Blocaine

(The open-source solution for automation)

Table of Contents

- 1 - Generality

Blocaïne is a software platform designed for the automation of industrial processes. It comprises two main components:

- A **block editor,** which allows you to create programs by assembling blocks functional.
- An **Executor** **,** which executes programs created with **the Block Editor,** thus controlling industrial equipment via remote inputs/outputs.

The Block Editor is installed on a "Development PC" and communicates via Ethernet with **the Executor,** deployed on another computer called the "Target Machine".

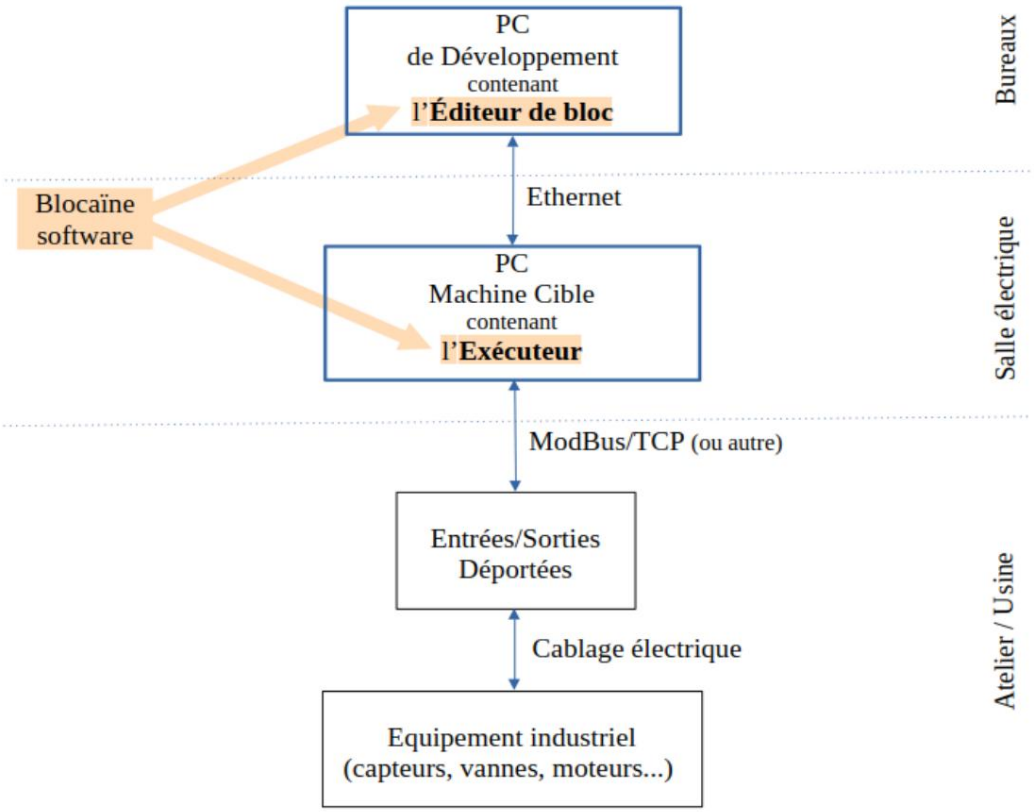


Figure 1 : Architecture générale

The Ethernet connection allows programs created with **the Block Editor** to be transferred to **the Executor;** it also enables real-time visualization within **the Block Editor.**
variables of the programs currently being executed by **the Executor.**

- 1.1 - Features

- “ Hot **Swap” :** A new version of a program can be loaded while the old version is running.
- Graphical programming in the form of linked functional blocks
- Each variable natively has a validity bit, which propagates from block to block.
- Dynamic typing of variables.
- The order in which the blocks are executed is managed automatically
- The layout of the blocks is done automatically
- Blocaïne is free and open-source software

- 1.2 - Project progress

The Blocaïne project is at the functional prototype stage.

It is available for download for evaluation on "github.com"; the installation procedure is described in the document " evaluation.pdf" »

If you would like to participate in the project, you can contact me by email: blocaine@barachet.com

- 1.3 - Precautions for use, responsibilities

The use of Blocaïne is at the user's own risk.

- 2 - Principles

- 2.1 - Everything is a block

Each block consists of input(s), output(s), and the code to evaluate the outputs based on the inputs.

- For blocks in the base library (called **System blocks) ,** it is a function written in Python evaluates outputs based on inputs
- For user-created blocks (user **blocks) ,** it's a chain of **system** or **user** blocks . which allows us to evaluate the outputs based on the inputs

An executable program is simply a **user block.** A subroutine also takes the form of a **user block.** The inputs/outputs of blocks are themselves blocks (<inputs>, <output>). The <input> block has an output defined by an <output> block, while the <output> block has an output defined by an <input> block. In short, everything is a block.

- 2.2 - Hot Swap

The Executor has two versions for each program (user **block) : Shift A** and **Shift** B. The new version of a block is always downloaded into the **Shift** that is not currently running. When **a HotSwap** is initiated, the current values of the stored variables from the **Shift** that is currently running are transferred to the other **Shift,** which then takes over.

- 2.3 - Method of execution

For each running **user** block, **the Executor** periodically evaluates all the <output> blocks associated with a periodic task. Each input of an <output> block is evaluated by **the Executor** by first executing the previous block, and so on.

This approach eliminates the need to explicitly define the execution order of the blocks.

- 2.4 - Variable typing

Since Blocaïne is based on the Python language, the variables (input and output) used by the blocks are dynamically typed; their type can be anything, including complex structures, as long as it is a combination of types recognized by Python (string, integer, float, list, dictionary, etc.).

- 2.5 - Validity bit

A validity bit associated with each block output is automatically evaluated at each execution. depending on the validity of the inputs and the possibility or impossibility of evaluating the output.

- 2.6 - Default values

Each block entry, whether it is a **system** or **user block,** has a default value that can be modified at instantiation.

- 2.7 - Remote entrances/exits

The "remote I/O" is managed by dedicated **system** blocks .

Currently, only the Modbus/TCP protocol is supported via these blocks:

- modbus_conn: TCP connection with the remote input/output module
- modbus_read: read from a Modbus slave
- modbus_write: writes to a Modbus slave

- 3 - The Block Editor

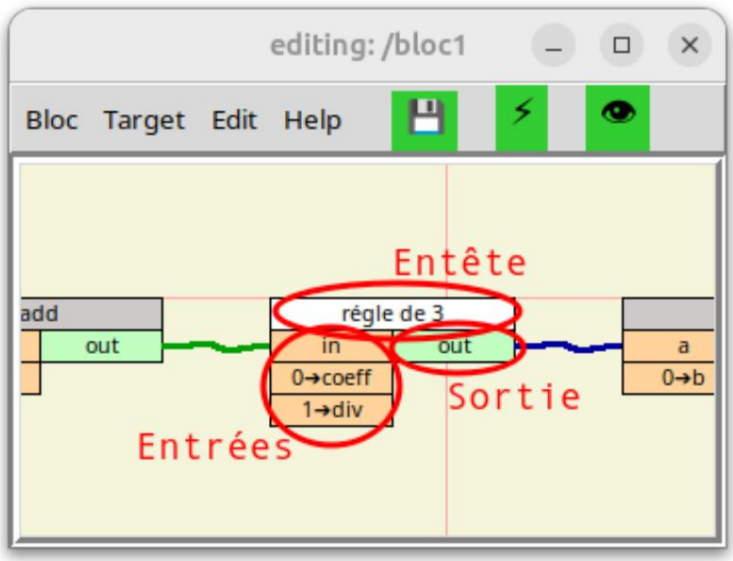
- 3.1 - Generalities

The Block Editor allows you to:

- The creation and modification of programs in the form of **user** blocks containing blocks functions chained together.
- Compiling, downloading and launching the execution of the current **user** block publishing in the "Target Machine".
- Dynamic visualization and forcing of input/output for **user** blocks and sub-blocks running.

Each block consists of:

- A header: containing the block name
- Entry(ies): located on the left below the header, containing the entry name
- Output(s): located on the right under the heading "to", containing the name of the output



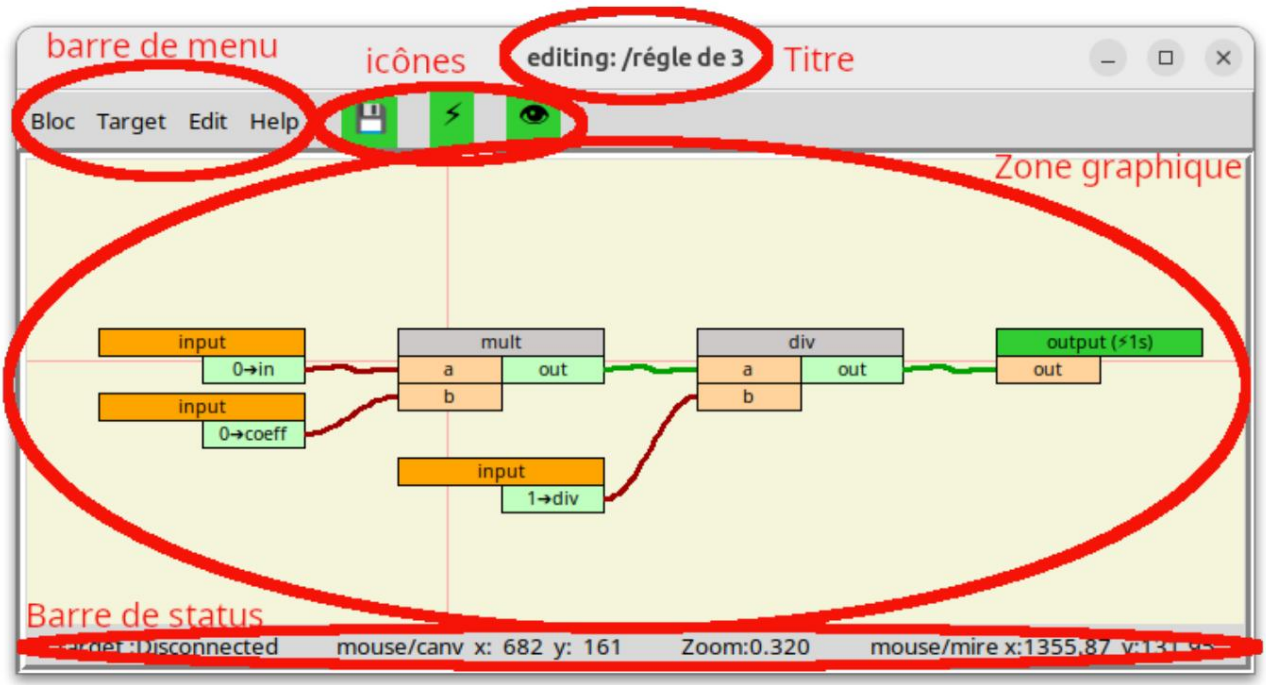
Blocks can be chained together by links, note that an output can only be linked to one input and an input can only be linked to one output.

Each block corresponds to a file named: " *block name.block* ". **System** block files are in the /blocks/system directory, while the **user** block files are in /blocks/user.

- 3.2 - Progress of the Graphical Interface Project

The graphical interface of **the "Blocaïne" block editor** consists of one or more windows, each containing the following elements:

1. A title bar (of the window)
2. A menu bar
3. Three icons: , ÿ ÿ
4. A graphical area, where you place and link blocks together to create a program (user **blocks).**
5. A status bar:



Graphical interface of the block editor (Blocaine)

- 3.2.1 - Title bar

At the top of each window is a title bar which contains:

- Current mode: **Editing** or **Monitoring**
- the name of the block being edited or debugged

Note that if the block was opened as a subblock, sub-subblock, etc., the path used to open it will appear before the block name. This is useful in **Monitoring** (debugging) mode to determine which instance of the block the variables displayed in this window belong to.

- 3.2.2 - Menu bar

The menu bar located in the top left corner below the title bar contains the following four menus:

- **Block :** is equivalent to the "File" menu in most software.
- **Target :** manages the interface with the "Target Machine" (i.e., with **the Executor).**
- **Edit :** manages the presentation.
- **Help :** displays documentation.

The " **Block** " **menu contains the following submenu:**

- **Open :** allows you to open a file containing a block.
- **Save [F3] :** allows you to save the block being edited to a file whose name corresponds to the name of the current block.
- **Save as [F4] :** allows you to save the block being edited to a file whose name will be entered by the user.
- **Update [F5] :** Updates all external interfaces of the **user** blocks used as a subroutine within the **user** block being edited.
- **Change Properties :** This allows you to modify the fields associated with the header of the **user** block being edited, such as "the author of the block", "keywords"...but also its "name".
- **Open new windows :** opens a new window, which allows you to open another **user block.**
- **Quit [Escape] :** quits the current window, without saving the **user** block being edited.

The " **Target** " **menu contains the following submenu:**

- **Disconnect from Target** or **Connect to Target:** to connect to or disconnect from the "Target Machine" (i.e., **the Executor).**
- **Check *<block_name> :*** to check if the block being edited is valid and compilable.

- **Check & Transfer *<block_name> :*** checks that the block being edited is valid, compiles it, then transfers the result in the "Target machine" to the **Shift** that is not "running".

- **Check, Transfer & Execute *<block_name> :*** verifies that the block being edited is validates, compiles, transfers the result in the "Target Machine" to a **Shift** that is not "running", then **the Executor** launches its execution: in the case where the block is already running on the other **Shift** a hot swap will be initiated, otherwise a simple start (Run) will be initiated.

- **Monitoring (Start/Stop) [Space] :** allows switching between edit mode and debug mode.

The " **Edit** " **menu contains the following submenu:**

- **Auto layout [F7]:** This command performs an automatic layout.
- **Status bar (show/hide):** This command allows you to show or hide the "status bar". status ».

The " **Help** " **menu contains the following submenu:**

- **Versions :** shows the version of **the Block Editor** and the block structure.
- **Mouse usage :** provides instructions on how to use the mouse:
  - ÿ [Left click] to select, to use menu
  - ÿ [Left held down] to scroll within window, to move bloc, to link I/O bloc
  - ÿ [Left double-click] to open **user** bloc
  - ÿ [Right button] to open the context menu
  - ÿ [wheel button] to monitor the target variables (on/off)
  - ÿ [wheel rotation] to zoom (in/out)
- **Key usage :** provides instructions on how to use the keyboard:
  - ÿ [F1] Open the documentation for the bloc whose header is located under the cursor
  - ÿ [F3] to save curant bloc
  - ÿ [F4] to save curant bloc as
  - ÿ [F5] to update curant bloc
  - ÿ [F7] to layout curant bloc
  - ÿ [Delete] Delete the bloc whose header is located under the cursor
  - ÿ [Space] to monitor the target variables (on/off)
  - ÿ [Escape] to close window

• **General Documentation :** opens this document (in HTML format) in the browser default web browser.

- 3.2.3 - Icons

These icons provide quick access to the most frequently used features:

- : saving the current block.
- "ÿ: Checks the consistency of the current block, converts it (compiles it) into a format understandable by **the Executor,** transfers it to the Target Machine, and then asks **the Executor :**

  1. Its startup (Run), if the current block is not already running in the Target machine.
  2. Hot-swap it, if the current block is already running in the Target machine.

- " ÿ : switching between Editing and Monitoring modes.

- 3.2.4 - Graphics Area

This area allows you to build a program (user **block)** by placing **system** or **user** blocks in it and linking them together.

- 3.2.4.a - Navigation

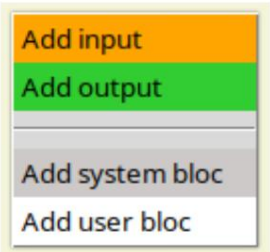
To navigate this area, you need to use a mouse:

- **Rotating the dial :** allows you to zoom in/out.
- **Moving the mouse while holding down the left button :** allows you to move the visible area (scrolls)
- **Right click :** brings up the context menu.
- **Double-click on the header of a user block :** opens the pointed-to **user** block in a new window.
- **Double left click on an input or output of a user block :** opens the **user** block of the pointed I/O in a new window and positions the input or output in the center of the graphics area.
- **Left click :** allows you to select a field in a menu.
- **Left click and drag :** creates a link (between an input and an output) or to move a block.
- **Click on the scroll wheel button :** toggles the real-time display of values variables located in the target (Monitoring On/Off)

- 3.2.4.b - Context menus

3.2.4.b.1 Default Menu

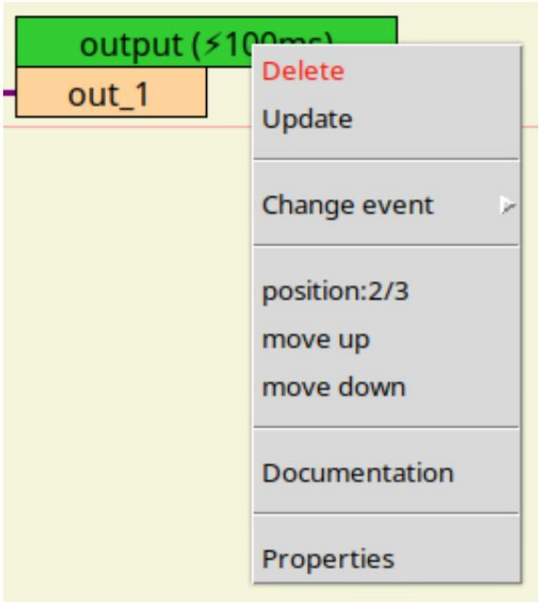
Right-clicking on a blank area brings up the following context menu:



This menu allows you to add a block.

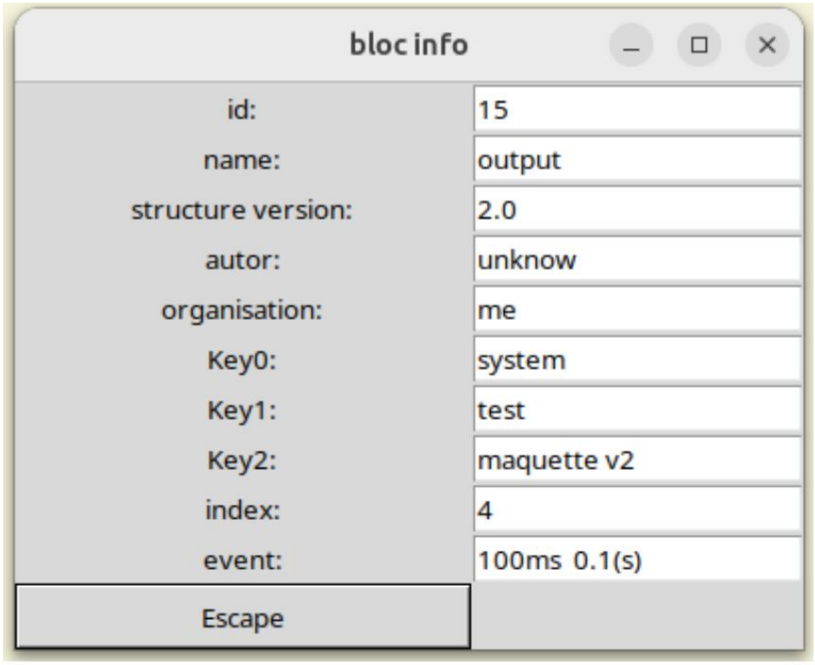
3.2.4.b.2 Menu « bloc »

Right-clicking on a block header brings up the following context menu:



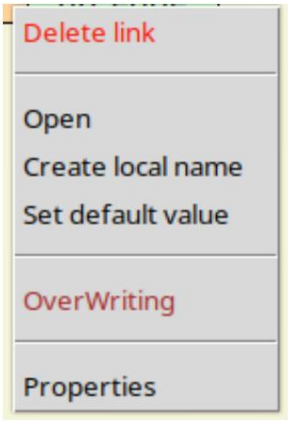
1. **Delete :** allows the deletion of the block
2. **Update :** updates the block's external interface, from the block's file.
3. **Change event :** allows the block to be associated with a periodic task so that it can be executed. This field is only available for <output> blocks
4. **Position: 2/3, move up, move down :** allows you to change the order of inputs or outputs in the external interface of the **user** block being edited. This field is only available for <input> or <output> blocks.
5. **Documentation :** Opens the block's help file in the default web browser. This field is only available if a help file named " *block_block type_block* name.html " is present in the "/ Documentation" directory.
    1. *Block type :* can be; **user** or **system**
    2. *Block name :* corresponding to the name defined in the block header

6. Properties: display the block properties like this.

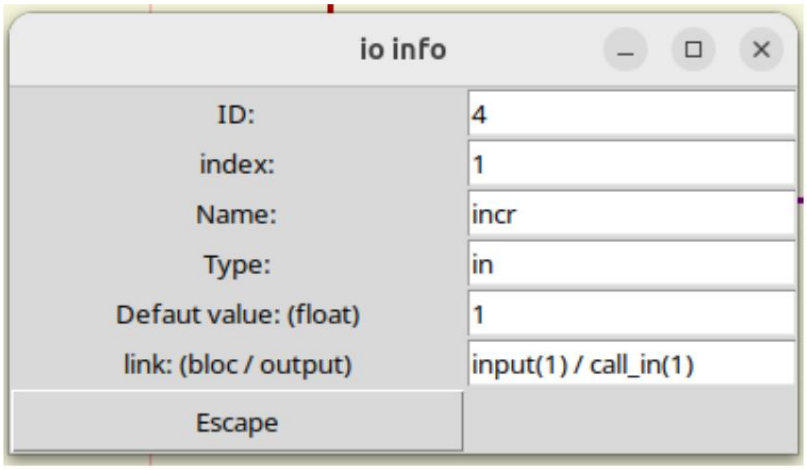


3.2.4.b.3 "Entry" Menu

Right-clicking on a block header brings up the following context menu:

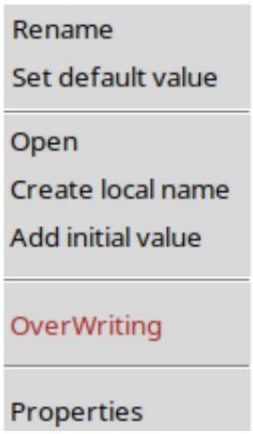


1. **Delete link :** allows the removal of the link with the previous block. This field is only available if there is a link between the input and output of another block.
2. **Open:** allows opening a new window that edits the **user** block of the input, and positions the input in the center of the graphics area.
3. **Create locale name :** allows you to name the entry locally; this name is associated with the external interface of the instance.
4. **Create locale comment :** allows you to associate a locale comment with the entry. The comment is associated with the instance's external interface.
5. **Set default value :** allows you to give a local default value to the input; this value is associated with the external interface of the instance.
6. **OverWriting :** allows you to overwrite the value of the input.
7. **Properties :** display the properties of the entry like this.

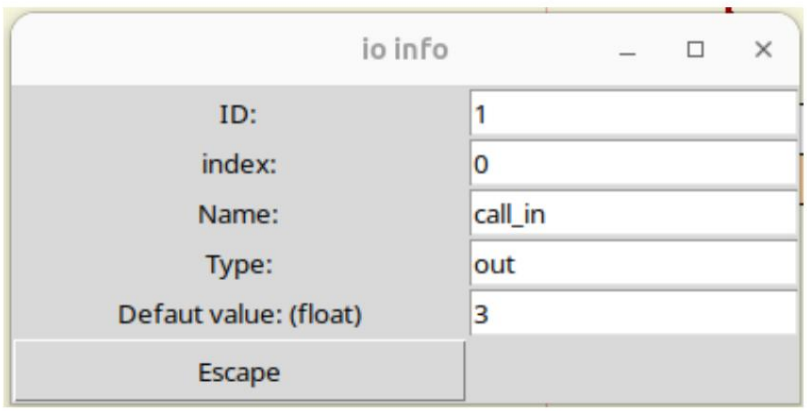


3.2.4.b.4 Menu "Sort"

Right-clicking on an output from a block brings up the following context menu:

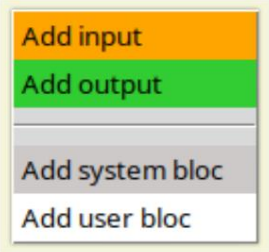


1. **Rename:** Allows you to rename the output; this new name will appear in the block's external interface. This field is only available for <input> blocks.
2. **Set default value:** allows you to define the default value of the input linked to the instance of the block. This field is only available for <input> blocks.
3. **Open:** opens a new window that edits the **user** input block and positions the output in the center of the graphics area. This field is only available for **user blocks.**
4. **Create locale name :** allows you to name the entry locally; this name is associated with the external interface of the instance.
5. **Create locale comment :** allows you to associate a locale comment with the input; this comment is associated with the instance's external interface.
6. **Add initial value :** allows you to assign an initial value to a "stored" output. This field is only available for all blocks whose output is "memorized" from one cycle to the next, e.g. <memory>, <previous>, <edge>.
7. **OverWriting :** allows you to overwrite the output value.
8. **Properties :** display the output properties like this.



- 3.2.4.c - Insert as a block

To insert a block, right-click on an empty area to bring up the following context menu:

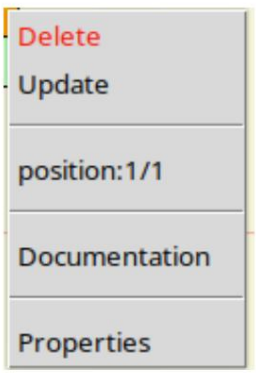


You then need to choose the type of block you want:

- input: to create a new entry in the **user** block being edited.
- output: to create a new output in the **user** block currently being edited.
- system: to insert a block that is part of the standard library.
- user: to insert a previously created **user** block.

- 3.2.4.d - Delete in one block

To delete a block, you can right-click on the header of the block you want to delete, which will bring up the following context menu:



then select Detete.

You can also position the cursor on the header of the block to be deleted, then press the "Delete" key.

- 3.2.4.e - Create a link

To link the output of one block to the input of another, click and hold the left mouse button on the output of the first block, then drag (while still holding the left mouse button) to the input of the second block, and then release the left mouse button. Links can also be created from either input to output or output to input, but an input can only have one source (i.e., originate from only one output).

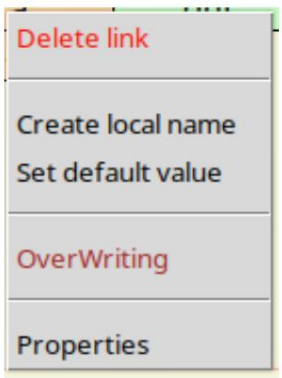
- 3.2.4.f - Remove a link

There are two ways to remove a link:

Either right-click on the link to bring up the following context menu:

then select Delete link

Either right-click on the entry located at the end of the link to bring up the following context menu:



then select Delete link

- 3.2.5 - Status Bar

She states:

- the status of the connection with the Target Machine.
- the position of the mouse relative to the graphic area (canvas).
- the current zoom coefficient.
- the position of the mouse relative to the target.

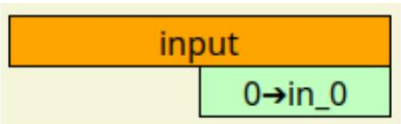
- 3.3 - Inputs/Outputs

The interface of a **user** block consists of inputs and outputs; the latter are optional but essential for interconnection with other blocks.

- Inputs are collected using **system** <input> blocks
- Outputs are delivered using **system** <output> blocks

- 3.3.1 - bloc <input>

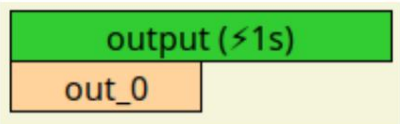
Here is the graphical interface of the **system** block <input>



Its function is to define an input for a **user block.** The <input> block has no input, but possesses
An output named "in_0" is the variable (of any type) that will be entered into the **user block.** Its default value is 0 (int), and it can be adjusted for each instance of an <input> block. This output can be renamed; the new name will appear in the **user** block's external interface as an input.

- 3.3.2 - bloc <output>

Here is the graphical interface of the **system** block <output>



Its function is to define an output for a **user block.** The `<output>` block itself has no output, but it does have an input named "out_0". This is the variable (of any type) that will output the **user block.** It has no default value, but a default value can be added for each instance of an `<output>` block. This input can be renamed; the new name will appear in the **user** block's external interface as the output.

- 3.3.3 - External interface of a user block

Here is a **user** block named "rule of 3" which contains three <input> blocks and one <output> block.

- The output of the first <input> block has been renamed to "in"
- The output of the second <input> block has been renamed to "coeff"
- The output of the third <input> block has been renamed to "div"
- The <output> block entry has been renamed to "out"

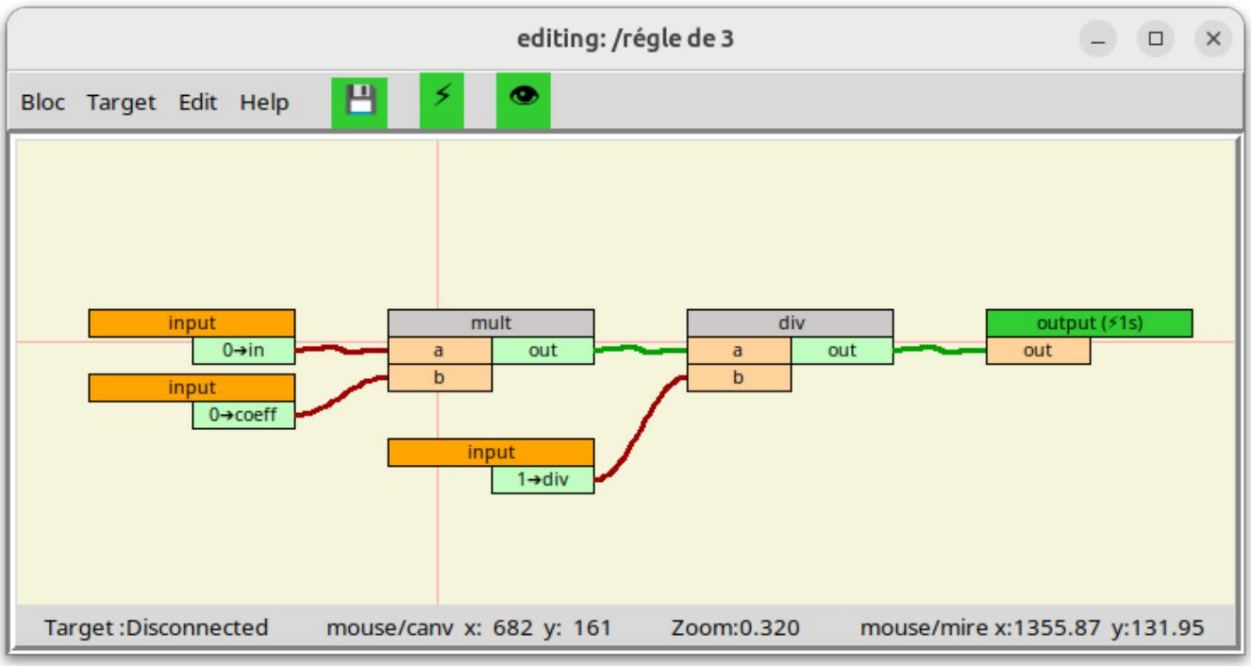
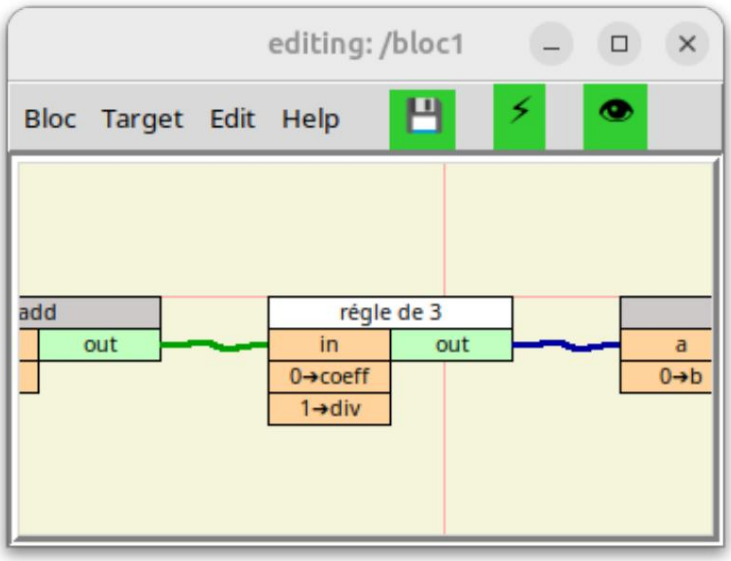


Figure: **user** block <rule of 3>

Here is the external interface of the "rule of 3" block, here a **user** block named <block1> refers to the "rule of 3" block, the 3 inputs and the output correspond to the <input>/<output> defined in the "rule of 3" block and are indeed present in its external interface.



Figure; external interface of the **user** block <rule of 3>

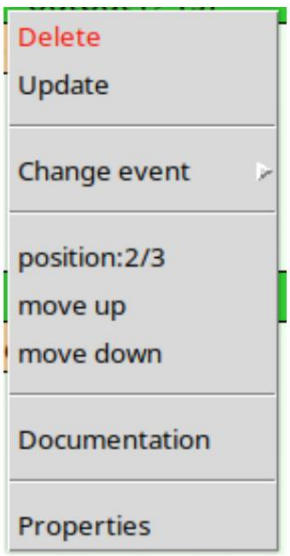
The order in which inputs and outputs appear in the external interface of a **user** block can be modified as follows:

1. Open the **user** block whose input and/or output order you want to modify.

   external interface, by double-clicking on the header of the block to be modified, if it is present in the graphic area, or by using the menu bar (block/open) or (block/open new window).

2. Right-click on the header of the <input> or <output> block, depending on whether you want to change the order of the inputs or outputs. This will display the following context menu:

3. **Position: 2/3** indicates the position in the external interface, of the input or output which you can then select you have selected . **move up** or **move down.** to shift it one position up or down.



- 3.4 - Mode : Editing / Monitoring

Switching between "edit" mode and "dynamic view" mode, and vice versa, is done either by clicking on the "eye" icon ( ), or by pressing the [Space] key; of course, the block being edited must be in progress to be able to switch to "dynamic view" mode.

- 3.4.1 - Mode: Editing

This mode is dedicated to creating and modifying **user blocks.**

Here is an example of a **user** block that calculates the average of 2 numbers:

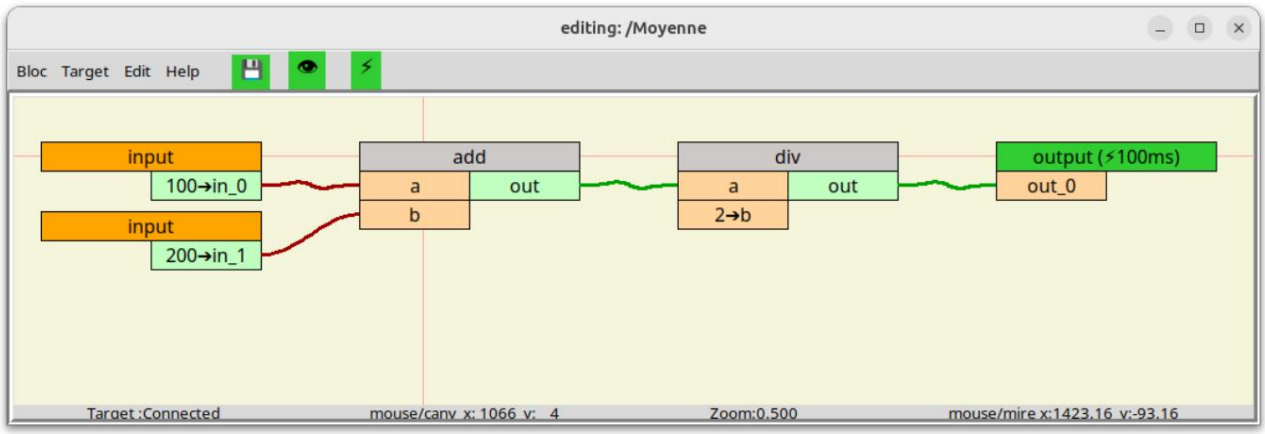


Figure 2: Block in edit mode

- 3.4.2 - Mode: Dynamic visualization (Monitoring)

This mode is dedicated to debugging; it allows real-time visualization of the values of variables in a **user** block or sub-block .

The validity of each variable is represented by the background color:

- Green = valid
- Red = invalid
- yellow = enforced and valid
- Black = Forced and invalid

Here is an example of the "average" block (in dynamic visualization):

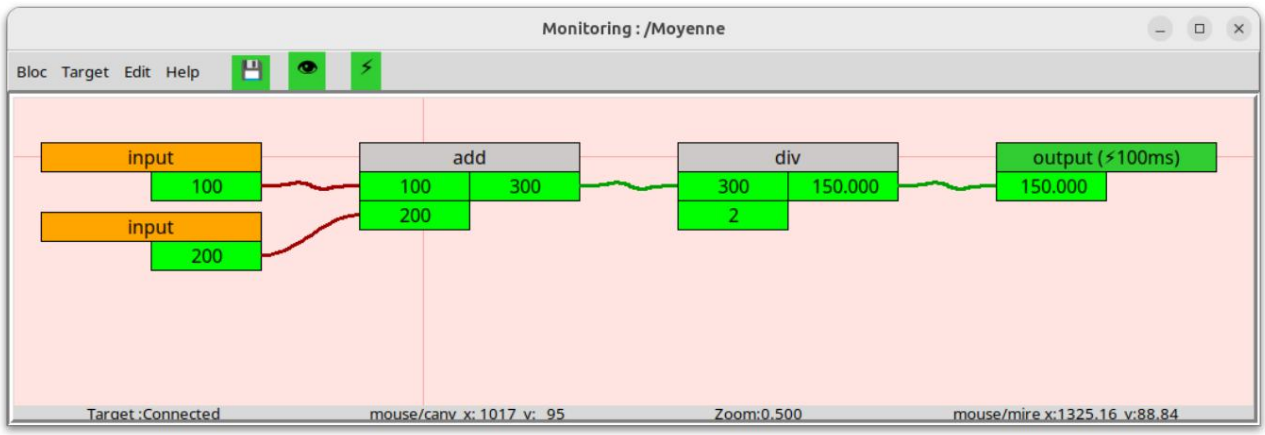


Figure 3: Block in dynamic visualization mode

- 3.5 - Validity bit

The validity bits can be read or modified using the following **system** blocks :

- <validRead>: allows reading the validity bit; this allows, for example, selecting a fallback position in the case of "invalid" variables
- <valideWrite>: allows writing the validity bit, allowing the user to define their own validity equation.

- 4 - The Executioner

- 4.1 - Generalities

Reminder: **The executor** is installed on a PC called the "Target Machine", it executes the programs (user **blocks)** that control the industrial equipment using the remote inputs/outputs.

Certain commands related to **user** blocks already present in **the executor,** such as: startup, Stopping, initializing, swapping, and deleting can be accomplished by **the executor** through its integrated web server (see chapter #7.5).

Operations to modify a **user** block or to view in real time the values of variables of **user** blocks in execution, are only available from **the Block Editor.**

- 4.2 - Method of execution

For a chain of blocks within a **user** block to be executable, it is essential that the chain terminates with an `<output>` block and that this block be associated with an event (such as the periodic event at 100ms). **The executor** periodically evaluates all `<output>` blocks within the **user block,** triggering the evaluation of the preceding block, which in turn requires the evaluation of the preceding block, and so on. This approach eliminates the need to explicitly define the execution order.

A **user** block can contain multiple `<output>` blocks, each `<output>` being associated with a different periodic event. Therefore, it's possible to include within the same **user** block chains that will be executed with different periodicities. If there are links between block chains with different periodicities, a ` **system`** block must be inserted on each inter-chain link.
<previous>, so that the string that needs to be executed slowly is not executed at the period of the fastest string.

- 4.3 - Hotswap

To enable on-the-fly loading, **the Executor** has **a user** for each block (program) of two versions **Shift A** and **Shift** B. The new version of a **user** block is always downloaded into the **Shift** that is not running; when a "hot swap" is initiated, the current values of the variables of the running **Shift** are transferred to the other **Shift,** which then takes over.

- 4.4 - Validity bit

Generally speaking, as long as a block has not been evaluated, its outputs are "invalid".

The validity associated with each output is automatically evaluated when the block is executed. If one of the inputs required to evaluate the output is "invalid" or if the block's output cannot be evaluated due to a type conflict or other issue, the output will be "invalid"; otherwise, it will be "valid".

- 4.5 - Web Server

- 4.5.1 - General Information

An HTTP server is integrated into **the Executor,** allowing the user, via a simple web browser, to view the status of the Target Machine:

- View available tasks and the number of associated <output>
- View overall and per-task CPU load
- View the list of blocks available in the Target Machine and the status of the **Shifts** (A & B) (Pending, Ready, Running)
- Visualize the state and value of the <output> of all blocks for **Shifts** (A & B)
- Visualized the current Ethernet connections managed by **the Executor**

It also allows you to place simple commands such as:

- Stop the execution of a **user** block (command: Stop)
- Start the execution of a **user** block (command: Run)
- Initialize a **user** block (command: Initialize)
- Delete a **user** block (command: Delete)
- Swap **Shift A** and **Shift B**
  - when no **Shift** is "Running" (command: ClodSwap)
  - when **a Shift** is "Running" (command: HotSwap)
- Run commands dedicated to debugging

  ÿ (commande : print list_compiled)

  ÿ (commande : print list_threads)

- 4.5.2 - Example of a "Task list" page

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu

Task list
Bloc & Output list
Connexions
print list_compiled
print list_threads

Task list

Setting			Feedback					
name	id	period	cycle time	cycle time min	cycle time max	counter	number of output	CPU load
10s	0	10s	10.000011s	9.999804s	10.000738s	155	0	0.000%
3s	1	3s	3.000265s	2.999798s	3.000808s	518	0	0.000%
1s	2	1s	1.000031s	0.999822s	1.002054s	1555	2	0.004%
200ms	3	200ms	200.046ms	199.769ms	201.699ms	7775	0	0.001%
100ms	4	100ms	99.994ms	99.745ms	103.087ms	15547	0	0.003%
50ms	5	50ms	49.937ms	49.749ms	52.764ms	31083	0	0.010%
20ms	6	20ms	19.953ms	19.744ms	22.160ms	77657	0	0.020%
10ms	7	10ms	9.996ms	9.743ms	12.555ms	155273	0	0.043%
5ms	8	5ms	5.028ms	4.743ms	7.809ms	310276	0	0.062%
2ms	9	2ms	2.022ms	1.708ms	3.533ms	792271	0	0.379%

User tasks CPU load = 0.52%

Refresh

- 4.5.3 - Exemple de page « Bloc & Output list »

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu
Task list
Bloc & Output list
Connexions
print list_compiled
print list_threads

bloc list			
bloc name	status		order
	shift A	shift B	
call2loops	Pending	Running	Stop Initialize HotSwap

bloc output list									
bloc			output					task	
name	shift	building time (yyyy/mm/dd)	name id		value	validity	status	name id	
call2loops A		2025/07/30 - 18:19:10	out_1	5	...	...	Down	1s	2
call2loops A		2025/07/30 - 18:19:10	out_1	10	...	...	Down	1s	2
call2loops	B	2025/07/30 - 18:25:40	out_1	5	1303		Running	1s	2
call2loops	B	2025/07/30 - 18:25:40	out_1	10	6515		Running	1s	2

Refresh

- 4.5.4 - Example of a "Connections" page

Target: jbar-HLYL-WXX9 (ip:192.168.5.58)

Menu
Task list Bloc & Output list Connexions print list_compiled print list_threads

HTTP protocol			
...	@ip	port	Host name
Server	192.168.5.58	80	jbar-HLYL-WXX9
Last Client 192.168.122.1		39032	Jbar-HLYL-WXX9

TCP/IP protocol		
...	@ip	port
server	192.168.5.58	8000
client[0]	192.168.122.1	49062

Refresh

- 4.6 - Create a system block

To add a block to the standard library, you need to:

1. Create the external interface (graphical interface) of the new block.
2. Code the "Python" function associated with the new block and add the block name to the list of blocks in the standard library.

Note that you will need to restart **the Executor** and **the Block Editor** for this new system block to take effect .

- 4.6.1 - Create the external interface

Open **the Block Editor.**

In the menu bar, click on "Block" then "Change Properties" to open the following "block info" window:

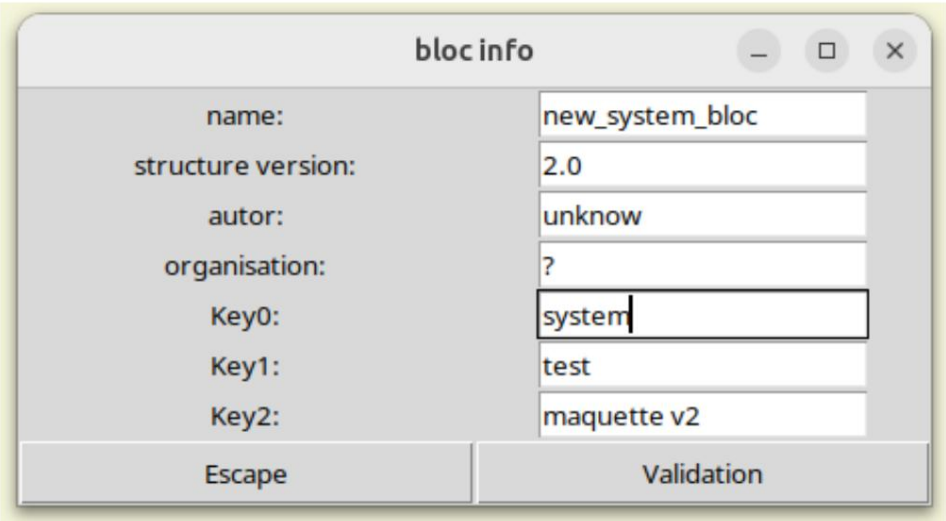


Figure : bloc info

Enter the name of the new **system** block in the "name:" field. Enter "system" in the "Key0:" field. Confirm your entry by clicking "Validate".

In the graphics area, add **system** blocks <input> and/or <output> depending on the number of inputs. and the necessary outputs, rename these inputs/outputs and give them a default value, then press the [F3] key to save.

To verify that the interface has been created correctly, open a new window and insert a **system block;** the newly added block should appear in the list, select it, and verify that the external interface corresponds to what you want.

- 4.6.2 - Create the Python code

The functions associated with the **system** block are in the /Target/exec.py file; for example, here is the Python function associated with the <and> **system** block:

```
def c_exesubloc_and (pebloc, pieb, pio, pcounter):
    """ exécution du bloc a et b (dans la boucle récurcive)"""
    global exec_level
    exec_level +=1
    cesubloc = pebloc.sublocs[pieb]
    if debug_blocs:
        trace_txt = trace_proc(cesubloc, inspect.currentframe().f_code.co_name, exec_level)
        print (trace_txt, "début AND les paramètres sont: pieb=", pieb, ",   pio=", pio, ",   counter=", pcounter)
    if cesubloc.header['counter'] == pcounter:
        if debug_blocs: print (trace_txt, "  cesubloc['counter'] == pcounter: =", pcounter, "   (output[", pio, "] inchangée)")
    else:
        cesubloc.header['counter'] = pcounter
        try:
            pebloc.c_exebloc_recup_inputs(pieb, pcounter)
            cesubloc.c_exesubloc_validation_standard()

            cesubloc.outputs[0]['var'] = cesubloc.inputs[0]['var'] and cesubloc.inputs[1]['var']
        except:
            trace_txt = trace_proc(cesubloc, inspect.currentframe().f_code.co_name, exec_level)
            print (trace_txt, PARAM_TEXT_EXCEPTION)
            cesubloc.outputs[0]['valide'] = False
    if debug_blocs: print (trace_txt, " AND retourne l'outout [", pio, "]: var=", cesubloc.outputs[pio]['var'], "val=", cesubloc.outputs[pio]['valide'])
    exec_level -=1
    return cesubloc.outputs[pio]['var'], cesubloc.outputs[pio]['valide']
```

Figure: fonction « c_exesubloc_and »

If we disregard the elements related to debugging, only this remains:

```
1 def c_exesubloc_and (pebloc, pieb, pio, pcounter):
2         thissubblock = thisblock.subblocks[pieb]
3         if cesubloc.header['counter'] == pcounter:
4                 pass
5         else:
6                 cesubloc.header['counter'] = pcounter
7                 try:
8                         pebloc.c_exebloc_recup_inputs(pieb, pcounter)
9                         cesubloc.c_exesubloc_validation_standard()
10                        cesubloc.outputs[0]['var'] = cesubloc.inputs[0]['var'] and cesubloc.inputs[1]['var']
11                catch:
12                        cesubloc.outputs[0]['valide'] = False
13        return cesubloc.outputs[pio]
```

Explanation of the code:

Line 1: The function name is "c_exsubloc_" followed by the *system block name.* The parameters are:

- **pebloc :** is the "executable" structure of the **user** block currently being executed.
- **pieb :** is the index of the **system** block to execute
- **pio :** is the index of the output whose value needs to be returned

Line 2: allows you to point to the block to be executed.

Line 3: checks if the block has already been executed

Line 6: marks the block as already executed

Line 8: retrieves all entries from the block

Line 9: assigns all output enable bits of the block based on its inputs

Line 10: assigns the output based on the inputs (logical AND)

Line 12: sets the validity bit to "invalid" in case the block calculation did not complete correctly

Line 13: returns the requested output (pio index output)

In most cases, your new function will be identical to that of the **system** block `<and>`, except:

- line 1: since the function name will be different.
- line 10: since you will assign the outputs according to the inputs as needed.

Still in the file: /Target/exec.py, you need to add the association between the name of your new **system** block and the newly created Python function; this takes place in the "recup_procedure" function.

Note that this function is used by **the Block Editor** when a **user** block is compiled:

```
def recup_procedure(psubloc):
    proc_name = "recupe_procedure"
    """ ajout l'adresse de la procédure qui correspond au nom du bloc"""
    if   psubloc.header['name'] == PARAM_NAME_BLOC_DT:          procedure= c_exesubloc_dt
    elif psubloc.header['name'] == PARAM_NAME_BLOC_OUTPUT:      procedure= c_exesubloc_output
    elif psubloc.header['name'] == PARAM_NAME_BLOC_INPUT:       procedure= c_exesubloc_input
    elif psubloc.header['name'] == PARAM_NAME_BLOC_PREVIOUS:    procedure= c_exesubloc_previous
    elif psubloc.header['name'] == PARAM_NAME_BLOC_SELECT:      procedure= c_exesubloc_select
    elif psubloc.header['name'] == PARAM_NAME_BLOC_VALIDREAD:   procedure= c_exesubloc_validRead
    elif psubloc.header['name'] == PARAM_NAME_BLOC_VALIDWRITE:  procedure= c_exesubloc_validWrite
    elif psubloc.header['name'] == PARAM_NAME_BLOC_COMP:        procedure= c_exesubloc_comp
    elif psubloc.header['name'] == PARAM_NAME_BLOC_AND:         procedure= c_exesubloc_and
    elif psubloc.header['name'] == PARAM_NAME_BLOC_OR:          procedure= c_exesubloc_or
    elif psubloc.header['name'] == PARAM_NAME_BLOC_NOT:         procedure= c_exesubloc_not
    elif psubloc.header['name'] == PARAM_NAME_BLOC_EDGE:        procedure= c_exesubloc_edge
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CABLIN:      procedure= c_exesubloc_cablin
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CABLOUT:     procedure= c_exesubloc_cablout
    elif psubloc.header['name'] == PARAM_NAME_BLOC_ADD:         procedure= c_exesubloc_add
    elif psubloc.header['name'] == PARAM_NAME_BLOC_SUB:         procedure= c_exesubloc_sub
    elif psubloc.header['name'] == PARAM_NAME_BLOC_MULT:        procedure= c_exesubloc_mult
    elif psubloc.header['name'] == PARAM_NAME_BLOC_DIV:         procedure= c_exesubloc_div
    elif psubloc.header['name'] == PARAM_NAME_BLOC_MINMAX:      procedure= c_exesubloc_minmax
    elif psubloc.header['name'] == PARAM_NAME_BLOC_CONST_PI:    procedure= c_exesubloc_const_pi
    else:
        print (proc_name, " ERROR: function not defined for this bloc <"+psubloc.header['name']+">")
    return procedure
```

Figure: function "recup_procedure"

To do this, you will need to add a line before the "else" and create a new parameter whose name will consist of:

“PARAM_NAME_BLOC_” followed by the name of your new **system** block in uppercase.

You will also need to modify the file /Target/PARAM_NAME_BLOC.py, by adding a line to define the constant "PARARM_NAME_BLOC_yourblock ", the string of characters located to the right of the sign "=" corresponds to the file name of your new **system block.**

```
# ATTENTION: la source de ce fichier ce trouve dans le répertoire "Target"

# nom des blocs "system"
PARAM_NAME_BLOC_DT = "dt"
PARAM_NAME_BLOC_OUTPUT = "output"
PARAM_NAME_BLOC_INPUT = "input"
PARAM_NAME_BLOC_PREVIOUS = "previous"
PARAM_NAME_BLOC_SELECT = "select"
PARAM_NAME_BLOC_VALIDREAD = "validRead"
PARAM_NAME_BLOC_VALIDWRITE = "validWrite"
PARAM_NAME_BLOC_MINMAX = "minmax"
PARAM_NAME_BLOC_COMP = "comp"
PARAM_NAME_BLOC_AND = "and"
PARAM_NAME_BLOC_OR = "or"
PARAM_NAME_BLOC_NOT = "not"
PARAM_NAME_BLOC_EDGE = "edge"
PARAM_NAME_BLOC_ADD = "add"
PARAM_NAME_BLOC_SUB = "sub"
PARAM_NAME_BLOC_MULT = "mult"
PARAM_NAME_BLOC_DIV = "div"
PARAM_NAME_BLOC_CABLIN = "cablin"
PARAM_NAME_BLOC_CABLOUT = "cablout"
PARAM_NAME_BLOC_CONST_PI = "const.pi"

```

Figure: PARAM_NAME_BLOCK.py file