APIMOU 

Version 2.6

User's manual

WARNING – DISCLAIMER

This software is provided “as is”, without any kind of warranty, express or implied, including but not limited to warranties of merchantability, fitness for a particular purpose, or error-free operation.

Under no circumstances shall the author be held liable for any damage, data loss, hardware failure, or any other adverse effects resulting directly or indirectly from the use of this software, even if the author has been advised of the possibility of such damage.

The use of this software is entirely at your own risk.



Table of contents

1 Presentation 3
2 Licensing information 3
3 Quick starting guide 4
3.1 Example of a program in combinatorial logic 4
3.2 Example of a program in sequential logic 4
4 Variables 7
4.1 Mnemonic change 7
4.2 Input variables 7
4.3 Output variables 7
4.4 Step variables 8
4.5 Internal bits 8
4.6 Timing variables 8
4.7 Counting variables 9
5 Logical equations 9
6 Timers and Counters 9
7 Grafcet 11
7.1 Steps and actions 11
7.2 Transitions and transition-conditions 12
7.3 Directed links 13
7.4 Parallel sequences 13
7.5 Selection of sequences 14
7.6 Multiple Grafcet 14
7.7 Multipage editing 14
7.8 Forcing orders 14
7.9 Various editing functions 15
8 Special functions 17
8.1 Clock signal or flashing 17
8.2 Rising or falling edge 17
8.3 Initialization bit 18
9 Operation in simulation mode 19
10 Operation in connected Mode 19
11 Arduino code generation 20
12 Direct upload to an Arduino board 21
13 ESP32 code generation 22
13.1 Overview 22
13.2 Prerequisites : 23
13.3 Station and Access Point modes 23
13.4 Procedure 24
14 Direct upload to an ESP32 26
15 File management 27

1 Presentation

Apimou is a Programmable Logic Controller (PLC) simulator that allows the creation and testing of simple programs implementing combinatorial and sequential logic functions. It can work in simulation mode or in connected mode, connected to a software or real operating part. It has 32 binary inputs and 32 binary outputs as well as internal variables (step variables, timing variables, counting variables, internal bits).

The outputs can be direct functions of the inputs, and/or depend on a sequence materialized by a grafcet.

The Apimou.zip archive includes :

-> Apimou.exe, the main program (which may be self-sufficient in simulation mode),

-> Apimou.pdf, this document,

-> Bees.exe, Afficheur.exe, Feux_tri.exe, Convoyeur&Verins.exe, Ascenseur.exe, Chariots.exe (and maybe others) which are the currently available software operating parts. Apimou will only be able to communicate with one operative part at a time,

-> a Sources folder containing the source code (C++Builder 10.3.3) of the Bees operative part. This code is provided to understand the communication protocol between Apimou and the operative parts. It is perfectly possible to dispense with opening this folder.

-> A folder *Fichiers api* which contains, in the corresponding subfolders, some example programs for the proposed operating parts. The subfolders "pour Afficheur", "pour Ascenseur", "pour Feux tricolores", "pour Convoyeur et vérins" contain only one file "config_(name of the OP).api". When this file is opened in Apimou, we simply find mnemonics (for inputs/outputs) corresponding to what we see in the corresponding operating part (labels of buttons, sensors, etc.). The program still has to be created. Exercises relating to these operating parts can be found on the Internet, and Apimou enables the solutions of these exercises to be tested in practice.

2 Licensing information

Apimou is freeware.

The Apimou source code is available on the author's website, or on request.

Apimou uses the DOSCommand component of TurboPack, licensed under MPL 1.1.

Apimou uses the Boost library, licensed under Boost Software License 1.0.

Apimou uses DelphiZXingQRCode of Foxit Software, licensed under Apache-2.0.

3 Quick starting guide

The following two examples will be tested in simulation mode.

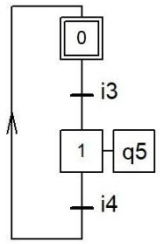
3.1 Example of a program in combinatorial logic

1. Start Apimou
2. Check that Simulation mode is selected (Mode -> Simulation)
3. Put the PLC in Run (PLC -> Run)
4. Click on the cells %I1.0, %I1.1, etc. (corresponding mnemonics i0, i1, etc.) : you will see the input leds light up on the PLC (and that's all: for the moment the PLC is not programmed)
5. Put the PLC in Stop (PLC -> Stop)
6. Click on the Outputs tab, and in the Equation column of the output %Q2.3 (mnemonic q3) type i2+i7
7. Put the PLC in Run (PLC -> Run)
8. Click on the Inputs tab and test the program: click on the boxes %I1.0, %I1.1, etc. You can see that output 3 (address %Q2.3, mnemonic q3) is activated (the corresponding output LED lights up) if you click on %I1.2 (mnemonic i2) or %I1.7 (mnemonic i7).

The PLC was programmed with a purely combinatorial example (q3 = i2 + i7) and tested in simulation mode.

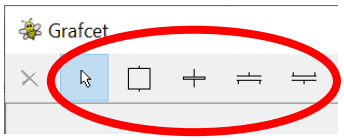
3.2 Example of a program in sequential logic

Let's program the following grafcet:



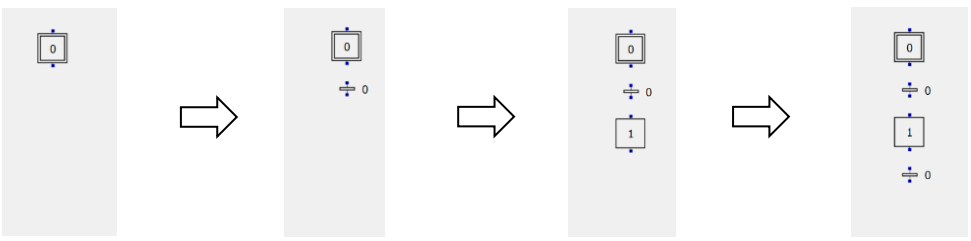
(made with Editsab)

1. PLC -> Stop.
2. File -> New (we delete everything).
3. Show -> Grafcet. In the toolbar of the sheet which appears are the buttons corresponding to the elements necessary for the constitution of a grafcet: step, transition, activation and synchronization of parallel sequences.

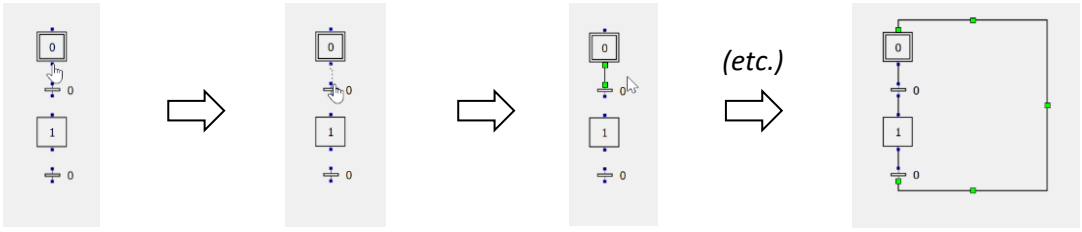


4. Choose a step and place it on the form, then a transition and place it underneath. Repeat until you have all the elements you want.

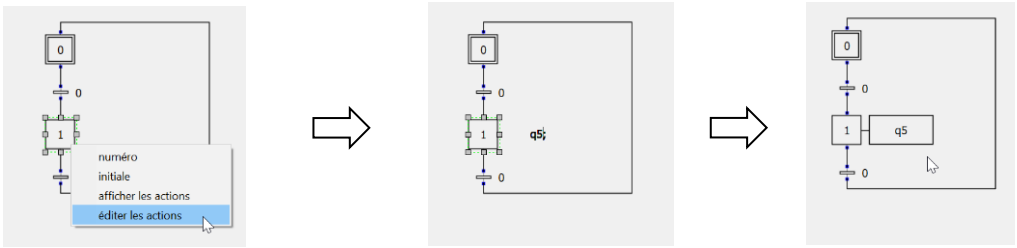
   The first step inserted on the form is by default an initial step (double square). The step numbers are automatically incremented. The transitions have by default always false conditions (value 0).



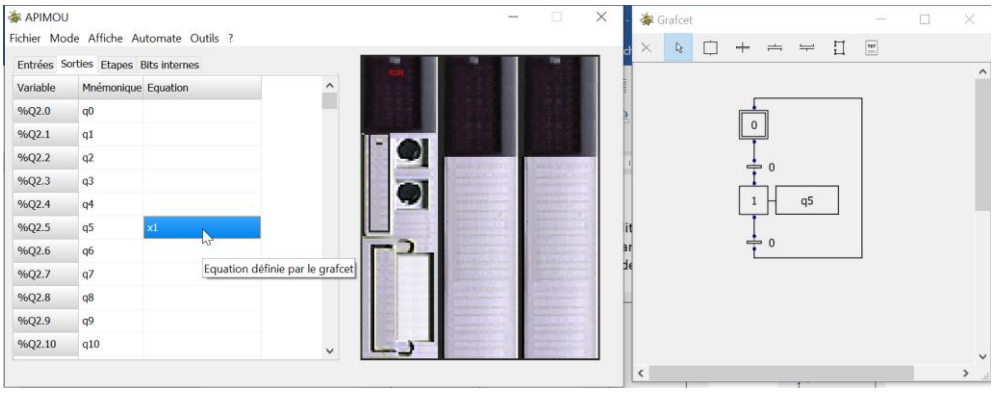
5. Link the steps to the transitions and the transitions to the steps by clicking on one link point (small blue square), pulling and releasing on the second link point. For the last link (last transition to initial step), click on the right handle (small green square) and pull it to the right to have enough space for the actions.



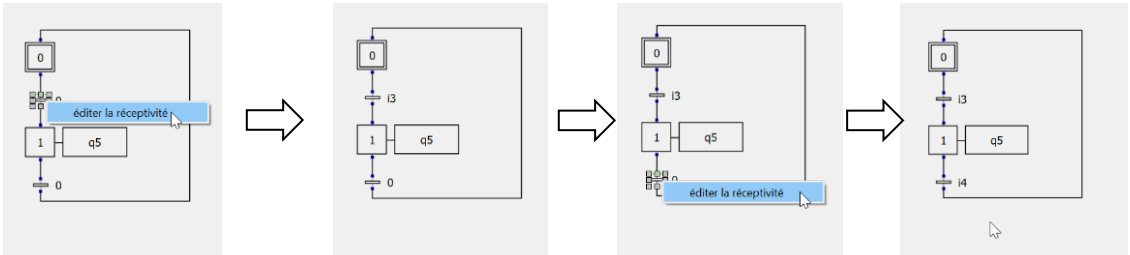
6. Right click on step 1 and choose *edit actions*. Enter the chosen action (here q5). The semicolon ";" is used to separate the actions. As there is only one here, you can delete it (this is not an obligation). Validate by clicking on a free space of the form or with the Ctrl+Enter key combination.



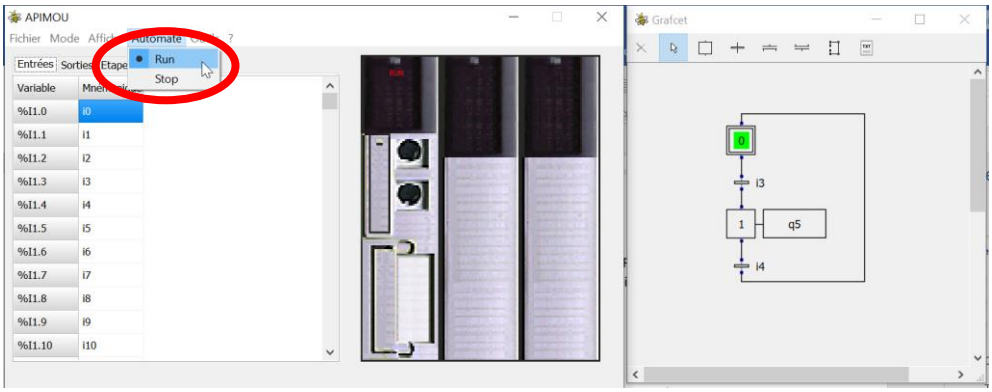
You can see in the main sheet, by clicking on the outputs tab, that the equation for the output q5 has been automatically entered (q5 = x1).



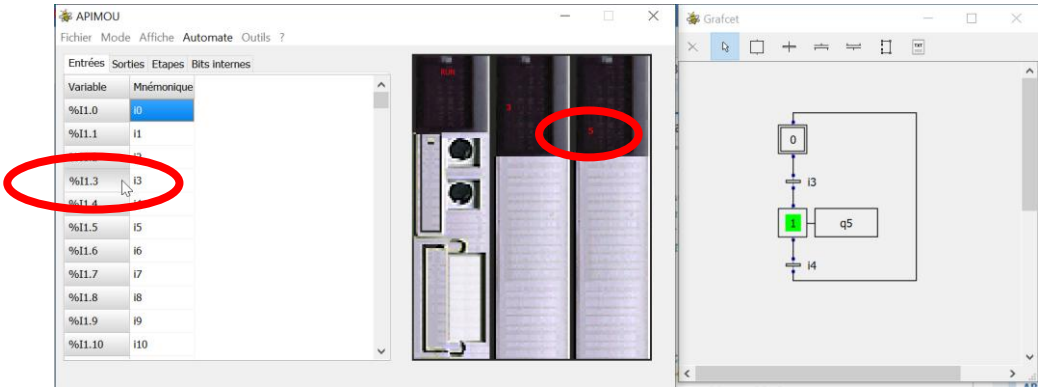
7. Right click on the first transition and choose *edit the transition condition*. Enter the chosen condition (here i3). Validate by clicking on a free space of the form or with the Ctrl+Enter key combination. Do the same for the second transition, which condition is i4. The input of the grafcet (of the program) is finished.



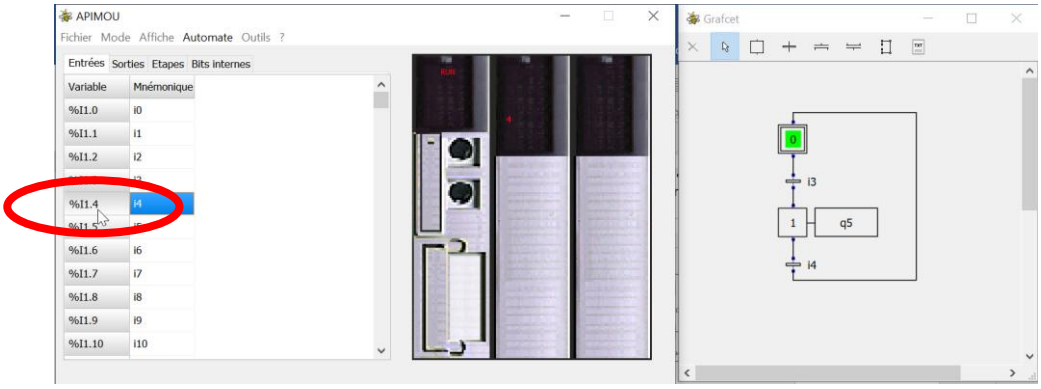
8. Test of the program: in the main file, menu PLC -> Run. A green square (the "token") indicates on the grafcet that step 0 (initial) is active.



9. Click on the tab Inputs then click on the box %I1.3 (mnemonic i3): we see on the grafcet that the step 0 was deactivated and that the step 1 is now active. The corresponding action is carried out, thus the output q5 passes to 1 (the led of the output 5 lights up).



10. Click on the cell %I1.4 (mnemonic i4) : we see on the grafcet that the step 1 was deactivated and that the step 0 is now active, we return to the initial step. The output q5 returns to 0 (the corresponding output LED turns off).



4 Variables

Input, output, step and internal bit variables are identified by a pair (address, mnemonic). Timer and counter variables are identified only by a mnemonic (t0 to t7 for timers, ct0 to ct7 for counters). Addresses are only used here for identification purposes, they cannot be used in logical expressions. Mnemonics are essential and are used in logical expressions. Each mnemonic is unique.

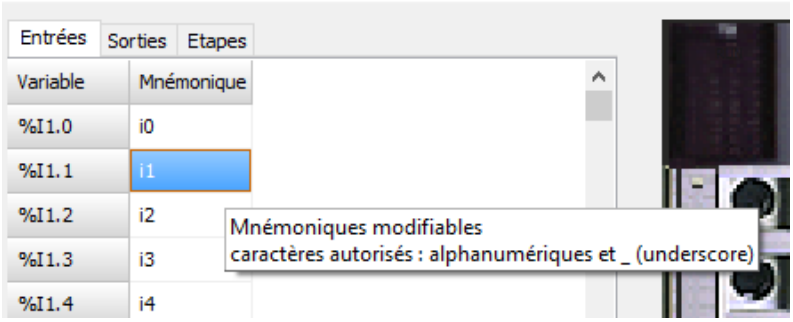
4.1 Mnemonic change

The default mnemonics can be changed from the main form, provided the PLC is in STOP.

The characters allowed for mnemonics are all alphanumeric characters (unaccented) and the underscore (_). A mnemonic may consist of only one character, but mnemonics 0 and 1 are reserved (0 corresponding to an always false condition and 1 to an always true condition). The mnemonics t0 to t7 and ct0 to ct7 are also reserved (timing and counting variables).

Mnemonics are case sensitive: A1 and a1 are two different mnemonics.

To modify a mnemonic, select it and click to enter the table editing mode:



4.2 Input variables

They correspond to the states of the on/off sensors connected to the PLC input card. Their values are read by the PLC. In simulation mode, these variables are forced: they are assigned a value by clicking on the %I.x cells.

4.3 Output variables

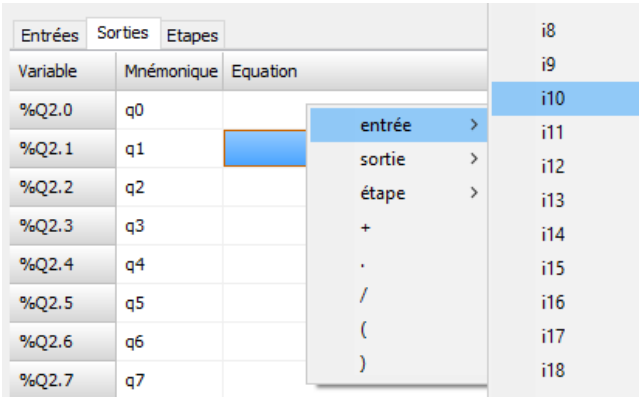
They correspond to the states of the (pre-, micro-) switching actuators connected to the output card of the PLC.
Their values are calculated by the PLC.
These calculations are the results of logic functions which are defined here under the Outputs tab of the main form, in the Equation column.

If one (or more) grafcet was (were) created, the equations of the outputs corresponding to the actions of this (these) grafcet are automatically defined and it is not possible to modify them. One can also not program any action in the grafcet and enter manually the equations of the outputs, which leaves more flexibility.

There are two ways to enter a logical equation for an output variable:

- either by selecting the cell and clicking on it to enter the edit mode. The equation is then entered using the keyboard.

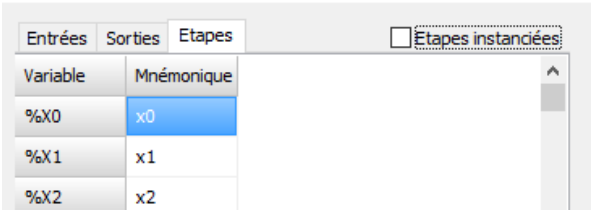
- or by selecting the cell and then right-clicking, without entering edit mode, to select the necessary variables and operators:



4.4 Step variables

They correspond to the states of the steps of the grafcet(s). They are internal variables whose values are calculated by the PLC.

By default, all step variables can be viewed, even if not all are used.
To view only the used (instantiated) steps, check the *Instantiated steps* box on the Steps tab of the main form :



4.5 Internal bits

The internal bits are used for what we want, they are defined, like the output variables, by a logic equation.

For example, they can be used to express:

- a "complex" logical condition, which will thus be summarised as a variable,
- a rising or falling edge,
- an initialisation bit.

One can also use them to program a grafcet in logical equations.

4.6 Timing variables

Timer variables (t0 to t7) are internal variables that change to 1 when the corresponding timer (T0 to T7) has expired (see § Timers and Counters). They can be used as needed to express an output, an internal bit, a transition condition or any other logical expression.

4.7 Counting variables

The count variables (ct0 to ct7) are internal variables that change to 1 when the corresponding count (CT0 to CT7) is completed (see § Timers and Counters). They can be used as needed to express an output, an internal bit, a transition condition or any other logical expression.

5 Logical equations

The logical equations make it possible to define the outputs, the transition conditions of a grafcet, the internal bits. They also make it possible to declare the conditions of activation or reset of the function blocks (temporization, counting). The following rules apply in all cases:

The operands are the mnemonics of the input, step, timer, count, possibly output variables, and internal bits.

The only three operators used are AND symbolised by a dot (.), OR symbolised by a plus (+) and NOT symbolised by a slash (/).

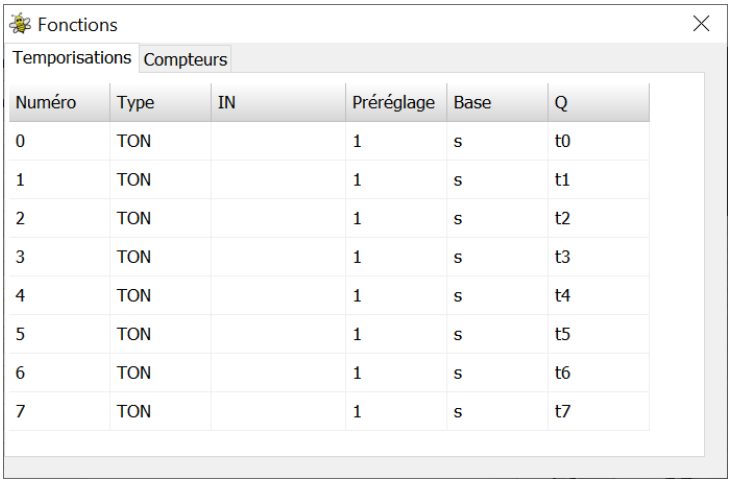
Example : I1 OR i2 AND i3 AND NOT i4 will be written as i1+i2.i3.i4/

N.B. :

- the / operator is applied to the preceding variable or expression in parentheses,
- an equation is declared directly in the space that receives it, it does not start with an equality sign,
- brackets are used to prioritise operations if necessary: i1+i2.i3 corresponds to i1+(i2.i3), different from (i1+i2).i3,
- the . operator is not implicit: i1.(i2+i3) cannot be written i1(i2+i3),
- it is possible to separate operators from variables with spaces,
- the values 0 and 1 (false and true) can be used (1 allows for example to declare a transition condition always true).

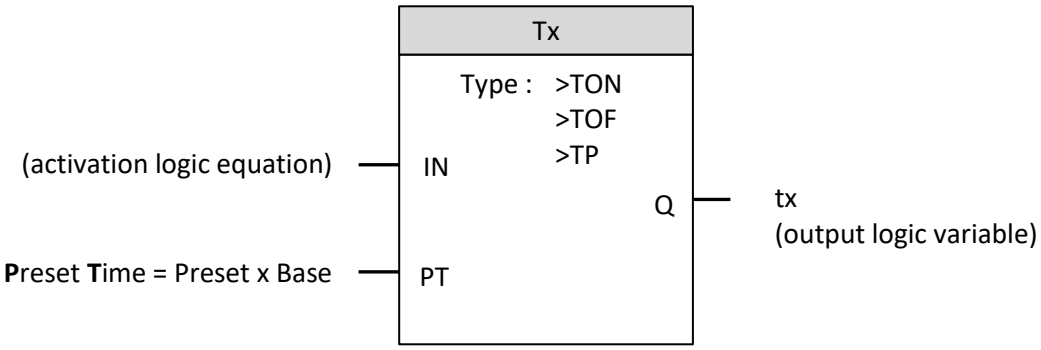
6 Timers and Counters

Apimou allows the use of 8 timers (numbered from 0 to 7) that can be configured as "TON", "TOF", "TP" and 8 counters (also numbered from 0 to 7) of type "CTU". These functions can be accessed via the menu View -> Functions.



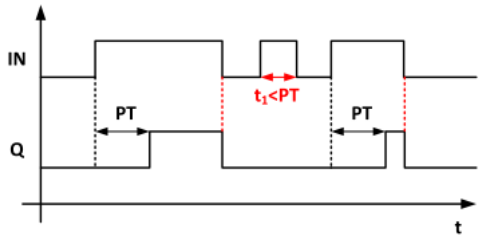
Each timer or counter has an associated output logic variable Q imposed (e.g. t4 for timer 4 and ct3 for counter 3).

Timers can be represented by the following function block:



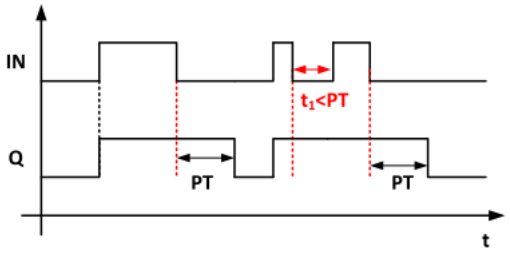
The activation logic equation (IN) can be entered from the keyboard or from the context menu that appears when the cell is right-clicked. It can, like any other equation, consist of only one variable.

Chronogram of a TON timer :



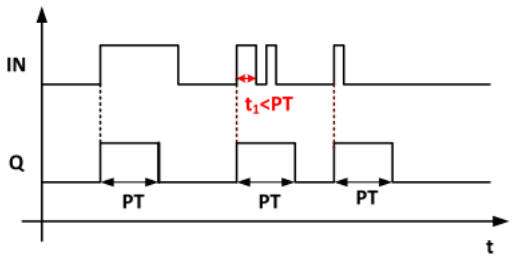
source : International Journal of Mechatronics and Applied Mechanics, 2018, Issue 4

Chronogram of a TOF timer :



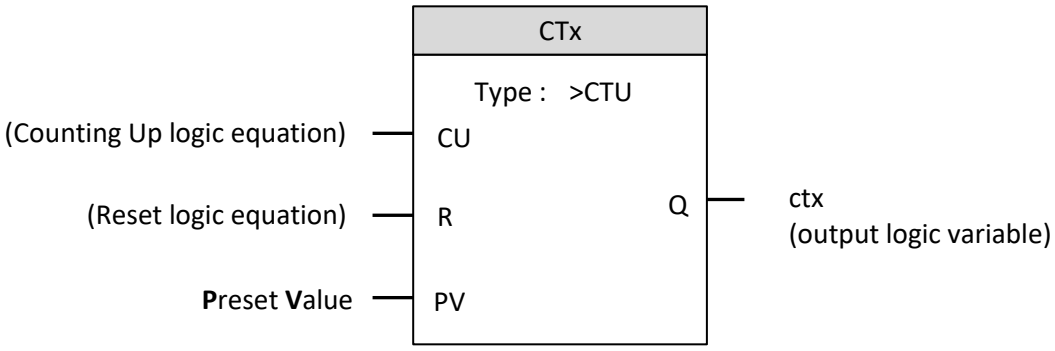
source : International Journal of Mechatronics and Applied Mechanics, 2018, Issue 4

Chronogram of a TP timer (pulse of fixed duration):



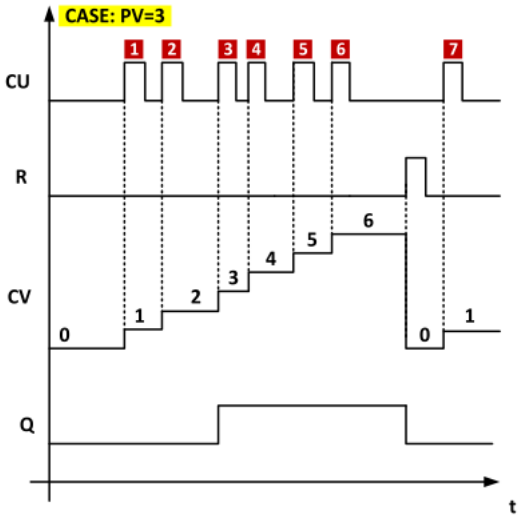
source : International Journal of Mechatronics and Applied Mechanics, 2018, Issue 4

Counters can be represented by the following function block:



Count (CU) or reset (R) logic equations can be entered using the keyboard or the context menu that appears when the cell is right-clicked. They can, like any other equation, consist of only one variable.

Chronogram of a CTU counter (for PV = 3) :



CV : Current Value

source : International Journal of Mechatronics and Applied Mechanics, 2018, Issue 4

7 Grafcet

The essential to know about how to draw a grafcet is described in the corresponding paragraph of the quick starting guide.

7.1 Steps and actions

Steps can be simple or initial (activated when the PLC switches to RUN). The choice is made by a contextual menu (right click on the step -> initial)

Each step has a unique number. It is also possible to modify it by contextual menu (right click on the step -> number, enter the desired value, validate by clicking on a free space of the form or with the combination of keys Ctrl+Enter).

The definition of an action associated with a step is done by contextual menu (right click on the step -> edit actions, enter the desired expression, validate by clicking on a free space of the form or with the combination of keys Ctrl+Enter).

If several actions are associated with the same step, they must be separated by a semicolon (;).

A continuous action must be declared by the mnemonic of the corresponding output. It is not possible to assign an internal bit to an action (it can always be done outside the grafcet by defining the equation of the desired bit, m2=x5 for example if one wants to activate the m2 bit with the step 5).

A conditional action must be declared using the *condition ? output* syntax.

The equations of the outputs corresponding to the actions of the grafcet are automatically defined and it is not possible to modify them. One can also not program any action in the grafcet and enter manually the equations of the outputs.

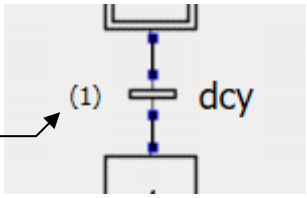
Examples of actions :

Grafcet	Grafcet Apimou	Output equations (automatically defined)
3 — q7	3 — q7	q7 = x3
3 — q7 9 — q7	3 — q7 9 — q7	q7 = x3 + x9
3 — q7 q10	3 — q7 ; q10	q7 = x3 q10 = x3
I0.(i1/+i4) 3 — q7 q10	3 — i0.(i1/+i4)?q7 ; q10	q7 = x3.(i0.(i1/+i4)) q10 = x3
I0.(i1/+i4) 3 — q7 q10 9 — q7	3 — i0.(i1/+i4)?q7 ; q10 9 — q7	q7 = x3.(i0.(i1/+i4)) + x9 q10 = x3

7.2 Transitions and transition-conditions

Transitions are marked with a number which uniquely identifies them. These designations can be shown or hidden by right-clicking on any transition and choosing "show designations"

Transition designation



A transition has by default an associated condition which is always false (0), which implies that even if the transition is enabled (all immediately preceding steps linked to this transition are active), it will never be cleared. The modification of a transition condition is done by contextual menu (right click on the transition -> *edit the transition condition*, enter the desired expression, validate by clicking on a free space of the form or with the combination of keys Ctrl+Enter).

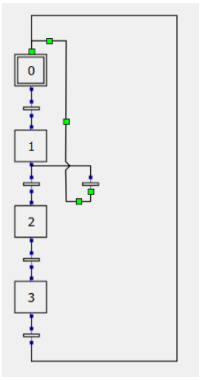
An always true transition-condition is defined by the value 1. If a transition-condition is not defined, it is equivalent to an always false condition.

7.3 Directed links

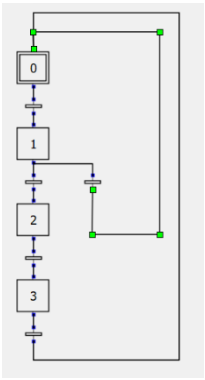
Step <-> transition links can only be drawn in the direction of their reading :

- ✓ For a step->transition link, we start from the downstream link point of the step to the upstream link point of the transition, the reverse is impossible.
- ✓ For a transition->step link, we start from the downstream link point of the transition to the upstream link point of the stage, the reverse is impossible.

If the default drawing mode of the link (automatic) is not visually satisfactory, because of lines which may cross each other, the "manual path linking" mode can be selected in the toolbar, which allows multi-segment lines to be drawn and these segments to be arranged as desired.



Automatic path linking

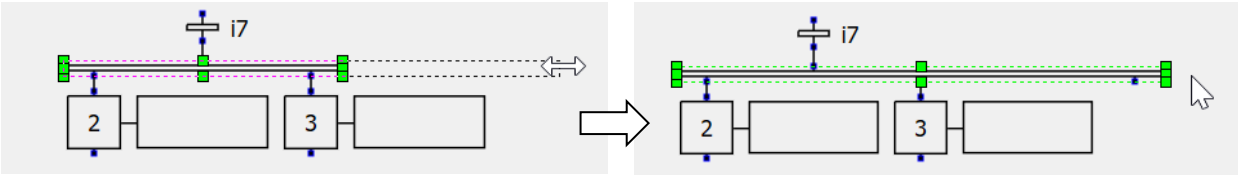


Manual path linking

Constructions that do not correspond to the rules of the grafcet, in particular the step-transition alternation, are not allowed. Everything that can be drawn is therefore compliant.

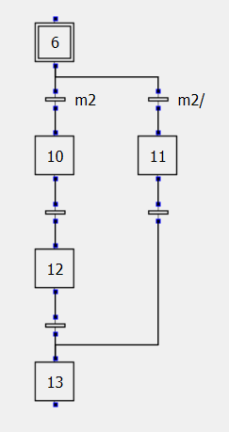
7.4 Parallel sequences

These constructions are used for simultaneous (parallel) sequences.
The symbol for activation of parallel sequences has one input and, by default, two outputs. If you want to start three (or more) parallel sequences, simply resize the block by dragging it to the right:



The principle is the same for synchronization (end) of parallel sequences.

7.5 Selection of sequences



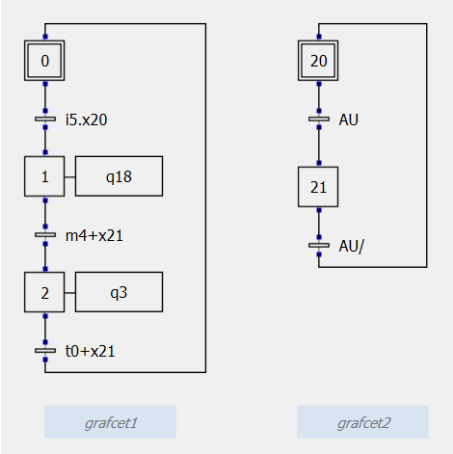
These constructs are used for sequence selections (or jumps).

There is nothing special to report here, it is simply necessary to ensure that the transition-conditions from which the selection is made are (logically or physically) exclusive, to avoid token proliferation.

7.6 Multiple Grafcet

It is possible, for the same application, to enter several grafcet, synchronized or independent.

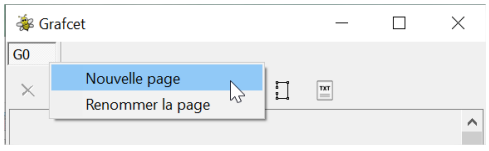
Example of synchronized grafcet:



7.7 Multipage editing

If several grafcets are to be entered, they can each be entered on a different page. If you want to structure the program by prioritizing the grafcets and using forcing orders, this is mandatory: each grafcet is named in this way, and a forcing command can then be triggered to designate a specific grafcet.

By default, there is only one page in the editing space, named "G0". You can add a new page by right-clicking on the tab and selecting "new page" from the contextual menu that appears.



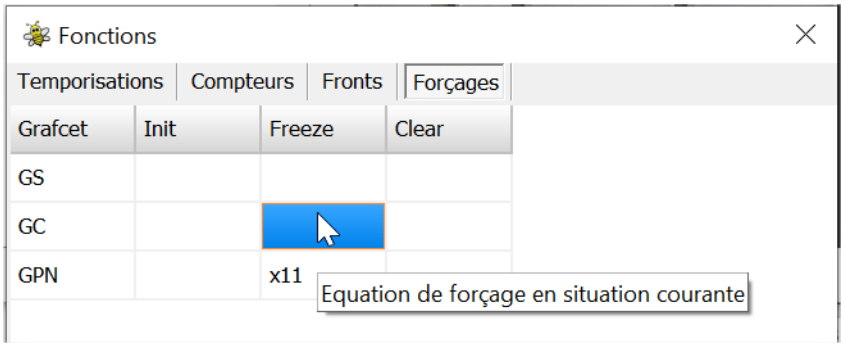
7.8 Forcing orders

A complete control logic system often needs to be structured in such a way that it can handle and coordinate several operating modes, taking into account all possible situations (cf. GEMMA and the hierarchical grafcets derived from it). In this case, it can be useful to use forcing orders to immediately put a grafcet in a particular situation.

Apimou allows three forcing actions for each grafcet: INIT, FREEZE and CLEAR. These actions are available in the menu View -> Functions, Forcing orders tab.

INIT	Resets the grafcet: the initial step is activated (the initial steps if there are several), the others deactivated.
FREEZE	Freezes the grafcet in the current situation: it no longer evolves, and no transition can be cleared, whatever the transition-condition values.
CLEAR	forces the grafcet in an empty situation: all steps, initial or otherwise, are deactivated

Each of these actions is triggered by a logical equation. It can, of course, be reduced to a simple step variable.



The above configuration, for example, corresponds in standard grafcet representation to :



(réalisé avec Editsab)

NB: if two or more of these three commands are active simultaneously for the same grafcet (which should not happen), CLEAR takes precedence over INIT, which in turn takes precedence over FREEZE.

7.9 Various editing functions

To insert several blocks (step or transition or comment) without clicking each time on the toolbar: press the *Shift* key on the keyboard and the *Step button* or *Transition button* or *Comment button* on the toolbar, then release the *Shift* key. Click on the editing space as many times as necessary, moving the mouse to distribute the blocks. Click on the Cancel inserting toolbar button to stop.

To align several elements vertically : select the elements (click-drag), then click the *right mouse button* and choose *Align vertically* from the context menu.

To distribute several elements vertically : select the elements (click-drag), then click the *right mouse button* and choose *distribute vertically* from the context menu.

To increase or decrease the space between several vertically aligned and distributed elements : press the *Alt* key on the keyboard and, while holding down the key, rotate the *mouse wheel* up or down.

To insert a linear sequence of 1 to 9 steps : position the mouse on the editing space where the sequence is to be inserted, then simultaneously press the *Ctrl* key on the keyboard and any of the *keys 1 to 9* on the keyboard (not those on the numeric keypad).

To zoom in/out : press the *Ctrl* key on the keyboard and, holding down the key, turn the *mouse wheel* up or down.

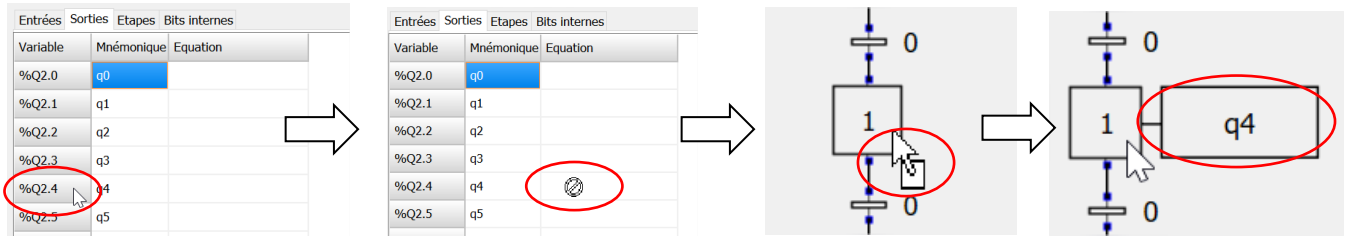
To return to the default zoom level (100%) : click on the *central mouse button*.

To enter actions and transition conditions by drag & drop :

Inputs, *outputs*, *steps* and *internal bits* can be dragged and dropped onto the grafcet by left-clicking on the corresponding Variable box on the main form, then moving the mouse towards the grafcet and releasing the button on a step or transition. For *timers* and *counters*, you can do the same by clicking on the output box (Q) of the Functions form corresponding to the desired timer or counter.

Steps accept only *output variables*, while *transitions* accept *all* variables.

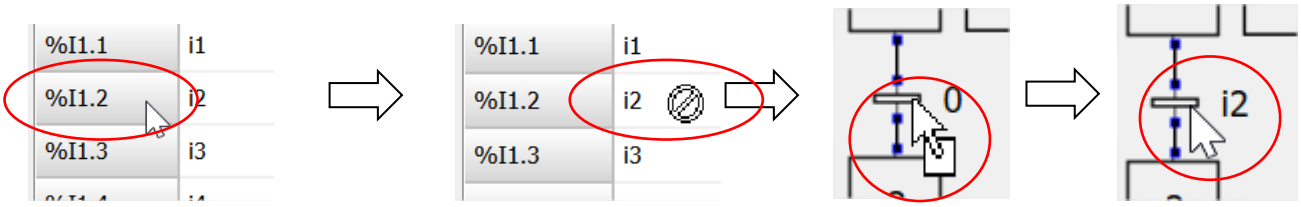
→ Case of an output variable dragged and dropped on a *step* :



If the output is placed on a step that already has one or more actions, it is added to the existing actions. Conditional actions cannot be declared in this way.

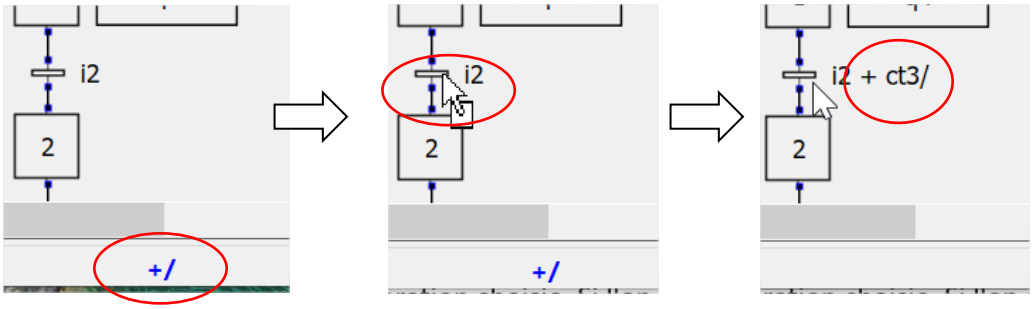
→ Case of any variable dragged and dropped on a *transition* :

Without prior action, a defined or undefined transition condition (non-empty, empty or with default value 0) will be replaced by the dropped variable :



If you wish to drop the complement of the chosen variable (i.e. NOT i2 or i2/), you must first press the "/" (Slash) key on the keyboard. This character ("/") will then appear in the status bar of the grafcet form. You can then drag and drop.

To *complete a transition condition* with a sequence of logical operations, you must first press the key corresponding to the desired operation ("+" for OR or "." for AND). The corresponding character ("+" or ".") will appear in the status bar of the grafcet form. The drag-and-drop variable will then be added to the transition condition with the chosen operation. If you wish to add the complement of a variable in the same way, you must first press the chosen operation ("+" or "."), then the "/" key, before dragging and dropping the variable :



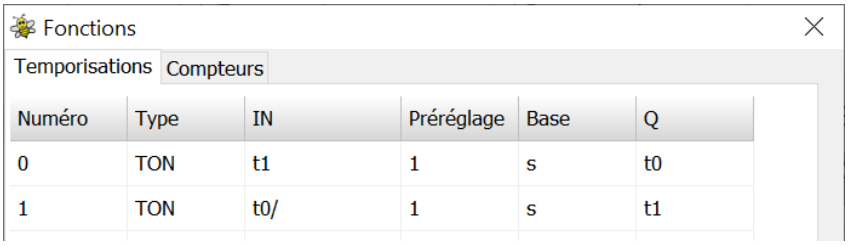
It is not possible to prioritize operations in this way.

8 Special functions

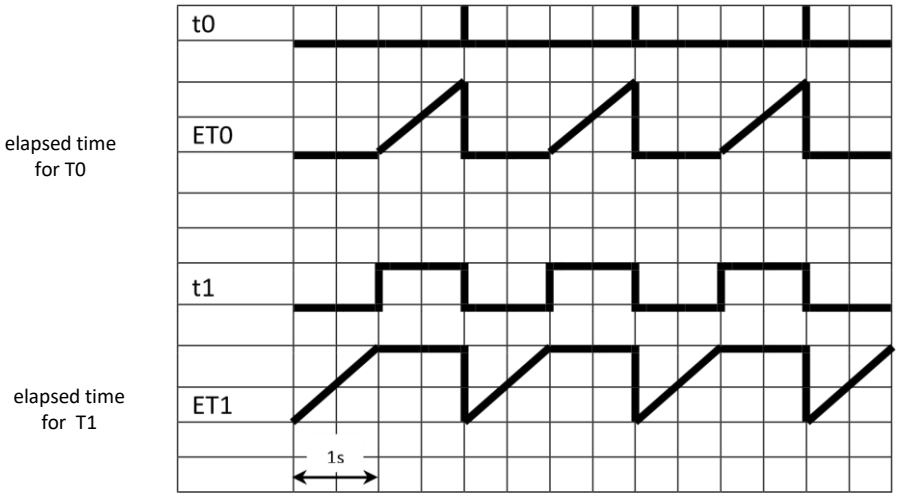
The following functions are not directly available in Apimou but it is possible to create equivalents.

8.1 Clock signal or flashing

To obtain a flashing bit, two Ton timers can be used:



The operation follows this chronogram :

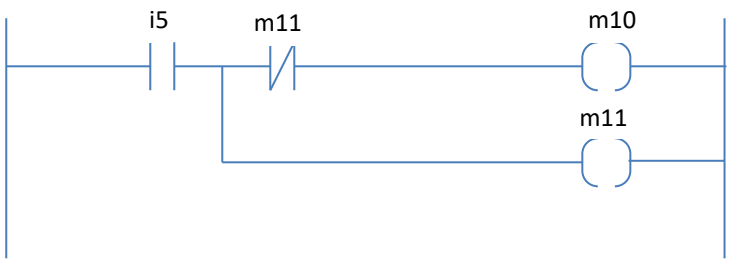


The frequency of the signal is varied by adjusting the preset time (PT) of the two timers. With a "real" PLC this type of signal is often available on a system bit.

8.2 Rising or falling edge

To create a rising edge trigger of a variable, e.g. i5, two internal bits must be used, e.g. m10 and m11.

The equivalent scheme in ladder language would be as follows:



The PLC scans the program from left to right and from top to bottom. When i5 goes to 1, with bit m11 set to 0, bit m10 goes to 1 (1st line). m11 immediately afterwards (in the same PLC cycle) goes to 1 since i5 is 1. In the next (PLC) cycle, i5 is still 1, m11 is 1, and m10 falls back to 0. This gives m10 = i5↑, so m10 can be used as a rising edge trigger for i5.

Apimou is not programmed in ladder but in equations, which gives:

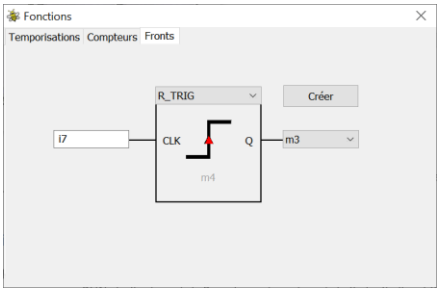
(i5↑ =) m10 = i5.m11/ and m11 = i5

For a falling edge trigger, we would have:

(i5↓ =) m10 = i5/.m11/ and m11 = i5/

N.B.: The bit that serves as the rising edge trigger (m10 for the example) must imperatively have a lower number than the one that serves as the intermediate variable (here m11), as Apimou calculates the internal bits in ascending order of their numbers.

To quickly create an edge trigger bit, according to the principle described above, one can use the assistant accessible from the menu View->Functions, Edges detections tab:



It is then sufficient to define the variable (or expression) whose edge you wish to detect, the type of edge (rising or falling) and the internal bit image of the edge. Pressing the Create button automatically generates the necessary equations.

The internal bit mnemonics accept the characters "↑" (Alt+24) and "↓" (Alt+25). If, for example, a rising edge of %I1.7 (default mnemonic "i7") is created with the internal bit %M3 (default mnemonic "m3"), the mnemonic of %M3 can be changed to "i7↑". N.B.: "↑" and "↓" are only characters, not functions. The creation of an edge trigger passes necessarily through the procedure detailed above.

8.3 Initialization bit

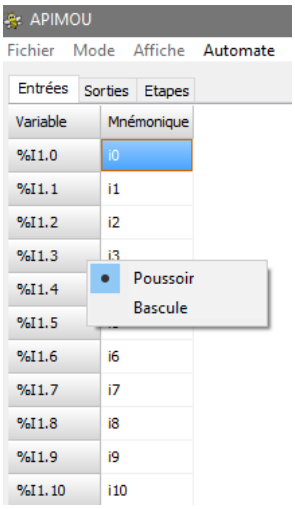
An initialization bit is a bit that is set to 1 on the first PLC cycle and 0 thereafter (i.e. a rising edge trigger for the PLC run). For real PLCs, it is a system bit. Since Apimou does not have one, we can make an equivalent in the following way:

Entrées | Sorties | Etapes | Bits internes

Variable	Mnémonique	Equation
%M0	init	1.m1/
%M1	m1	1

The explanation of how this works follows from what is written in the previous paragraph.

9 Operation in simulation mode



Menu Mode -> Simulation.

As the PLC is in RUN mode (i.e. running the program), the state of the inputs can be changed by clicking on the corresponding variables in the Inputs tab of the main form. The LEDs on the input card will light up.

For each input variable, it is possible to choose a push button operation (default mode) or a switch operation by right-clicking on the chosen variable.

The visualization of the state of the steps is done on the grafcet (active step(s) in green).

The status of the internal bits is shown in the corresponding table (mnemonic cell in green = bit set).

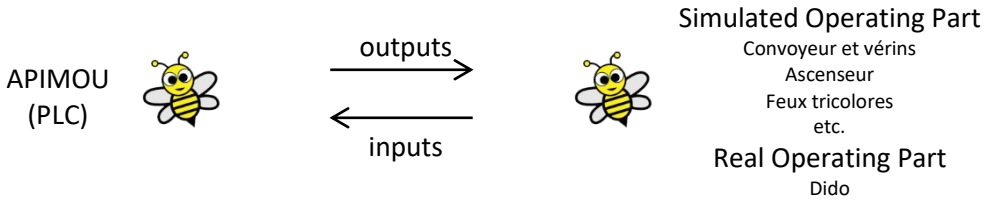
The status of the outputs is directly displayed by the LEDs on the output card.

10 Operation in connected Mode

Menu Mode -> Connected.

The operation of Apimou is the same as in simulation mode, when the PLC is in RUN mode, the state of the inputs/outputs can be viewed on the cards, the state of the internal bits on the table and the state of any step on the grafcet.

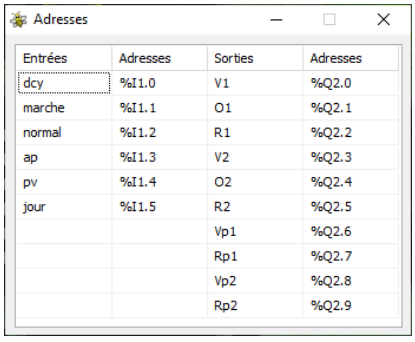
In this mode, Apimou interacts with a simulated or real Operating Part :



This requires that the operative part be connected to Apimou, which is done via the menu Apimou -> Connect.

It is also necessary to know the wiring diagram of the PLC (Apimou) for this operating part : to which inputs the sensors are connected, to which outputs the actuators are connected, i.e. an input/output table. This can be obtained from the menu View -> Adresses.

Example for *feux tricolores* :



Entrées	Adresses	Sorties	Adresses
dcy	%I1.0	V1	%Q2.0
marche	%I1.1	O1	%Q2.1
normal	%I1.2	R1	%Q2.2
ap	%I1.3	V2	%Q2.3
pv	%I1.4	O2	%Q2.4
jour	%I1.5	R2	%Q2.5
		Vp1	%Q2.6
		Rp1	%Q2.7
		Vp2	%Q2.8
		Rp2	%Q2.9

The push button ap (pedestrian call), for example, is connected to the fourth input of the PLC (Apimou), address %I1.3.

In Apimou, the default mnemonic for the %I1.3 input is i3. If we replace it with ap, we can then use ap in the program to refer to the %I1.3 input.

The "config_(OP name).api" files, once loaded into Apimou, avoid the tedious task of changing all the mnemonics to match the labels of the operating part.

For a real operating part (eight inputs/eight outputs maximum), Dido can be used to control an external I/O module on an RS-485 link in Modbus RTU (there is no need to understand what the RS-485 link or the Modbus RTU protocol are).

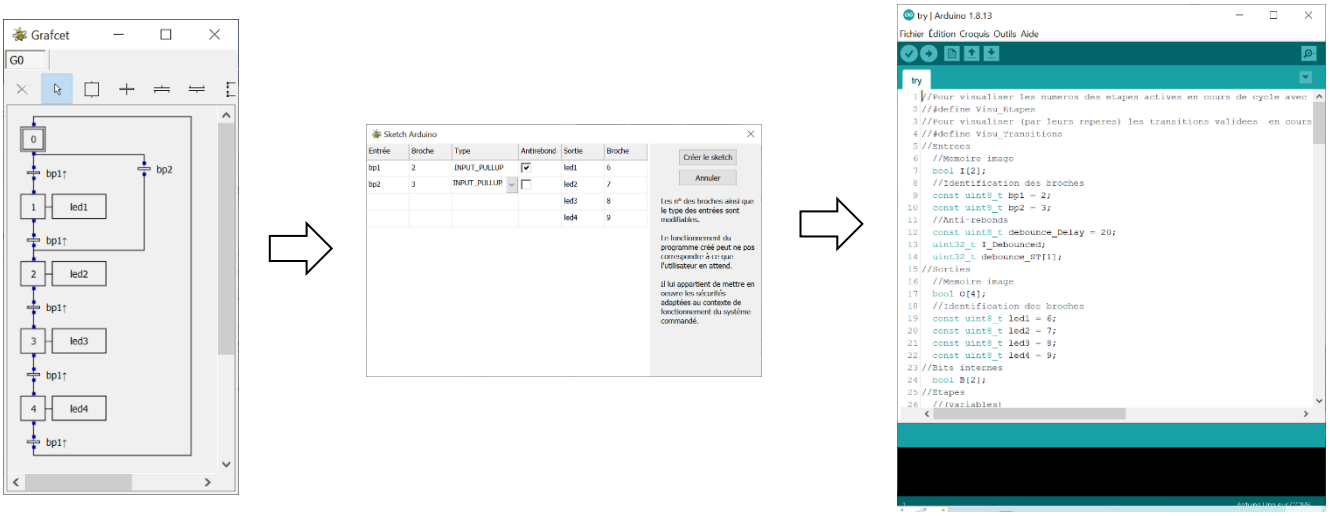
N.B. : The Operating Parts supplied with Apimou versions 1.x.x (including Dido) are not compatible with versions 2.x.x.

11 Arduino code generation

N.B. : This function only generates Arduino compatible C++ code. There is no communication between Apimou and Arduino.

The menu *File -> Create an Arduino sketch* allows you to transform the program into an Arduino sketch (file ".ino"). It is possible to modify the pins of the inputs/outputs as well as to choose the input type (INPUT_PULLUP, PULLUP PULLDOWN).

Each input can also be assigned a debounce function. This option can be useful for mechanical sensors whose signal is likely to bounce briefly on changes of state. If, for example, successive presses of a push-button are to be tested by a rising edge, or a counter is to be incremented in the same way, signal bounces can cause erratic operation. This problem can be remedied with a debounce mechanism. Debouncing is achieved here by means of a timer which freezes the input value for a set period of time as soon as the input changes state. The default time is 20 ms, but can be modified in the *Tools->Options* menu.



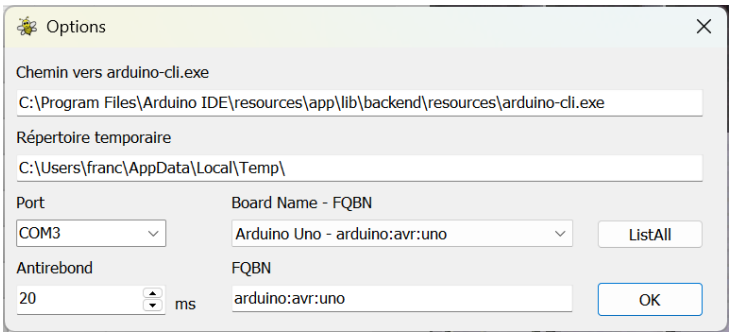
12 Direct upload to an Arduino board

A program created with Apimou can be directly transferred (uploaded) to an Arduino board, without having to go through the Arduino IDE. This feature requires the program **arduino-cli.exe** to be installed on the computer. It is already present on your computer if you have installed the **Arduino IDE version 2.x**. Otherwise, if you have an Arduino IDE version 1.x, you will need to install this program independently.

The only thing Apimou does here is to retrieve the information required for arduino-cli to compile and transfer the program, using the command :

arduino-cli compile --upload --verbose --no-color (sketch.ino) --port (port name) --fqbn (FQBN identifier)

Before you can carry out a first direct upload, you need to set the necessary parameters by clicking on the Tools->Options menu (these parameters will then be saved for later use in a file named Apimou.ini) :



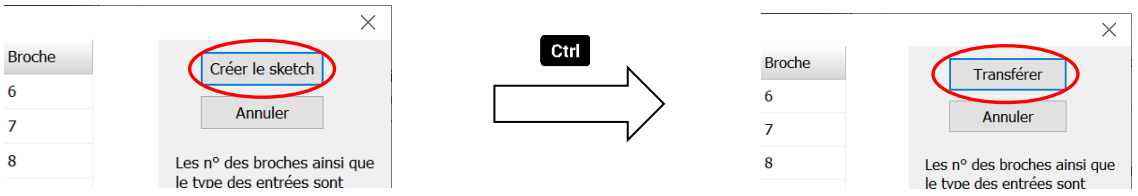
For users of the Arduino IDE 2.x, the path to arduino-cli.exe should in principle be filled in automatically. If arduino-cli.exe has been installed independently, or if Apimou cannot find it, it must be defined.

The default temporary folder is that of the user. It can be modified. Apimou will create a subfolder apimou_tmp in this folder, in which it will save a file apimou_tmp.ino. This is the file that will be compiled and uploaded by arduino-cli, and which will be deleted and rewritten with each new upload.

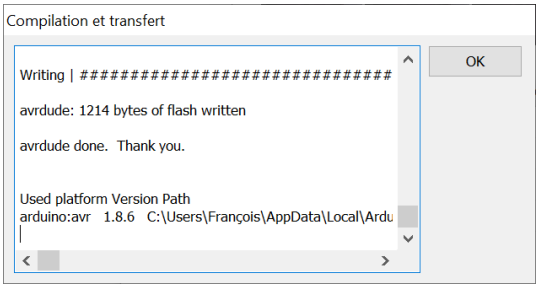
The port, to be selected from the drop-down list, must be the one to which the Arduino board is connected. If you have several USB devices connected, the port selected by default is not necessarily the right one. If the board is not connected, the port will not appear. You'll need to connect the card, then close and open the options window again.

The FQBN (Fully Qualified Board Name) is that of the board used. If the board is installed and the path to arduino-cli is valid, you can find the FQBN by clicking on the ListAll button and selecting the board from the drop-down list.

The procedure for uploading the program directly to the board is then via the File -> Create Arduino sketch menu, as explained in the previous paragraph. On the form that appears, by holding down the Ctrl key on the keyboard, the "Create sketch" button is renamed to "Transfer":



Clicking on this button starts the compilation and upload procedure. A window displays the results of the operation, provided by arduino-cli:

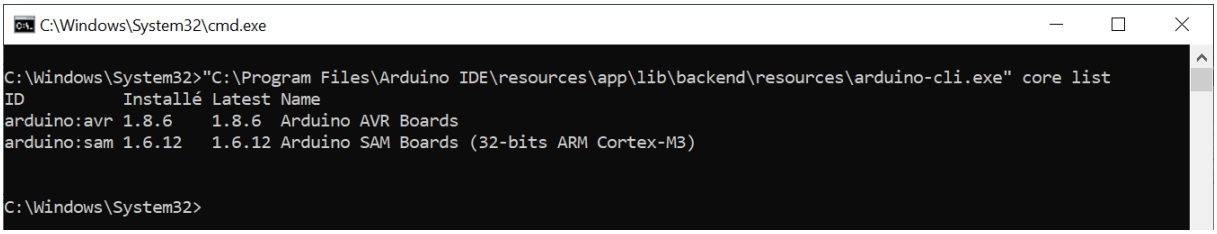


If the procedure fails, you may need to :

- Detect the connected board with the arduino-cli *board list* command
- List installed boards and obtain their FQBN with the arduino-cli *board listall* command.
- List installed board families with the arduino-cli *core list* command
- List board families that can be installed with the arduino-cli *core search* command
- Install a board family with the arduino-cli *core install* (...) command
- Etc. The official documentation can be found on github: https://github.com/arduino/arduino-cli

If necessary, to launch a command, you can for example :

- Copy/paste the path to arduino-cli into a text editor (notepad, for example)
  C:\Program Files\Arduino IDE\resources\(...)\arduino-cli.exe
- Surround it with quotation marks (to avoid misinterpretation due to spaces)
  "C:\Program Files\Arduino IDE\resources\(...)\arduino-cli.exe"
- Add the desired command
  "C:\Program Files\Arduino IDE\resources\(...)\arduino-cli.exe" core list
- Open the Windows command interpreter cmd.exe (type cmd.exe in the search field of the Start menu)
- Copy the line from the text editor and paste it into cmd.exe :



13 ESP32 code generation

13.1 Overview

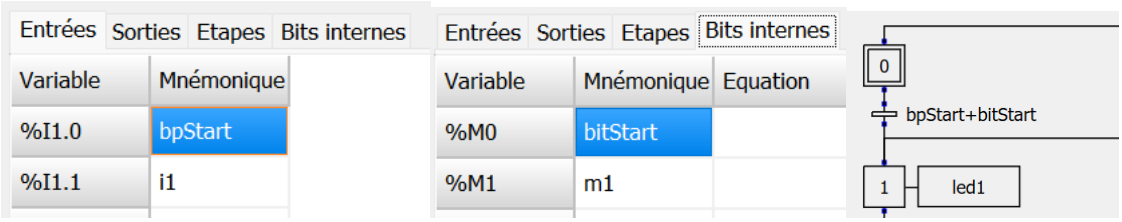
With an ESP32, it is possible to generate C++ code (.ino file) and a set of files (.html, .png) enabling the evolution of the grafcet(s) making up the program to be viewed with any web browser. Communication is via Wifi, so any terminal (PC, smartphone, tablet) connected to the same network as the ESP32 can be used.

In addition to this visualization, you may also be able to access settings via a dedicated web page:

- ✓ modification of bits, called *HMI bits* (if used),
- ✓ modification of timers preset times (if used),
- ✓ modification of counters preset values (if used).

HMI bits are, in Apimou, internal bits which are *not defined by an equation* and which *appear in any equation* (output or timer or counter or forcing or transition).

Example :



bpStart is an input (a push-button, for example), physically connected to an ESP32 pin. bitStart is an internal bit which has no equation. bitStart is involved in the receptivity of the step 0 -> step 1 transition (equation “bpStart + bitStart”, which means that the cycle starts if the bpStart push-button is pressed or if bitStart changes to 1). When translating the program into a sketch, bitStart will be considered as an HMI bit, and it will be possible to force it from the settings page.

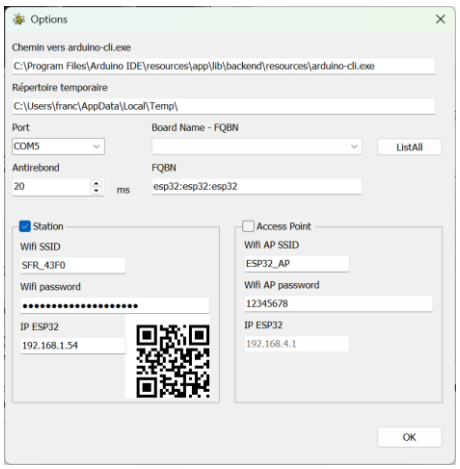
To *validate* a setting, click on the corresponding validation button. This setting remains valid until the ESP32 restarts. To *save* all validated settings (including HMI bits), click on the save button. The backup is saved in Flash memory (NVS) and loaded each time the device is restarted.

13.2 Prerequisites :

- IDE Arduino 2.x
- ESP32 card installed
- LittleFS Uploader plugin (see https://randomnerdtutorials.com/arduino-ide-2-install-esp32-littlefs/) installed
- Async TCP library by ESP32ASync installed
- ESP Async WebServer library by ESP32Async installed

13.3 Station and Access Point modes

Before generating the sketch, you must select the WiFi mode to be used in the Tools -> Options menu: **Station** and/or **Access Point**.



In **Station** mode, esp32 connects to an existing Wifi network. To do this, you need to know the network name (SSID) and the password that authorizes access. The network router (the "box") automatically assigns an IP address to the ESP32. If the signal is strong enough, Station mode is the most efficient

In **Access Point** mode, the esp32 generates its own Wifi network and acts as a router. The network name (SSID) and password are defined. The IP address of the esp32 is set to 192.168.4.1. Access Point mode is limited to 4 clients, in practice 1 or 2 for a smooth refresh rate and limiting the frequency to 100 to 200 ms.

It is possible to choose both modes simultaneously, **Station + Access Point**. In this case, Access Point mode can be used as a back-up, in case the existing Wifi network is down or out of range. This choice remains less efficient than the Station-only mode.



For a device (PC, smartphone, tablet) to access the ESP32 server in **Access Point** mode, it must first connect to the Wi-Fi network created by the ESP32. If this device was connected to the server (box) of the existing local Wi-Fi network, it will lose this connection and thus, among other things, access to the Internet.

The **QR codes** that appear in the Options sheet refer directly to the ESP32's IP addresses (LAN address and/or Access Point address), provided the device is connected to the corresponding Wi-Fi network

13.4 Procedure

For **Station** mode, it's *best* to define the Wifi network name (Wifi SSID) and Wifi password beforehand. These two parameters can also be set later.

For **Access Point mode**, you must first define the Wifi AP SSID and Wifi AP password. Default values are proposed. The password can be empty (in which case the network will be open).

After selecting ***File -> Create sketch -> ESP32***, the procedure is identical to that described in the Arduino code generation section. Unlike an Arduino board, here you can configure an input as INPUT_PULLDOWN.

After clicking on ***Create sketch***, Apimou will create an *.ino* file (the sketch) and a sub-folder containing the necessary *.html* and *.png* files. It's important that the *.ino* file is created in a folder with the same name (for example, below, projet.ino is in a folder named projet).

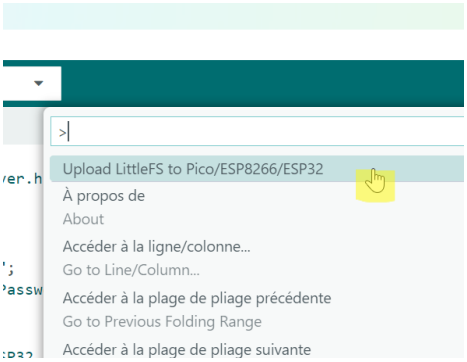
- projet\
  - projet.ino
  - (projet.api)
  - data\
    - home.html
    - vue.html
    - grafcet0.html
    - grafcet0.png
    - grafcet1.html
    - grafcet1.png
    - (…)

Next, open the *.ino* file created in the **Arduino IDE**.

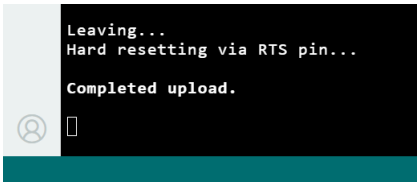
If the SSID and password for the WiFi network in Station mode have not already been entered in Apimou's Options menu, they must be entered in the Arduino IDE on lines 6 and 7 of the sketch (keeping the quotation marks).



After checking that the correct board has been selected, that the COM port is correct and that the **serial monitor is closed**, press **[Ctrl] + [Shift] + [P]** to open the commands menu and select *Upload LittleFS to Pico/ESP8266/ESP32*.

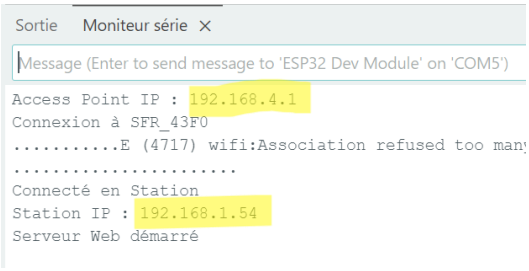


The LittleFS Uploader plugin will then upload all the files in the project's \data folder to the ESP32. After a few seconds, the message *Completed upload* should be displayed.

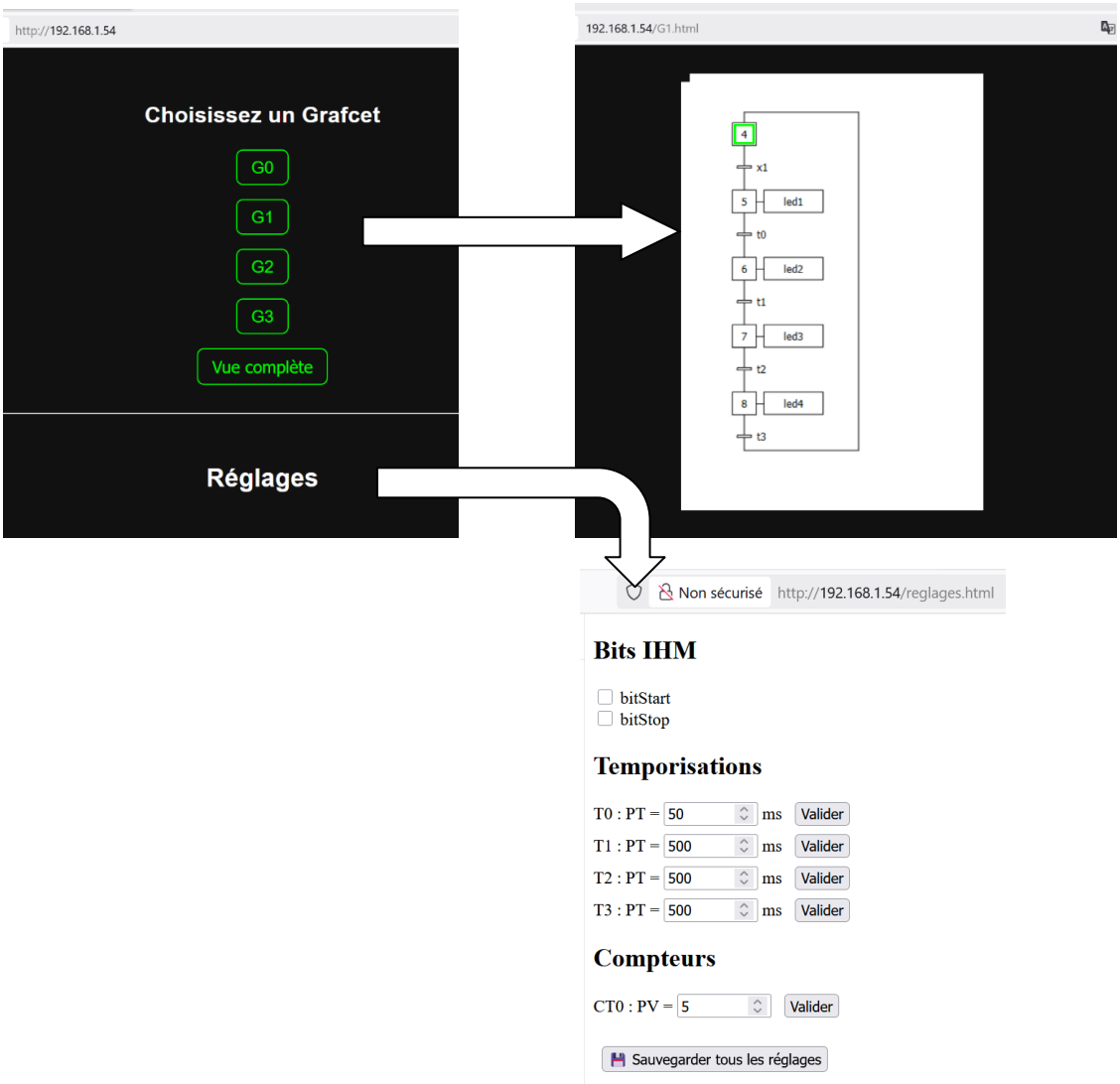


Next, **open the serial monitor** at 115200 baud and **upload the sketch** (Sketch -> Upload or [Ctrl]+[U]).

At the end of the upload, the serial monitor should display a message confirming connection to the WiFi network and the IP address of the ESP32 for Station mode and/or Access Point mode.

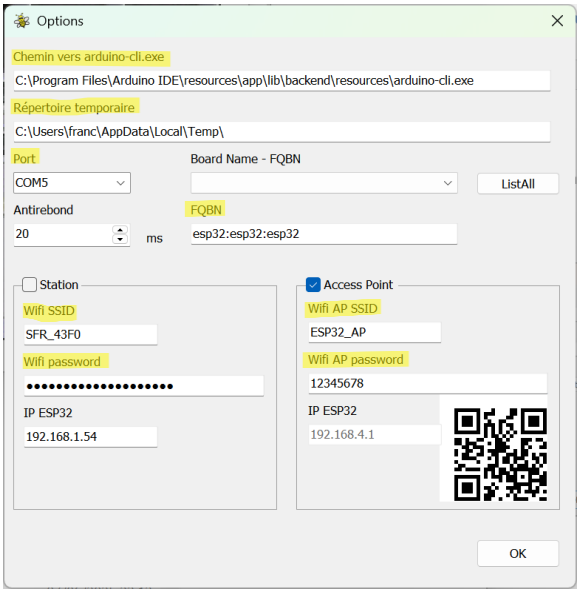


Then simply connect to the desired network (existing local network in Station mode, or network created by the ESP32 in Access Point mode), copy one of these addresses and paste it into the address bar of a web browser (or scan one of the QR codes) to access the home page of the visualization interface. From here, you can select the desired grafcet (there may be only one), and watch it animate in real time. You can also open the Settings page and modify HMI bits or preset timers and counters.



14 Direct upload to an ESP32

Before implementing this procedure, it is necessary to have generated and uploaded a program at least once with the Arduino IDE, as explained in the previous paragraph, to ensure that everything is in place and working properly.

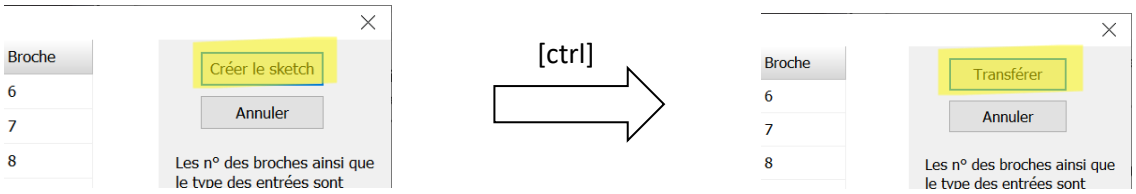


Then, in the options window (*Tools->Options*), enter all the necessary parameters: **path to arduino-cli**, **temporary directory**, **port**, **FQBN**, **SSID** and Wifi **password** for the selected mode(s) (Station and/or Access Point).

The first four parameters are explained in detail in the section Direct upload to an Arduino board.

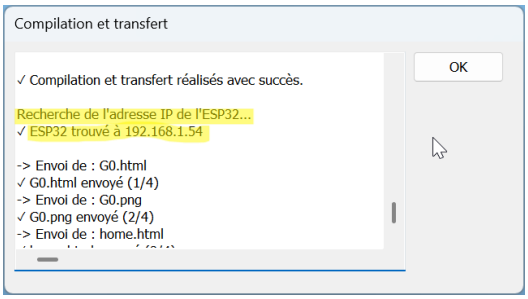
The IP address of the ESP32 in **Station** mode should be found automatically, so the “IP ESP32” field can be left blank. If a first transfer fails because this address is not found, you may need to fill it in.

After selecting the ***File -> Create sketch -> ESP32*** menu and configuring the inputs/outputs, simply press and hold the [ctrl] key on the keyboard, and the “Create sketch” button will be renamed “Transfer”:

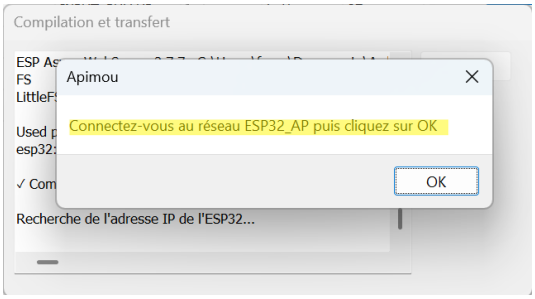


Clicking on this button starts the compilation and transfer procedure. First, arduino-cli compiles and transfers the sketch. Secondly, Apimou sends the necessary .png and .html files via WiFi (the LittleFS Uploader plugin is not used here).

When **Station** mode or **Station + Access Point** mode is selected, html and png files are transferred in Station mode, so the PC must be connected to the existing local network. Once the sketch has been compiled and transferred, the IP address of the ESP 32 is searched, and the html and png files are transferred.



When **Access Point** only mode is selected, the procedure stops after the sketch has been transferred. A message then appears asking you to connect to the network created by esp32 (by default ESP32_AP with password 12345678), **then** click on OK so that the html and png files can be transferred.



At the end of the procedure, the ESP32 IP address is copied to the clipboard. Simply paste it ([ctrl]+[V]) into the browser's address bar, or return to the Options menu to scan any of the QR codes

15 File management

The menu *File* -> *New* deletes everything (mnemonics, equations, grafcets) without prior warning.

To save a file (a program), select the menu *File* -> *Save as…*. The extension of the Apimou files is ".api" (extension added by default).

To open a file, use the menu *File* -> *Open*. It is also possible, if Apimou is not running, to open a file (and indeed Apimou) by double clicking from the folder in which it is located. The operating system will then ask for the executable to be associated with the ".api" extension.