Serial signal processing logic of the Nextion display

This intelligent display has the ability to embed all the logic necessary for one-way communication (although two-way communication is also possible). This aspect is very interesting, because it allows not to burden the process of a board that has to control it, like a Raspberry Pi for example. It is thus in the case of a man-machine interface intended for music, as with the Probatio, very adapted to remove a part of the processing if this screen is intelligently programmed.

For the case studied, i.e. that of the Probatio, here is the logic that had been developed :

Array communicated			
Boolean 0	**Integer 1**	**Integer 2**	**Integer 3**
Complexity indicator : --> 0 : Single level	Block selection	Mode / effect selection	Selection of the parameter to be influenced.
--> 1 : Multi-level	Configuration selection	Sub-selection	Additional value (if required)

The global idea here is to prepare all the logic and the interface of the screen to send, by serial communication, only an array of a defined length. The raspberry Pi, for its part, will know in advance the interpretation to be given to the values of the array and will only have to listen to its serial port. In this application case, the Rapberry Pi simply runs Pure Data and collects the serial communication directly from this software. Indeed, thanks to an adapted split of the array, the interpretation can be done directly in Pure Data. In summary, the values of the table are intended to activate and deactivate precise links between the sound / effect blocks and the signal inputs of the Probatio (see "mapper" block of the Libmapper project).

It is therefore important to prepare both the Probatio "mapper" inputs, activatable and deactivatable links, and Pure Data sound and effect blocks. Two examples of these Pure Data blocks are available in a folder in this directory.

Moreover, the preparation of these blocks is the very basis of what the user makes available or not. This is why the preparation stage is so important. It is also possible to prepare preset combinations of several types of blocks. If this is the case, it will be possible to activate them from the screen. In the rest of this document we will call them multi-level sets.

In terms of screen and array logic, things have been thought through to allow simple addition of new block types and new effects and sounds to Pure data. As can be seen in the table above, the arrays are divided into four elements and are always the same length.

The first element of the array is a boolean and is only used to indicate whether it is a single block change or whether it is the activation of a user-defined multi-level set.

In the case of a **single level change**, **the second element** is used to select a block. For example, number 3 could be associated with the block with two buttons and number 4 with a block with a joystick. This second element therefore selects the Probatio input you are working with.

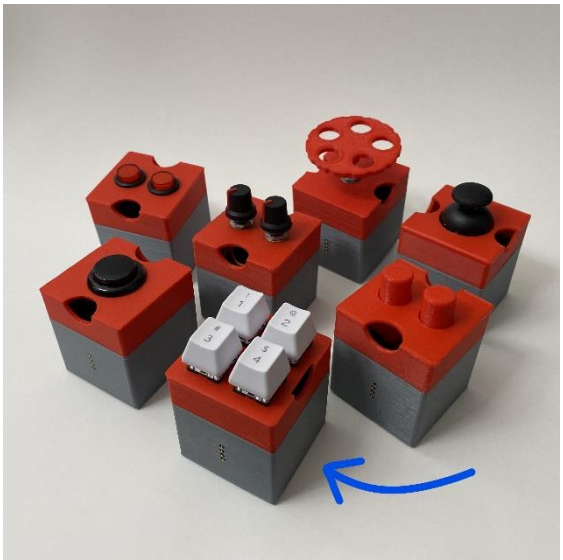
Still in the **single level**, **the third element** allows you to select the sound / effect you want to work on. The idea is to propose in the interface all the effects possible with the selected block. Once the user selects an effect or sound, the logic of the screen assigns a known number to the third element of the table.

Finally, as effects or sounds often have several possible inputs, **the fourth element** allows you to select the parameter modified by the block.

N.B.: In the case of a block with several controllable values, the integer 1 is also used to select the block parameter. For example, the first piston of a two-piston block may have the number 11 assigned and the number 12 for its second piston. In addition, the interpretations that you can make with Libmapper should not be forgotten. In all cases, the second element of the table can be used to select the desired element. The only thing to do is to give it a number and to correctly create its link in Pure Data.

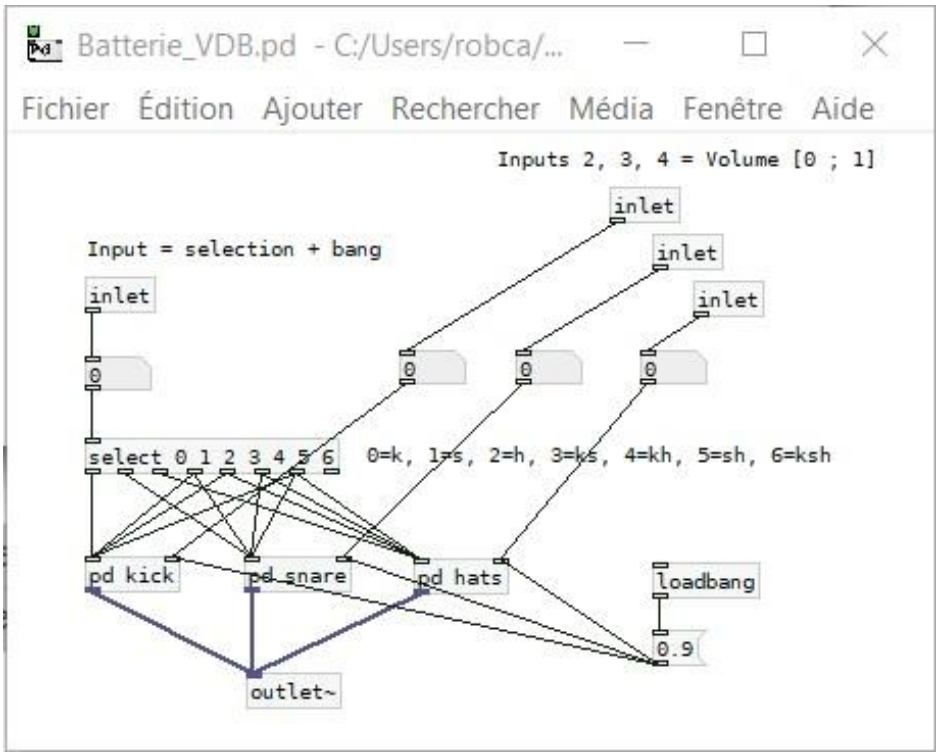
In the multi-level case, the integers 1, 2 and 3 are primarily selection and sub-selection elements. Thus, they are to be used according to the user's creations. For example, if there are sub-selections of a multi-level setting the user can use integers 2 and 3 to be precise in his selection. It is also possible that only the first integer is useful. However, it is advisable to keep the length of the array even if integers 2 and 3 are not useful.

To be clear, here is an example:



1. First of all, I want to make a classic single-level change. The first element is therefore **0**.
2. After some reflection, I would like to work with the four-button block. Also, I'll start with button 1. In this case, the display system validates the number **21** as the second element, which corresponds to button 1 of the four-button block.
3. After this block selection, I choose the drum sound for this button. Element three will therefore be number **5**.
4. Below, you can see a Pure Data patch for a drum kit. As you can see, thanks to the "inlet" element, several inputs are possible. Thanks to the element four we will then choose this input. The button will thus, according to my choices, be assigned to a kick drum sound. This selection corresponds to cell 1 of the battery. This selection corresponds to cell 1 of the battery in which

a 0 must be sent at each impulse. For this, the integer 3 will be a **1**. (The system for sending a 0 at each impulse is not shown here).



At the end of the selection, the smart screen system therefore knows that it can send to the Raspberry Pi the array: **[0 ; 21 ; 5 ; 1]**.

As for the displayed interface, it is very simple and corresponds to a sub-drawer menu:

Page 1 : Simple or multi-level

Page 2 : List of blocks

Page 3 : Effects / sounds available for this block

Page 4 : Lists of accessible audio synthesis parameters.

As needed:

Page 2 bis : List of parameters linked to the block.

When page 4 is validated, the system sends a complete array.