile Pressing the *Remove profile* button while a profile is selected in the *Available profiles* list will delete the profile.

Cloning a profile To create a profile that is identical to a selected profile in the *Available profiles* list(except for the name), use the *Clone profile* button. As seen in Figure 3.6, a combobox will appear, asking the user for the clone's name.

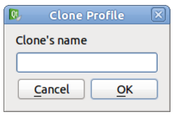


Figure 3.6: Clone profile

Copy accept function It is possible to copy the accept function of the currently selected profile. Selecting the profile and pressing the *Copy accept function* button will lead you to the Copy Wizard(see Figure 3.7). Just select the profiles you want the accept function copied to by checking the corresponding checkboxes.

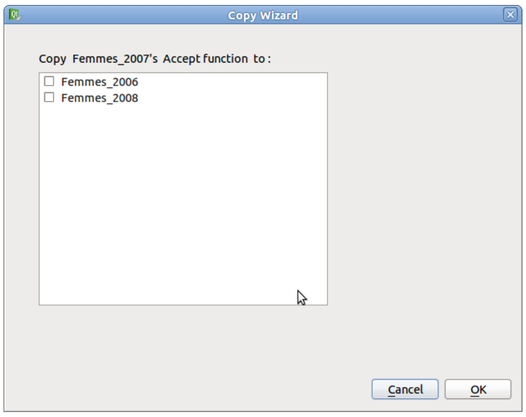


Figure 3.7: Copy Wizard

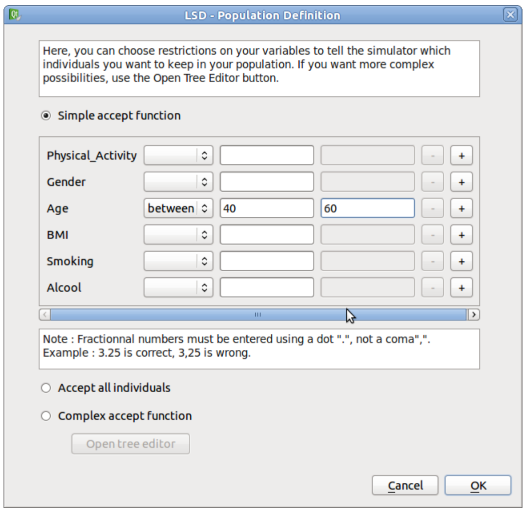
Copy simulation variables It is possible to copy the accept simulation variables of the currently selected profile. Just use the same method as described above.

3.1.2.2 Accept function submenu

The accept function submenu is used to add restrictions on how the individuals are going to be generated. The simplest choice between the three radio buttons is the *Accept all individuals* choice. This choice tells SCHNAPS to generate a population that is going to fit the distribution of the variables found in the demography file of the currently selected profile.

However, some simulations target specific groups of the population. For example, a simulation can target individuals that are in a 40 to 60 years old range. This is where the *Simple accept function* choice can be useful. Figure 3.8 shows the Accept Function Manager. The above example is shown in the simple accept function area. User can choose between equals, greater or equal(>=), smaller or equal(<=) and between in the condition combobox. The second value line edit is disabled when the three first conditions are used(equals, >=, <=). The plus and minus signs are used to add or remove conditions on the same variable.[4]

To specify even more complex accept functions, you can use tree editor by selecting the *Complex accept functions* radio button and pressing the *Open tree editor* button.



Here, you can choose restrictions on your variables to tell the simulator which individuals you want to keep in your population. If you want more complex possibilities, use the Open Tree Editor button.

[✓ Simple accept function]

Physical_Activity			
Gender			
Age	between	40	60
BMI			
Smoking			
Alcool			

Note : Fractionnal numbers must be entered using a dot ".", not a coma",".
Example : 3.25 is correct, 3,25 is wrong.

Accept all individuals

Complex accept function

Figure 3.8: Accept function manager

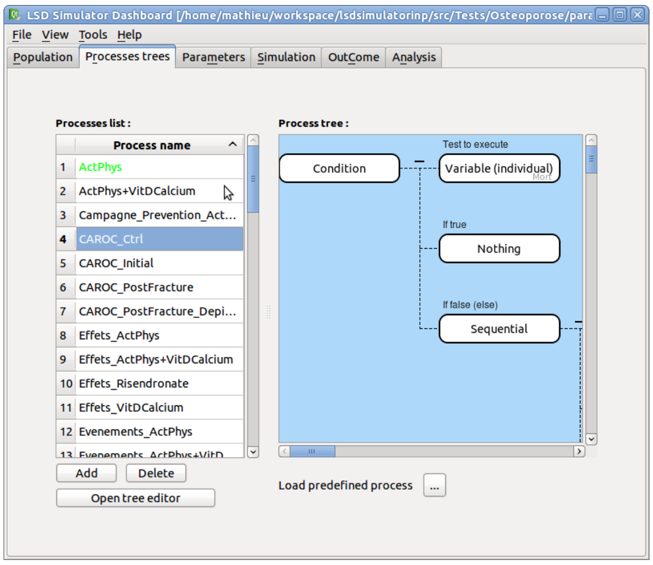
3.2 Processes Tab

The processes tab is where you create all the subtrees that the are going to be used in a simulation. In medical terms, a model is often seen as one big tree with multiple branches. However, building such a tree using the tree editor would be complex, error-proned and quite difficult to understand. To make it simpler, processes can be seen as subtrees. It is the user's job to separate the theoretical model's tree in multiple logical subtrees. It takes a little bit of practice, but it is not so hard to achieve.

3.2.1 Main Tab

Figure 3.9 shows the different elements composing the processes tab.

[4]Conditions between variables can be seen as Logical AND. Conditions on the same variable can be seen as Logical OR


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/para
File View Tools Help
Population Processes trees Parameters Simulation OutCome Analysis
Processes list :
Process tree :
Test to execute
Condition
Variable (individual)
If true
Nothing
If false (else)
Sequential
Add Delete
Open tree editor
Load predefined process ...

	Process name
1	ActPhys
2	ActPhys+VitDCalcium
3	Campagne_Prevention_Act...
4	CAROC_Ctrl
5	CAROC_Initial
6	CAROC_PostFracture
7	CAROC_PostFracture_Depi...
8	Effets_ActPhys
9	Effets_ActPhys+VitDCalcium
10	Effets_Risendronate
11	Effets_VitDCalcium
12	Evenements_ActPhys
13	Evenements_ActPhys+VitD

Figure 3.9: Processes tab

3.2.1.1 Processes List

Processes of the currently loaded project are listed in the *Processes list*. To modify a process name, double-click on the line. A line edit will appear, hence allowing the editing.[5] Use the *Add* and *Delete* buttons at the bottom of the list to add or remove processes. Use the Open Tree Editor button to edit the currently selected profile. More on the tree editor can be found in chapter 4

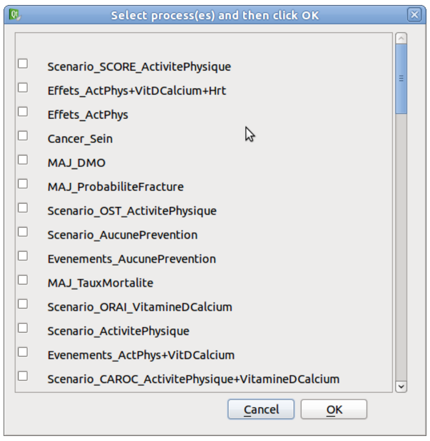
3.2.1.2 Process Tree

A preview of the currently selected process is shown in this area. The preview is in fact a picture that was taken using the tree editor. You can navigate through the preview using the scroll bars. However, editing is not possible.

3.2.2 Submenu

Apart from the tree editor, only one submenu can be accessed using the Process tab. This menu can be opened by pressing the *Load predefined process* button. This submenu is intended for importing processes found in other project. Two projects simulating different problematics might have some of their processes similar enough to have one or more of them imported from one project to the other. When pressed, the user is asked to select a project where the processes to import can be found. The GUI then lists the processes of the selected project. Choice(s) can ben done checking the appropriate checkbox(es).

[5]Ensure the modified process name is not referenced in other processes, or else this might lead to a simulator crash at runtime.



Scenario_SCORE_ActivitePhysique
Effets_ActPhys+VitDCalcium+Hrt
Effets_ActPhys
Cancer_Sein
MAJ_DMO
MAJ_ProbabiliteFracture
Scenario_OST_ActivitePhysique
Scenario_AucunePrevention
Evenements_AucunePrevention
MAJ_TauxMortalite
Scenario_ORAI_VitamineDCalcium
Scenario_ActivitePhysique
Evenements_ActPhys+VitDCalcium
Scenario_CAROC_ActivitePhysique+VitamineDCalcium

Figure 3.10: Import submenu

3.3 Parameters Tab

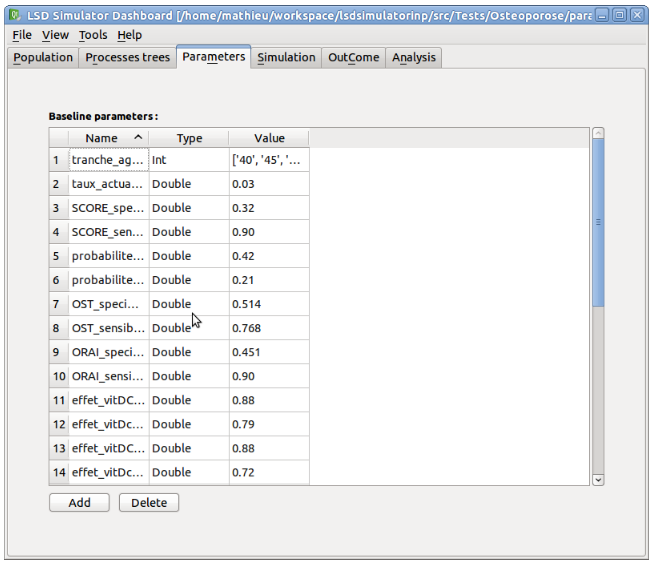
When building trees using the tree editor, many nodes can be used to steer virtual individuals between different branches or to modify their variables. These nodes need information on how to interact with the individuals. A simple but common example would be a *Treatment* node, where the price of the treatment has to be provided. Now, let's say this treatment can be found at 20 different locations in the model, all in different subtrees. And now, let's say you realize the price you entered initially was wrong. A lot of job ahead if you hardcoded the price.

This is one of the main reason why the parameters tab exists. Most of the nodes used in the processes offer two ways of entering the data ; by hardcoding it or by referring to it via a parameter. Using parameters is a good approach if :

1. The same data is used at multiple places in the processes
2. The data is sensitive
3. The data is going to be part of a sensitivity analysis

3.3.1 Main Tab

As seen in Figure 3.11, the main tab is quite simple. It consists of the *Baseline parameters* table, which lists all the parameters of the currently loaded project. The *Add* and *Delete* buttons are used to add or remove parameters.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/para

File View Tools Help
Population Processes trees Parameters Simulation OutCome Analysis

Baseline parameters :

	Name	Type	Value
1	tranche_ag...	Int	['40', '45', '...
2	taux_actua...	Double	0.03
3	SCORE_spe...	Double	0.32
4	SCORE_sen...	Double	0.90
5	probabilite...	Double	0.42
6	probabilite...	Double	0.21
7	OST_speci...	Double	0.514
8	OST_sensib...	Double	0.768
9	ORAI_speci...	Double	0.451
10	ORAI_sensi...	Double	0.90
11	effet_vitDC...	Double	0.88
12	effet_vitDc...	Double	0.79
13	effet_vitDC...	Double	0.88
14	effet_vitDc...	Double	0.72

Figure 3.11: Parameters tab

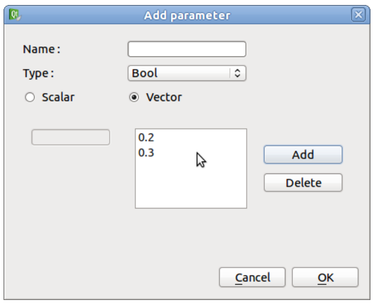
Each line of the table describes a parameter. As for the columns:

1. Name : Name of the parameter, as found in the configuration file
2. Type : Parameter type
3. Depends on : Parameter value

You can modify the listed parameters using the table. Double-click on the parameter's name to enable a line edit, allowing the edition of the parameter's name.[6] Like variables, user has to specify parameter's type(See section 3.1.1.3). Modifying the type can be achieved by double-clicking on the type of the currently selected parameter. Finally, to modify the value, double-click on the value of the currently selected parameter. You can always delete parameters by using the *Delete* button.

3.3.2 Submenu

Unlike most of the lists and tables found in the GUI, pressing the *Add* button leads to a submenu, as seen in Figure 3.12. This submenu is used to create a new parameter.


Add parameter

Name:

Type: Bool

○ Scalar [✓ ● Vector]

0.2
0.3

[6]As always, changing a parameter's name when it is referenced in a process will lead to problems at runtime.

Figure 3.12: Parameters tab

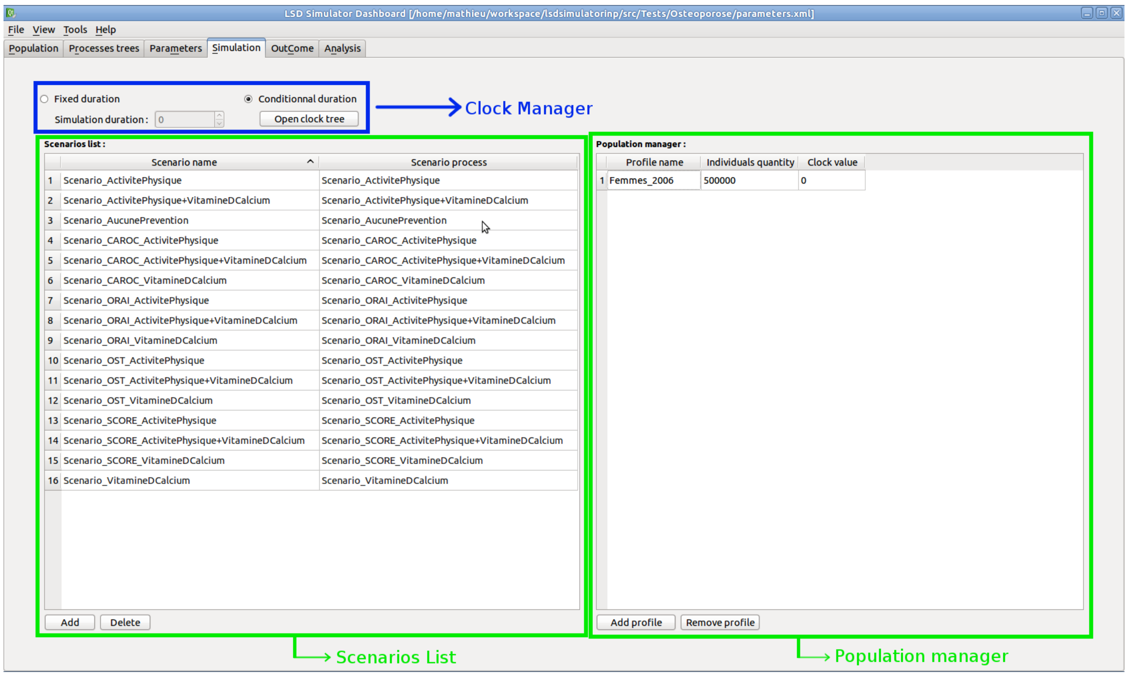
First, name your parameter using the line edit. Then, select parameter's type using the combobox. Finally select if the parameter is a scalar(single value) or a vector, and enter the value in the line edit for a scalar parameter or using the list and the *Add/Delete* buttons if it is a vector. Vector parameters are used with branching nodes. See chapter 4 for more details.

3.4 Simulation Tab

The simulation tab contains the metadata of the project.

3.4.1 Main tab

The simulation tab, as seen in Figure 3.13, is principally made of two tables named *Scenarios list* and *Population manager*. Some radio buttons, line edit and push button are also part of the tab, and this group of widgets is called the Clock Manager.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/parameters.xml]
File View Tools Help
Population Processes trees Parameters Simulation OutCome Analysis
Open clock tree
Add
Delete
Add profile
Remove profile
Clock Manager
Scenarios List
Population manager

Fixed duration [✓ Conditionnal duration]

Simulation duration : 0

Scenarios list :

	Scenario name	Scenario process
1	Scenario_ActivitePhysique	Scenario_ActivitePhysique
2	Scenario_ActivitePhysique+VitamineDCalcium	Scenario_ActivitePhysique+VitamineDCalcium
3	Scenario_AucunePrevention	Scenario_AucunePrevention
4	Scenario_CAROC_ActivitePhysique	Scenario_CAROC_ActivitePhysique
5	Scenario_CAROC_ActivitePhysique+VitamineDCalcium	Scenario_CAROC_ActivitePhysique+VitamineDCalcium
6	Scenario_CAROC_VitamineDCalcium	Scenario_CAROC_VitamineDCalcium
7	Scenario_ORAI_ActivitePhysique	Scenario_ORAI_ActivitePhysique
8	Scenario_ORAI_ActivitePhysique+VitamineDCalcium	Scenario_ORAI_ActivitePhysique+VitamineDCalcium
9	Scenario_ORAI_VitamineDCalcium	Scenario_ORAI_VitamineDCalcium
10	Scenario_OST_ActivitePhysique	Scenario_OST_ActivitePhysique
11	Scenario_OST_ActivitePhysique+VitamineDCalcium	Scenario_OST_ActivitePhysique+VitamineDCalcium
12	Scenario_OST_VitamineDCalcium	Scenario_OST_VitamineDCalcium
13	Scenario_SCORE_ActivitePhysique	Scenario_SCORE_ActivitePhysique
14	Scenario_SCORE_ActivitePhysique+VitamineDCalcium	Scenario_SCORE_ActivitePhysique+VitamineDCalcium
15	Scenario_SCORE_VitamineDCalcium	Scenario_SCORE_VitamineDCalcium
16	Scenario_VitamineDCalcium	Scenario_VitamineDCalcium

Population manager :

	Profile name	Individuals quantity	Clock value
1	Femmes_2006	500000	0

Figure 3.13: Simulation tab

3.4.1.1 Scenarios list

A scenario is the first process SCHNAPS applies on all individuals when the simulation begins.[7]. Each line in the *Scenarios list* is a scenario. As for the columns :

[7]When individuals are introduced in the simulator during the simulation, the process is applied like if the simulation had just started for them.

1. Scenario name : Name of the scenario
2. Scenario process : Name of the associated process

The scenario's associated process has to be created in the process tab(See section 3.2). Double-clicking on the cell will trigger a combobox, listing all processes found in the process tab. As for the name, it can be confusing. One good reason to make the scenario's name differ from the process name is that in complex projects with multiple scenarios, the name of the processes can become quite long. Naming the scenario with simple names(like numbers) can become handy. And since you only refer to scenarios when launching your simulation, you're not going to spend your time going through a lookup table searching for scenario 35's description, something that might happen if you don't use good naming conventions with your processes.

If this explanation doesn't satisfy your need for good reasons to justify the distinction between scenario's process and name, here's a better one. The GUI has two "advanced" tabs. One of them is the Environment tab(See section 3.7.1). The environment can be seen as a meta-individual, holding variables that are common to all virtual individuals. Like all other individuals , the Environment can go through a process that modifies its variables. So, a scenario can be a combination of two processes : one process that is applied to all the virtual individuals at the beginning of the simulation and one process that is applied to the environment, also at the beginning. In this case, naming a scenario takes all its sense. See section 3.8.4.2 for details on how to enable this feature.

Using the *Add* and *Delete* buttons, you can add or remove scenarios to/from the project. Double-click on cells to modify their value. Scenario's name can be modified via a line edit while all processes found in the process tab(See section 3.2) are listed in a combobox.

3.4.1.2 Population Manager list

This is where you tell the simulator how to generate the individuals during the simulation. The columns of this table are :

1. Profile name : Name of the profile chosen for the generation
2. Individual quantity : Number of individuals to generate
3. Clock Value : Simulation step when the individuals are going to be created

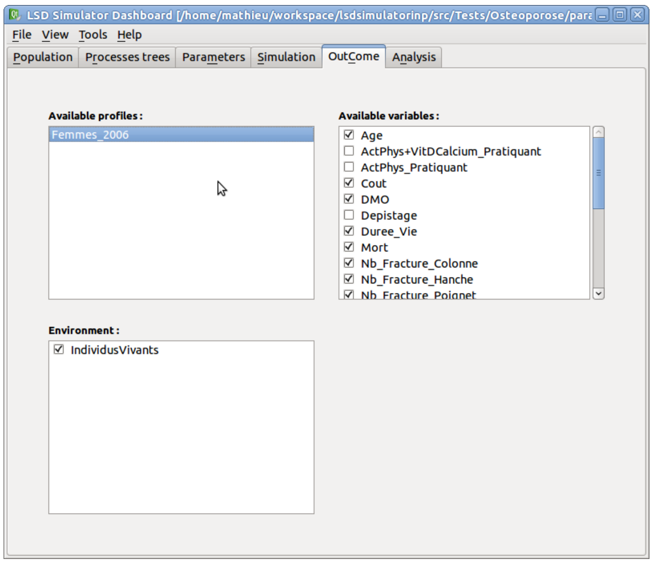
Let x be a value in Column 1, y be a value in Column 2 and z be a value in Column 3. So, for all lines, the simulator's generator creates x individuals using profile y and introduce them in the simulator at time z. Use *Add* and *Delete* buttons to manage the generations. Modify the different cells by double-clicking on them. Quantity and time can be modified via a line edit while available profiles are listed in a combobox.

3.4.1.3 Clock manager

There are two ways for the simulator to select the amount of steps the simulation is going to last. First one is to define a fixed number of steps using the *Fixed duration* radio button. More complex possibilities can be set using the *Complex duration* radio button, and pressing the *Open clock tree* button. A complex condition ends the simulation when clock's tree returns True(see chapter 4).

3.5 OutCome Tab

In most cases, half of individual variables found in a simulation are control or state variables. Usually the user care more or less about such variables final values. The outcome tab is there to tell the simulator which variables to keep and write to a file and which one to throw away at the end of a simulation.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/parε
File View Tools Help
Population
Processes trees
Parameters
Simulation
OutCome
Analysis

Available profiles:

Femmes_2006

Available variables:

- ☑ Age
- ☐ ActPhys+VitDCalcium_Pratiquant
- ☐ ActPhys_Pratiquant
- ☑ Cout
- ☑ DMO
- ☐ Depistage
- ☑ Duree_Vie
- ☑ Mort
- ☑ Nb_Fracture_Colonne
- ☑ Nb_Fracture_Hanche
- ☑ Nb_Fracture_Poignet

Environment:

- ☑ IndividusVivants

Figure 3.14: Outcome tab

3.5.1 Profile List

The *Available profiles* list displays the profile found in the population tab. When clicking on a profile, it updates the *Available variables* list.

3.5.2 Variable List

The *Available variables* list displays the variables found in the currently selected profile. It first lists the population variables in alphabetical order. The event variables are then listed, also in alphabetical order. There's a checkbox next to each variable name, that can be checked or unchecked, depending if the user wants to keep the variable or not. Population variables that are not chosen in the population tab at first are going to be listed, but the user won't be able to interact with their checkbox.

3.5.3 Environment List

When the environment is used in a simulation, environment variables are also listed. Rad the section above for more details.

3.6 Analysis Tab

The analysis tab is still under construction and might be modified/renamed or even disappear.

3.7 Advanced Tabs

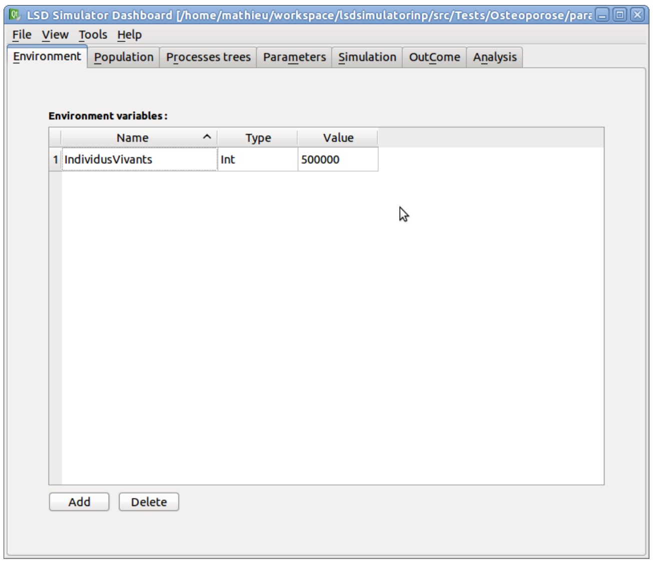
Advanced tabs are tabs that are not initially shown in the tab list. When enabled[8], they allow the user to define complex elements of the simulation. Unlike normal tabs, they do not need to be filled.

3.7.1 Environment Tab

The environment, is some kind of master individual holding variables that are common to all virtual individuals. The environment is often used to carry control variables. A simple but telling example would be simulations where all individuals have to be "dead" for the simulations to stop. The common way to proceed in such cases is to keep a variable in the environment that registers the total number of individuals at the beginning of the simulation. This number is decremented when a virtual individual "dies". Once it has reached zero, the simulation is over. This is done by defining a complex duration for the clock(see section 3.4.1.3).

The environment tab is quite simple. It consists of the *Environment variables* table and the *Add/Delete* buttons. Each line of the table is a variable. The three colums are respectively the variable's name, type and value. Each cell can be double-clicked so the value it contains can be modified[9]. Like all other tables of the GUI, lines can be added or removed using the *Add/Delete* buttons.

Finally, when shown, the environment tab is the first tab of the GUI.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/para
File View Tools Help
Environment Population Processes trees Parameters Simulation OutCome Analysis
Add
Delete

Environment variables:

	Name	Type	Value
1	IndividusVivants	Int	500000

Figure 3.15: Environment tab

[8]See section 3.8.2 for more details on how to enable advanced tabs.

[9]As always, make sure you know what you're doing before changing a variable's name or value

3.7.2 Observers Tab

The observers tab is probably the most difficult tab to understand. Even though, it can be very useful in most projects.

Take a simulation where each clock steps represents a normal year in a virtual individual's life. Each time the clock's value changes, some individual variables have to be updated(age, death probability,etc...). The user will usually do this by creating a process that modifies the variables. He then will list in the observers tab all the processes that have to be applied to individuals when the clock's value changes.

Even though the observers tab is considered as advanced, its layout remains simple. It consists of the *Clock observers* table and the *Add/Delete* buttons. Each line represents an observer. The first column is the name of the observer's process. As for the *Targets* column, it dictates to who the process is going to be applied :

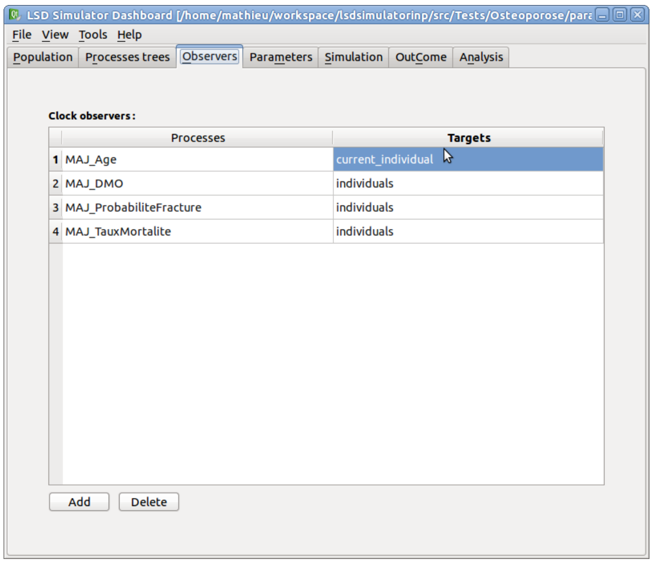
1. individuals : Apply the observing process to all virtual individuals
2. environment : Apply the observing process to the environment
3. current individual : Deprecated. Might be deleted in a near future.

The observing process and the targets can both be modified by double-clicking on the cell. Note that:

1. The observers are executed before any other processes at the beginning of a clock step.
2. The observers are executed sequentially in order they appear in the table.

The last statement is quite important. If observing processes results depend on variables that are modified in other observers, make sure the sorting of your observers is correct. For example, if a process updating the death probability of the individuals depends on Age(also modified in an other observer), make sure the observer modifying the age is placed before the observer updating the death probability in the table.

Finally, when shown, the observers tab is located between the processes and the parameters tab.


LSD Simulator Dashboard [/home/mathieu/workspace/lsdsimulatorinp/src/Tests/Osteoporose/para
File View Tools Help
Population Processes trees Observers Parameters Simulation OutCome Analysis
Add
Delete

Clock observers :

	Processes	Targets
1	MAJ_Age	current_individual
2	MAJ_DMO	individuals
3	MAJ_ProbabiliteFracture	individuals
4	MAJ_TauxMortalite	individuals

Figure 3.16: Observers tab

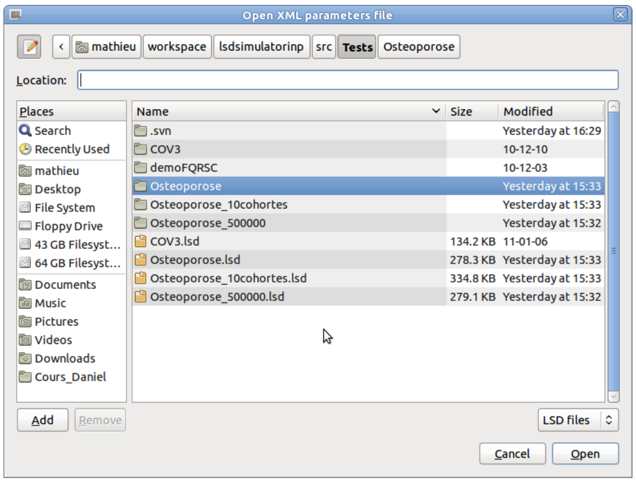
3.8 Menus

3.8.1 File

The file menu holds different command for managing the project files and leaving the application.

3.8.1.1 Open

The *Open* command pops open a file system navigator(see Figure 3.17) so you can select a project to open. LSD input GUI projects wear a **.lsd** file extension. Open command can be triggered using the *Ctrl-O* shortcut.



Name	Size	Modified
.svn		Yesterday at 16:29
COV3		10-12-10
demoFQRSC		10-12-03
Osteoporose		Yesterday at 15:33
Osteoporose_10cohortes		Yesterday at 15:33
Osteoporose_500000		Yesterday at 15:32
COV3.lsd	134.2 KB	11-01-06
Osteoporose.lsd	278.3 KB	Yesterday at 15:33
Osteoporose_10cohortes.lsd	334.8 KB	Yesterday at 15:33
Osteoporose_500000.lsd	279.1 KB	Yesterday at 15:32

Figure 3.17: File system navigator(Linux)

3.8.1.2 Start New Project

The command's name says it all. It starts a new empty project.

3.8.1.3 Save

Saves the current project. *Save* command can be triggered using the *Ctrl-S* shortcut.

3.8.1.4 Save As

Opens a file system navigator and asks the user to give a new name to the currently loaded project so it can be saved as a new project. *Save As* command can be triggered using the *Ctrl-Shift-S* shortcut.

3.8.1.5 Open plugin

The *Open Plugin* command is used to add functionalities to the GUI. These functionalities are, for the moment, only related to the tree editor. *Open Plugin* command can be triggered using the *Ctrl-Shift-O* shortcut.

Once activated, the command opens the Plugin Manager, as seen in Figure 3.18. The plugin manager lists the different plugins used by the currenty loaded project. It also lists currently unused plugins that are found in the GUI folders. If a plugin is located somewhere else on the file system, it can be added using the *Browse* button. Select plugins in one of the two lists and use the arrows to add or remove plugins from the currently selected project. More on plugins can be found in section 6.

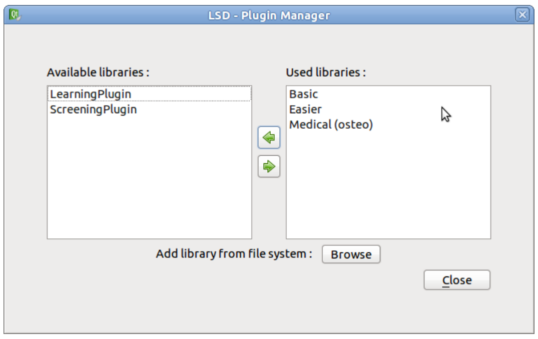


Figure 3.18: Plugin manager

3.8.1.6 Exit

Leaves the application. *Exit* command can be triggered using the *Ctrl-Q* shortcut.

3.8.2 View

The view menu allows the user to show/hide the advanced tabs(see section 3.6) via checkboxes.

3.8.3 Tools

The tools menu holds various extra functionalities of the GUI.

3.8.3.1 Take Screenshot

Takes a screenshot of what is currently visible in the GUI's main Window. *Take Screenshot* command can be triggered using the *Ctrl-T* shortcut. Once the screenshot has been taken, user is asked to save the picture somewhere on the file system. Actually, pictures can only be saved in a .png format. More types may be implemented in the future.

3.8.3.2 Check variables validity

For all simulation variables of all profiles, this command checks if there are any errors located in the tree. More on validation can be read in section 4.4.

3.8.3.3 Check processes validity

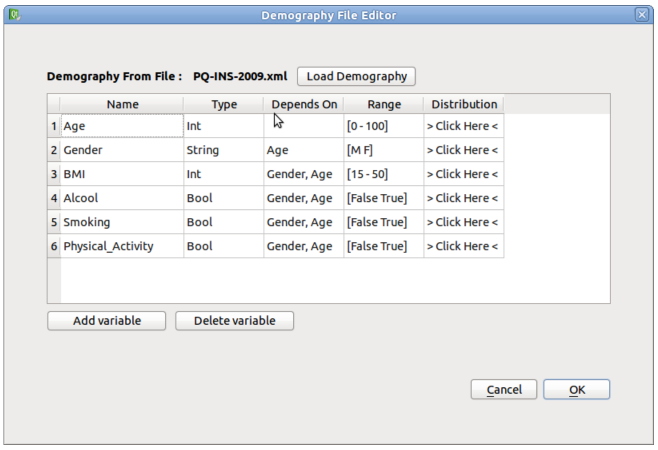
For all processes, this command checks if there are any errors located in the tree. More on validation can be read in section 4.4.

3.8.3.4 Check all

Runs the command of the two previous section sequentially.

3.8.3.5 Demography editor

Demography files can be added or edited via the demography editor. When pressed, the demography editor starts by asking the user if he wants to create a new demography or to load an existing one. The demography editor is then opened. Figure 3.19 shows the demography editor where an existing demography has been loaded.



Demography From File : **PQ-INS-2009.xml**

	Name	Type	Depends On	Range	Distribution
1	Age	Int		[0 - 100]	> Click Here <
2	Gender	String	Age	[M F]	> Click Here <
3	BMI	Int	Gender, Age	[15 - 50]	> Click Here <
4	Alcool	Bool	Gender, Age	[False True]	> Click Here <
5	Smoking	Bool	Gender, Age	[False True]	> Click Here <
6	Physical_Activity	Bool	Gender, Age	[False True]	> Click Here <

Figure 3.19: Demography editor

The demography editor shows the name fo the file currently being edited. Each line of the table corresponds to a variable found in the file. As for the columns:

1. Name : Name of the variable, as found in the demography file
2. Type : Variable type
3. Depends on : Variable dependencies
4. Range : Variable possible values
5. Distribution : Allows the user to edit variable's tree.

As always, double-click on a cell to edit value. Note that *Depends on* and *Range* cells are not editable. Use the *Add/Delete* buttons to add or remove variables to/from the demography.

You can also change the currently edited demography by pressing the *Load Demography* button. A file system navigator will then open, asking you to select a demography file to edit.

Finally, note that modifications brought to the demography won't be effective until you save them. Saving is offered when the user presses the *Ok* button or presses the *Load Demography* button. Pressing the *Cancel* or the Close button exits the demography editor without saving changes.

3.8.3.6 Configuration file generator

SCHNAPS uses pseudo-random number generation for its Generator and its Simulator. SCHNAPS Simulator and SCHNAPS Generator use pseudo-random generators, which can be configured via the configuration file generator. That makes a lot of generators :

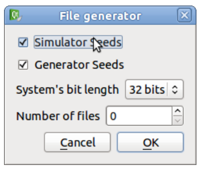
1. SCHNAPS Generator : Part of SCHNAPS in charge of creating the virtual individuals
2. Pseudo-random generator : Piece of software in charge of creating numbers in a pseudo-aleatory manner
3. Configuration File generator : See Figure 3.20

A pseudo-random generator uniformly generates numbers. However, it needs a number, called a seed, to start working properly. The reason why it is called a **pseudo-random** generator is that given the same seed when it is initialized, it will always generate the same exact sequence of numbers. Given this information, we can infer that, for two identical configuration files(except for the seeds):

	Same simulator seeds	Different simulator seeds
Same generator seeds	Same population Same events Same output	Same population Different events Different output
Different generator seeds	Different populations ? Different output	Different populations Different events Different output

Note the question mark. Given a model where the paths are never influenced by individual variables, we can argue that for this combination of seeds, the same events will happen for different individuals. Most likely, this will never be the case. We can almost always replace the question mark by a *Different events*.

So, since we are not interested in simulating the same exact thing forever, let's take a look at the different options. First, the user is asked to select which of the Simulation, Generation or both seeds he wants to generate. Depending on the system's hardware and/or operating system, seeds can take 32 bits or 64 bits values. The user has to be aware of this and choose the right bit's length(choosing 32 bits on a 64 bits system is not a big deal but the opposite may certainly lead to problems). Finally, user has to choose how many files he want to create. The files are going to be created inside the project's directory. Pressing the *Ok* button generates the files.



File generator

☑ Simulator Seeds

☑ Generator Seeds

System's bit length: 32 bits

Number of files: 0

Cancel | OK

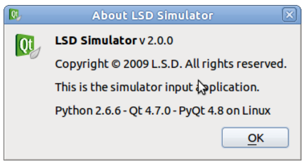
Figure 3.20: Configuration file generator

3.8.4 Help

For the moment, this section doesn't really contain any help or support doc. This is going to be done in a near future.

3.8.4.1 About

The *about* command displays information about the GUI. It also lists what version of Python, Qt and PyQt is installed on the computer, as well as the name of the operating system.



LSD Simulator v 2.0.0

Copyright © 2009 L.S.D. All rights reserved.

This is the simulator input application.

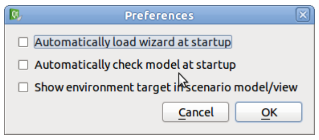
Python 2.6.6 - Qt 4.7.0 - PyQt 4.8 on Linux

Figure 3.21: About

3.8.4.2 Preferences

To display the preferences of the application, use this command. *Preferences windows* is going to be extended in the future. For now it contains three checkable/uncheckable options:

1. Automatically load wizard at startup : Loads wizard at startup.(See section 5)
2. Automatically check model at startup : Checks the project for errors at startup.(See section 3.8.3.4)
3. Show environment target in scenario model/view : Adds a column to the scenarios list.(See section 3.4.1.1)



☐ Automatically load wizard at startup
☐ Automatically check model at startup
☐ Show environment target in scenario model/view

Figure 3.22: Preferences

3.9 Control

3.9.1 Tables and Lists

3.9.1.1 Adding and Deleting

Tables and lists that support the addition and deletion have the following behavior:

1. Selecting a line in the table before pressing the *Add* button inserts a new item under the selected item
2. If no selection is made, item is inserted at the end of the table
3. As for deletion, at least one item has to be selected for the deletion to proceed
4. It is possible to select more than one item for multiple deletion

3.9.1.2 Sorting

Most tables and lists support sorting of the displayed items.

1. Pressing the title of the columns sorts the table in Ascending/Descending order
2. Drag and drop is enabled so custom sorting can be achieved

When saving or opening a project, the GUI makes sure the items are ordered the way they were when the user last modified the configuration file.

3.9.1.3 Colors

Colors might appear in tables where the tree editor can be launched. In fact, the words contained in the cells of the first column will have different colors. They are meant to be a quick display of the status/validity of the tree. Colors mean :

Black	Tree validity is unknown
Green	Tree is error/warning free
Red	Tree has errors
Yellow	Tree has warnings

Tree validity is unknown when the tree validator has not yet been able to validate the tree.(See section 4.4)

Chapter 4

LSD tree editor

The LSD tree editor can be seen as a whole different application. LSD's tree editor displays the information located in xml files in a user-friendly way. It has been developed to help non-programmers to read and write XML files. No deep knowledge of the xml architecture is required. In fact, most users will end up using the tree editor without even knowing what kind of file lays behind.

4.1 Layout

First, let's have a look at the tree editor and its components.

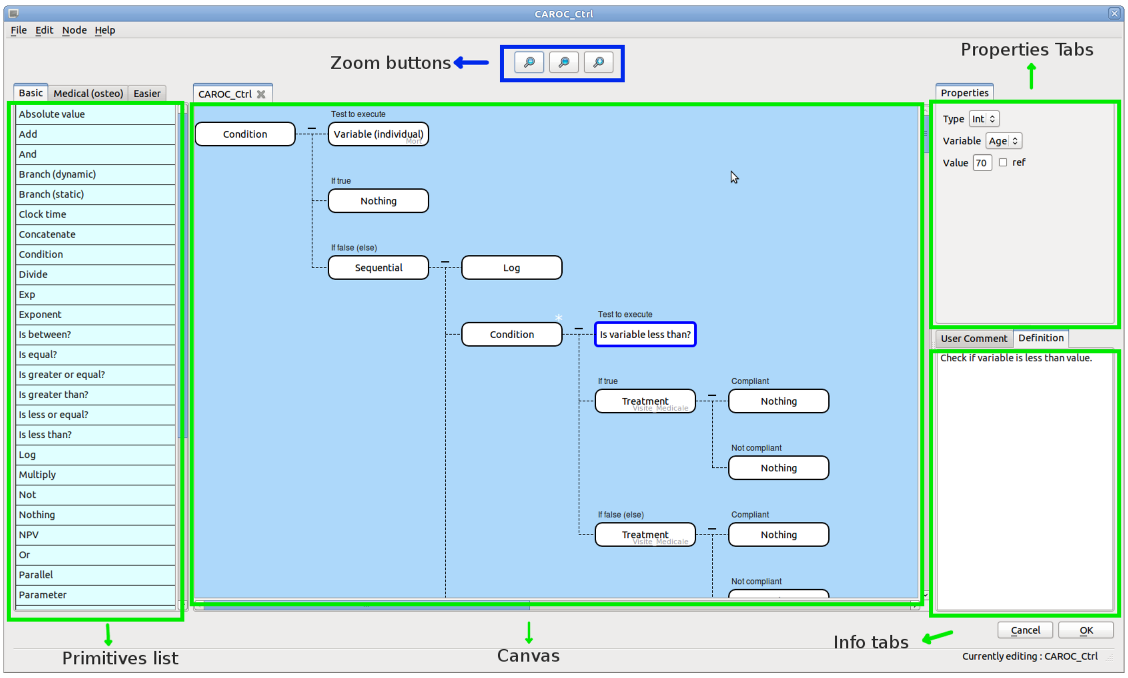


Figure 4.1: Tree Editor

There are 4 principal components in the tree editor:

1. Canvas : Graphical representation of the XML tree
2. Primitives list : Lists the available primitives of the currently loaded plugins
3. Properties Tab : Lists currently selected item's attributes
4. Info tabs : Lists currently selected item's additional information

4.1.1 Canvas

The canvas is where a graphical, user-friendly representation of the XML tree is drawn on a blue background.

Rounded corner rectangle Each rectangle represents a node in the xml tree. The name of the node is located in the center of the rectangle, and is often a good indicator of its function in the tree. A node can have 0, 1 or multiple children. A node normally has a parent, unless it is the root of the tree. Nodes will often be called primitives in the present chapter, as they can take a limited set of values defined in the primitives list. When the borders of a rectangle are drawn in blue, it means the node is the currently selected itme of the tree. The borders can also be painted in red or in yellow. See section 4.4 for more details.

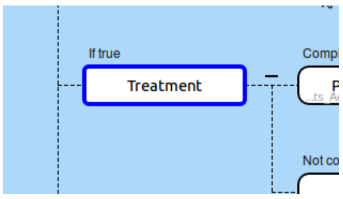


Figure 4.2: A selected node

+ or - signs The plus/minus signs are used to expand/collapse parts of a tree. Simply click on the sign to expand/collapse a subtree. Plus sign's color indicates the validity state of its subtree. See section 4.4 for more details.

Dashed lines Dashed lines are drawn to link a child to its parent.

Asterisks Users can write comments for each nodes of the tree. Normally, user will only write comments for nodes they judge complicated to understand, or for critical parts of the tree. When a node has been commented, a white asterisk is drawn over its rectangle. To see the comment, click on the node and the comment will be displayed in the comment window of the info tabs(see section 4.1.4).

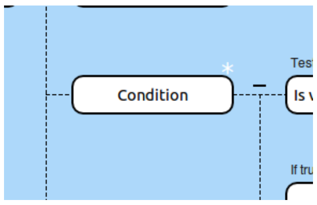


Figure 4.3: A commented node

Node tag Node tags are small pieces of text located over the nodes. They are hints to help the user understand what kind of node the parent expects to have at the position the node tag is located. In figure 4.4, node tags are labeled *Compliant* and *Not compliant*. Hence, first branch is what happens when patient is compliant to the treatment and second branch is what happens when patient is not compliant.

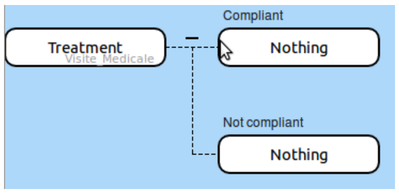


Figure 4.4: Node tags

Branch tag Branch tags are small pieces of text drawn to the left of branches. When a node branches a virtual individual in one of multiple choices, branch tags indicate which choice is associated with which branch. Choices can either be probabilities or possible variable values. Double-click on the branch tag to edit the value. Note that if the choice is a reference to a parameter, it cannot be edited[1]. When a red exclamation mark is shown, it means a value hasn't been yet entered for the branch, and that it is required for the tree to properly work. Note that all branch tags are linked to an attribute[2].

In figure 4.5, a simple *Branch* node is used to show a branch tag. A *Branch* node randomly routes individuals through one of its two branches, depending on a probability. The probablities of the two branches have to sum up to 1. Here, 80% of the individuals take the second branch and are given *Treatment B*, hence the 0.8 probability. We can safely assume 20% go through the first branch and are given *Treatment A*. With this example, we can see the probability of the first branch is not shown in a branch tag. It is always the case for the first child of a primitive, since its alignment with the parent doesn't leave enough space to put a branch tag.

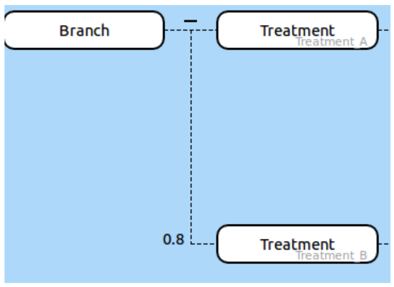


Figure 4.5: Branch tag

[1]See section 4.1.3 for more details on references.
[2]See section 4.1.3 for more details on attributes.

Node hint In previous pictures, little grey text appear in some nodes. Those are node hints. Some nodes display information on their attributes[3] via node hints. Hints can be quite useful to understand a tree without having to click on every node and look at its attributes.

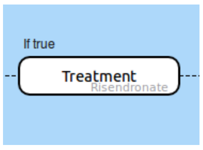


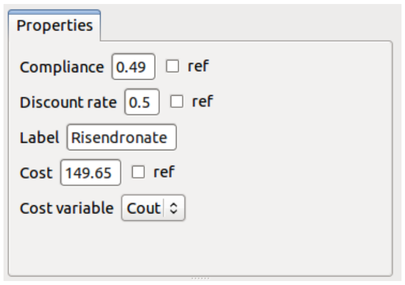
Figure 4.6: Treatment name displayed as a hint

4.1.2 Primitives List

As seen in the previous section, the nodes are the principal elements of LSD trees. Nodes are predefined primitives that are described in plugins(see section 6). Each primitive has a name, a definition, a behavior and some attributes. When put together, these features allow the nodes to be drawn on the screen and provide editing methods, all of this in a user-friendly way. Each plugin corresponds to a tab in the primitives list. Each tab lists the primitives found in its associated plugin. See section 4.2 to learn how to use the primitives to build and edit trees.

4.1.3 Properties Tab

The properties tab lists the attributes of the currently selected node. Nodes can have 0, 1 or multiple attributes. Attributes need to be filled by the user. Some attributes are optional, but it is always recommended to give a value for all attributes since no current method is given to distinguish which attribute is optional and which one is required for the node to be valid. To better understand what the properties tab and the attributes are all about, let's take the example using a *Treatment* node, which have relatively easy and telling attributes.



Properties

Compliance 0.49 ☐ ref
Discount rate 0.5 ☐ ref
Label Risendronate
Cost 149.65 ☐ ref
Cost variable Cout

Figure 4.7: Treatment attributes

For the treatment node to work properly, user has to enter :

1. Compliance : Compliance of individuals to the treatment(value between 0 and 1).

2. Discount rate : Discount rate to apply to treatment's cost(value between 0 and 1)

[3]See section 4.1.3 for more details on attributes.

3. Label : Name of the treatment
4. Cost : Cost of the treatment
5. Cost variable : Name of the individual variable in which to sum the cost

So, when a virtual individual passes in the *Risendronate* treatment(we can assume this treatment is a prescription of risendronate pills), *149.65* dollars adjusted with a *0.5* discount rate is added to its *Cost* variable. Once done, patient have a *0.49%* probability to be compliant and to go through treatment's node first branch. You can do the maths for the second branch.

References If you look at figure 4.7, you can notice two checkboxes next to *Compliance* and *Discount rate* attributes. When these checkboxes are checked, the line edit changes in a combobox. The combo box lists all parameters found in the parameters tab(see section 3.3). When the same node is found at multiple locations in a tree, it can be very useful to have a parameter store an attribute's value. Hence, user can quickly modify the value of an attribute by modifying the value of the parameter, instead of seraching for all similar nodes and modifying the attribute's value one by one.

As said before, the combobox lists all parameters found in the parameters tab. It also has one more row which is labeled *Add new parameter*. This way user can add parameters on the fly while creating trees, instead of having to close the tree and switch to the parameters tab. However, only the name of the newly added parameter can be supplied. The rest of the data has to be entered using the parameter tab. Figure 4.8 shows the parameter's name request that is displayed when a parameter is added via the tree editor.

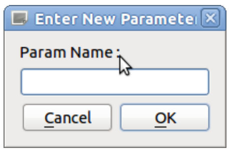


Figure 4.8: Add parameter in tree editor

Attributes(Additional Info) Some attributes have special behaviors. Some of these behaviors are quite easy to understand by the user, but others are more difficult to grasp. First behavior is the combobox behavior. It's probably the most easy to understand and in fact, such a behavior makes the tree editor a much more interesting tool than it would be without it. When an attribute's value is limited by a fixed set of choices, these choices are listed in a combo box. It is the case for attributes that refer to parameters, variables or processes. Another behavior is the dependency behavior. When an attribute uses a reference to a parameter, SCHNAPS can tell the type of the data by looking at the type of the parameter. But, when the reference checkbox is unchecked, SCHNAPS needs to be told what is the type of the attribute. So, a type attribute appears, and user has to select the attribute's type in a combobox. If the reference checkbox is later rechecked, the type attribute disappears. It is called a dependency behavior because the type attribute is dependent of the state of the other attribute. Finally, the most complicated behavior is found with attributes that indicate how branching is done in a node's children. A small example can be used to fully understand the concept.

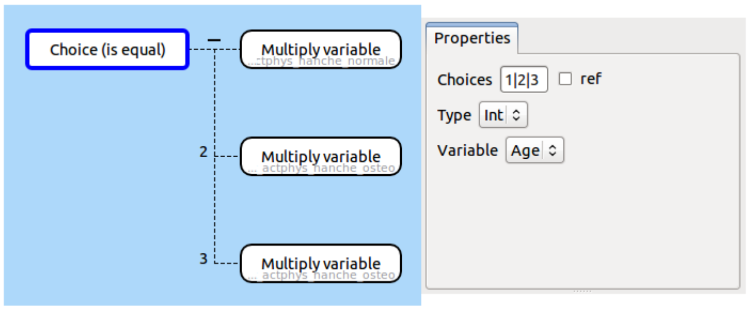


Figure 4.9: Attributes and branching

Here the, *Choice(is equal)* node looks at the *Age* variable of the individual and directs the individuals through the right child. The *Choices* attribute is where you define the age for each branch. Here, a 1 year old individual would take the first branch, a 2 years old individual the second branch and a 3 years old individual the third branch. Note each | character between the choices : they are needed to separate the choices. The GUI automatically sets the branch tags when you modify a branching attribute. The GUI also automatically modifies the branching attributes when you modify the choices via a branch tag. Note that branching attributes can often refer to a vector parameter. When a branch attribute refer to a parameter, branch tags are read-only.

4.1.4 Info tabs

These are two simple tabs. One tab is the comment tab, where you can enter comments for the currently selected node. The other tab is the definition tab, which holds a short description of the node's primitive.

4.2 Working with the tree

Creating a tree is usually quite intuitive, but some features can be used to help fasten tree modifications. Building the tree is mainly done by replacing *Nothing* nodes with primitives from the primitives list. To do so, simply click on a primitive in the list and while keeping the left mouse button pressed, drag and drop the primitive on a *Nothing* node. You can also replace existing nodes that aren't *Nothing* nodes. However, be careful doing so, as the replaced node's subtree will also be deleted.

Adding children To add a child to a node, select the node and press *Ctrl-A*. A child will be inserted as the the currently selected node's last child. You can also add a child to a node by clicking the right mouse button and selecting the *Add Child* option.

To add multiple children at one, select the node and press *Ctrl-Shift-A*. A window will pop up and ask you to type the number of chilren to add. Children will be inserted as the the currently selected node's last children. You can also add multiple children to a node by clicking the right mouse button and selecting the *Add Mutliple Children Option.*

Adding siblings To add a sibling to a node, select the node and press *Ctrl-Up*. A sibling will be inserted before the currently selected node. You can also press *Ctrl-Down* to insert a sibling after the currently selected node. Both options are available by right-clicking on a node that you want a sibling to be added to.

Copy-Paste It is possible to copy a node. To do so, select a node and press *Ctrl-C*. Note that the node's subtree is also copied. To paste the copied node, select another node you want replaced and press *Ctrl-V*.

Deleting nodes To delete a node, select the node and press *Ctrl-X*. Note that if parent primitive already has its minimum number of children, the node will be replaced by a *Nothing* node. Deleting root node also replaces the node by a *Nothing* node.

Double-Clicking Nodes with variables referencing other processes can be double-clicked. It opens a new tab with the referenced process tree.

Navigating You can navigate in the tree using the arrows. Left arrow selects currently selected item's parent, right arrow selects currently selected item's first child, up and down arrows respectively selects currently selected item's previous and next sibling.

Redo/Undo Tree modifications can be undone or redone by using the undo and redo shortcuts, respectively *Ctrl-Z* and *Ctrl-Y*. Note that undo and redo only take care of the modifications in the tree structure(attributes modifications are not pushed on the undo stack).

Zooming Sometimes, zooming can be useful when editing a big tree. See figure 4.1 for the location of zooming buttons. Left button zooms out, right button zooms in and central button resets the zoom at its original value.

4.3 Menu

Tree editor's menu is independant of the main window's menu. It contains options that are specific to the tree editor.

4.3.1 File

Open Opens a windows where the user can select a tree of a process or simulation variable of the currently loaded project. The tree is loaded as a new tab in the Canvas list and can therefore be edited.

Save Saves the modifications made to the tree. *Save* command can be triggered using the *Ctrl-S* shortcut. Warning : Saving in tree editor doesn't save the project itself. For modifications to be saved on the file sytstem, use the main window's save system(see section 3.8.1.3).

Open plugin Opens the plugin manager so plugins can be added on the fly while modifying trees. *Open plugin* command can be triggered using the *Ctrl-Shift-O* shortcut.

Save tree picture Saves a picture of the current tree on the file system. A file explorer is opened so the user indicates the location and the file name of the picture. Then, the user is asked if he wants a color or grayscale image. Note that both types are saved in a .SVG format. More formats might be available in the future.

Exit Leaves the tree editor. *Exit* command can be triggered using the *Ctrl-Q* shortcut.

4.3.2 Edit

Edit menu offers some commands to trigger actions of section 4.2. It also offers a *Take screenshot* command that saves a picture of the whole tree editor on the file system. *Take screenshot* command can be triggered using the *Ctrl+T* shortcut.

4.3.3 Node

Node menu offers some commands to trigger actions of section 4.2.

4.3.4 Help

Help menu contains the *About* command. The *About* command displays information about the tree editor. It also lists what version of Python, Qt and PyQt is installed on the computer, as well as the name of the operating system.

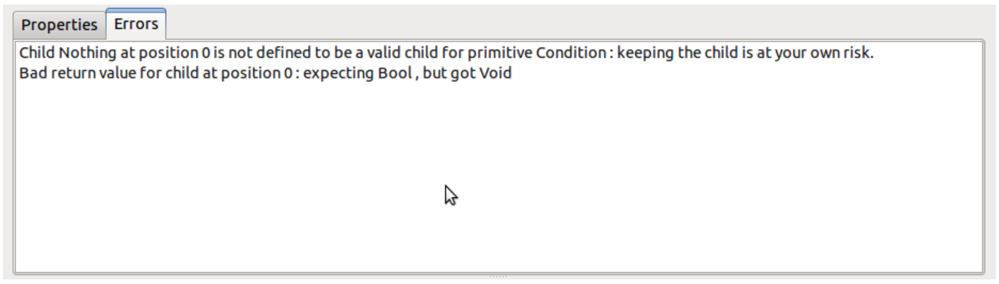
4.4 Validator

The validator is a tool developed for the real-time analysis of XML trees modified in LSD's tree editor. It uses the information found in the plugins to validate the trees built by users. This way, beginners can quickly learn the to-do's and not-to-do's with the tree editor. Errors are corrected at the source and do not need SCHNAPS to crash for the users to correct them, hence fastening the whole simulation process.

When an error is found in the tree editor, the border of the node associated with the error changes color. Red color is for errors and yellow color is for warnings[4]. Warnings need to be taken seriously. The validator can not determine if they are going to cause a crash at runtime, but often, it is the case. As for errors, they are for sure going to cause a problem at runtime.

[4]Unlike tables, a black border color indicates a valid node, not an unknown validity state node

When errors are found in a node, click on it. An *Errors* tab will appear next to the *Properties* tab. *Errors* tab lists all information about errors found in the currently selected node. Once all errors are corrected, the tab disappears. See figure 4.10.



```
Child Nothing at position 0 is not defined to be a valid child for primitive Condition : keeping the child is at your own risk.
Bad return value for child at position 0 : expecting Bool , but got Void
```

Figure 4.10: Error tab

The validator is also used as an asynchronous way of validating the project when used in tables and via the main Window's *Tools* menu[5]. The validator can be slow when dealing with big trees. It is in the user's advantage to minimize the size of its processes, even if it means ending up with more of them.

4.5 Types

Typing is a fundamental aspect of programmation. SCHNAPS and LSD's input GUI were developed in a way that a user only has to enter type information where it is absolutely needed. In the GUI, each time the user will be asked to enter type information, available types will be listed in a combobox. Currently available data types are:

1. Bool : a boolean value(true or false)
2. Double : a floating point number in range $\pm 1.7 \times 10^{308}$
3. Float : a floating point number in range $\pm 3.4 \times 10^{38}$
4. Int : an integer number in range ± 2147483647
5. Long[6] : an integer number in range $\pm(2^{63} - 1)$
6. String : a word
7. UInt : a positive integer type in range 0...4294967295
8. ULong[7] : a positive integer type in range $0 ... 2^{64} - 1$

[5]see section 3.8.3.4
[6]On 32 bits OS, Long is the same size as the Int type
[7]On 32 bits OS, ULong is the same size as the UInt type

Chapter 5

LSD Wizard

It is not clear at this moment what is going to happen with the wizard. It is currently developed but rarely used.

Chapter 6

Plugins

When a new C++ plugin developed on SCHNAPS is available, the GUI doesn't know anything about it. To bring these new features to LSD's input GUI, developers have to write files. Those files have a .xsd extension, and are called plugins in the LSD jargon. These files contain information on available nodes for the tree editor. The more time goes, the more features are developed for SCHNAPS and the GUI, the more vast are the possibilities for creating complex decision trees. Plugins can be added to a project with the Plugin Manager(see section 3.8.1.5). When a plugin has been added once to a project, the GUI loads the plugin automatically when the project is opened.

Plugins are currently being modified. A list of all plugins and their content will soon be available.