A step by step guide to build a Ctrlr panel

Using the Moog Sub37 as example

v1.3 – 2016-03-31

Note: the content of this Step by step document is at the moment only based on my understanding of Ctrlr. If errors in understanding, just let me know ☺

Introduction

Hi!

I'm a newbie in Ctrlr and I decided to describe my step by step discovery of this tool. First of all, despite the lack of a manual, we must all congratulate Atom for the creation of this higher level Midi controller software based on lower level components like Juce.

As he describes on the Ctrlr web site, the best way to learn is to look to the different existing panels and to learn from there. In the Panels section of the web site there is also a special one called **Demo panels**. I would add one very important point: you must have a goal. My goal was to implement a librarian for my brand new **Moog Sub37**.

In the 90's I wrote in C and assembler a program called DXDOS that allowed to retrieve and send sound banks of my DX7. So I was ready to take that direction but as we all know, things didn't stay static in the MIDI and software world during the last 25 years. The VSTi power to integrate almost any software emulated synth or effect in our DAWs is one of the most noticeable example. By browsing on the web I found thus a lot of documentation and possible way to build some librarian in today's techniques and decided to settle for Juce (without knowing any details of it at that moment) when I discovered Ctrlr that would allow me to go faster because at higher level.

Then I found the Moog Minitaur panel made by Stoner and my decision was taken: steel with pride, copy and adapt to the Sub37.

Yes, but from where do I start? Is there some documentation? Well, besides a good *Getting started* there is none. Looked in the forum and I saw I was not the only one requesting this. So, back to the beginning of this introduction, let's go for a step by step guide and have our goal set to the panel creation for the Moog Sub37.

Then I realized by browsing further in the forum that there were many posts containing key information, basic scripts and basic tips.

In the Step by step part I'll only document the things that I will address and that I will understand. So, my apology if I'm not addressing the particular topic specific for you own panel. At this stage, this document is a Step by step guide and not a complete reference manual.

I have seen many great things done by the Ctrlr user community mainly programming wise and of course this could also be added later on (for example a graph showing the EG curves).

At the end of the document, in the Appendix you'll find a list of topics that should be addressed later on, a list of (possible) bugs, enhancements and open questions but also a few references to useful posts written by the Ctrlr community. Thanks to all!

Good reading…Dominique (*goodweather*)

Table of Contents

Introduction **2**

Step 1 – Let's start with Ctrlr **5**

Installation 5

Starting Ctrlr and creating a new panel 6

Saving your panel 7

Exporting your panel 7

Step 2 – Adding some tabs and the main panel **8**

The uiTabs modulator 8

Background panel images 9

Step 3 – Adding buttons **11**

Button images concept 11

Adding a rotary button 12

The modulator list window 14

Copy/Paste buttons 14

Adding a toggle button 15

Adding a momentary button 16

Step 4 – Sending Midi data **17**

Understanding the Midi specifications of your device 17

Preparing modulators for sending a CC midi message 18

Preparing modulators for sending a NRPN midi message 18

The MIDI monitor 19

Setting Ctrlr to send data and testing 20

Step 5 – Introduction to automation with Lua **22**

What is Lua? 22

Lua references 22

Some automation 22

Step by step 22

Appendix A **26**

Version history 26

Things I didn't find 26

Identified things to add in this manual 26

Identified possible bugs in v5.3.94 26

Possible enhancements related to the topics treated in this manual 27

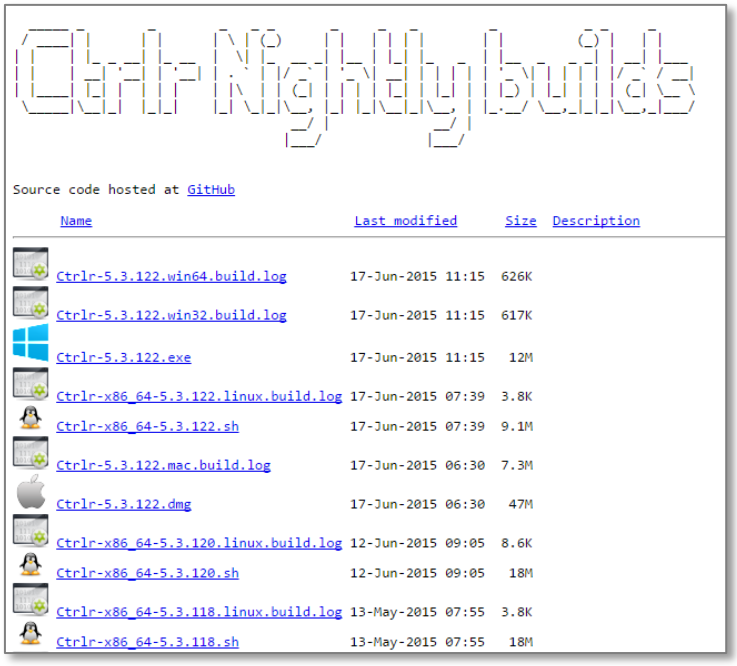
Appendix B **28**

Ctrlr methods documentation..... 28

Step 1 – Let's start with Ctrlr

Installation

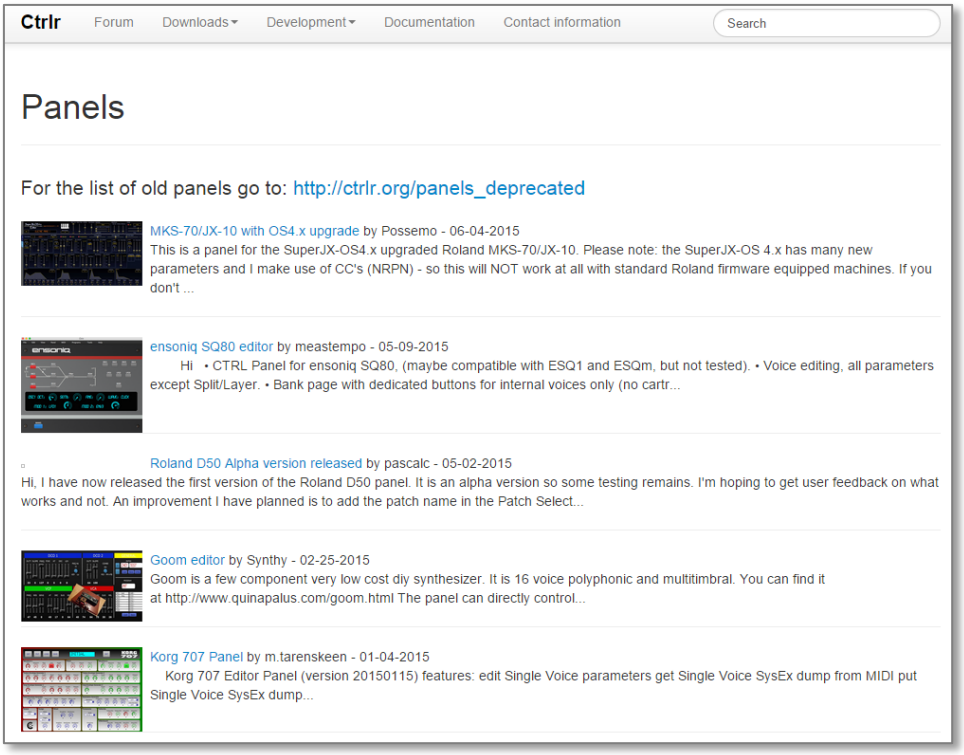
Install Ctrlr by downloading it from the **ctrlr.org** web site (you need to search for the last version by date and operating system).



Source code hosted at GitHub

Name	Last modified	Size	Description
Ctrlr-5.3.122.win64.build.log	17-Jun-2015 11:15	626K	
Ctrlr-5.3.122.win32.build.log	17-Jun-2015 11:15	617K	
Ctrlr-5.3.122.exe	17-Jun-2015 11:15	12M	
Ctrlr-x86_64-5.3.122.linux.build.log	17-Jun-2015 07:39	3.8K	
Ctrlr-x86_64-5.3.122.sh	17-Jun-2015 07:39	9.1M	
Ctrlr-5.3.122.mac.build.log	17-Jun-2015 06:30	7.3M	
Ctrlr-5.3.122.dmg	17-Jun-2015 06:30	47M	
Ctrlr-x86_64-5.3.120.linux.build.log	12-Jun-2015 09:05	8.6K	
Ctrlr-x86_64-5.3.120.sh	12-Jun-2015 09:05	18M	
Ctrlr-x86_64-5.3.118.linux.build.log	13-May-2015 07:55	3.8K	
Ctrlr-x86_64-5.3.118.sh	13-May-2015 07:55	18M	

Then download the panel(s) of your choice. In my case it is the Moog Minitaur panel created by Stoner and I'll refer to that one from now on.



Ctrlr Forum Downloads Development Documentation Contact information Search

Panels

For the list of old panels go to: http://ctrlr.org/panels_deprecated

MKS-70/JX-10 with OS4.x upgrade by Possemo - 06-04-2015
This is a panel for the SuperJX-OS4.x upgraded Roland MKS-70/JX-10. Please note: the SuperJX-OS 4.x has many new parameters and I make use of CC's (NRPN) - so this will NOT work at all with standard Roland firmware equipped machines. If you don't ...

ensoniq SQ80 editor by meastempo - 05-09-2015
Hi • CTRL Panel for ensoniq SQ80, (maybe compatible with ESQ1 and ESQm, but not tested). • Voice editing, all parameters except Split/Layer. • Bank page with dedicated buttons for internal voices only (no cartr...

Roland D50 Alpha version released by pascalc - 05-02-2015
Hi, I have now released the first version of the Roland D50 panel. It is an alpha version so some testing remains. I'm hoping to get user feedback on what works and not. An improvement I have planned is to add the patch name in the Patch Select...

Goom editor by Synthy - 02-25-2015
Goom is a few component very low cost diy synthesizer. It is 16 voice polyphonic and multitimbral. You can find it at http://www.quinapalus.com/goom.html The panel can directly control...

Korg 707 Panel by m.tarenskeen - 01-04-2015
Korg 707 Editor Panel (version 20150115) features: edit Single Voice parameters get Single Voice SysEx dump from MIDI put Single Voice SysEx dump...

Starting Ctrlr and creating a new panel

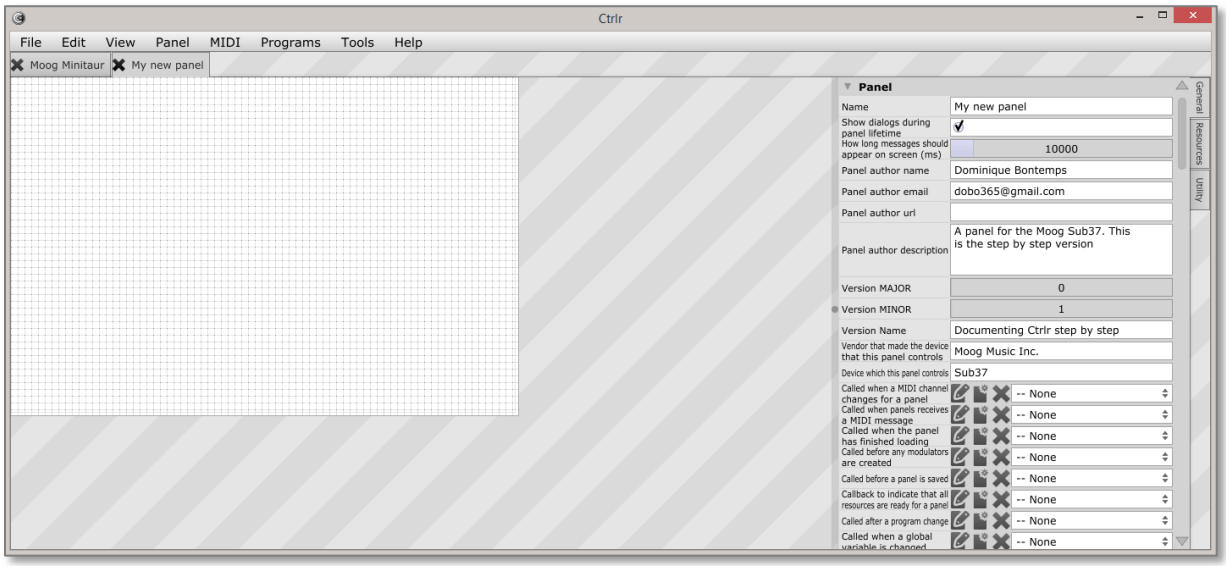
Start Ctrlr and open the Minitaur panel. Woaw! So nice!

Select **Panel** → **Panel Mode** to display the properties of all elements. To come back in normal mode, just select **Panel Mode** again (or Ctrl-E)

Under **File**, select **New Panel** to create your new panel.

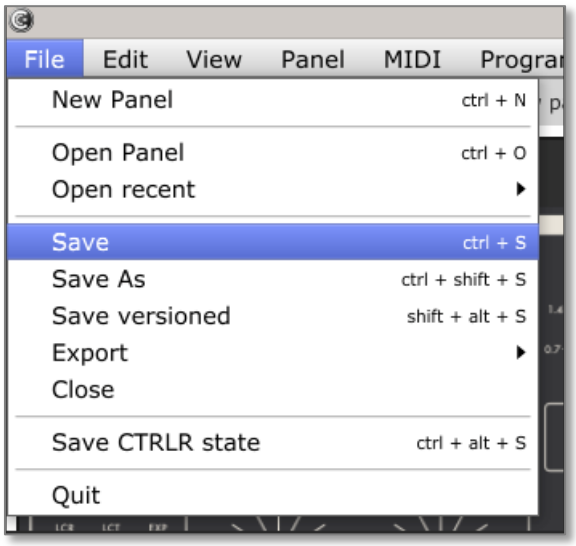
Enter some properties in the Properties panel visible on the right side:

- **Panel – Name**: the name of your panel (will appear as tab name if you have several panels opened at the same time)
- **Panel – Panel author name**
- **Panel – Panel author description**
- **Panel – Version MAJOR** and **Version MINOR**: set it to 0.1 (0 for major and 1 for minor)
- **Panel – Vendor**, **Panel – Device**: Manufacturer and Device names



Saving your panel

Under **File**, select **Save** or **Save As** to save your panel. A file with .panel as extension will be created (it is an XML file).

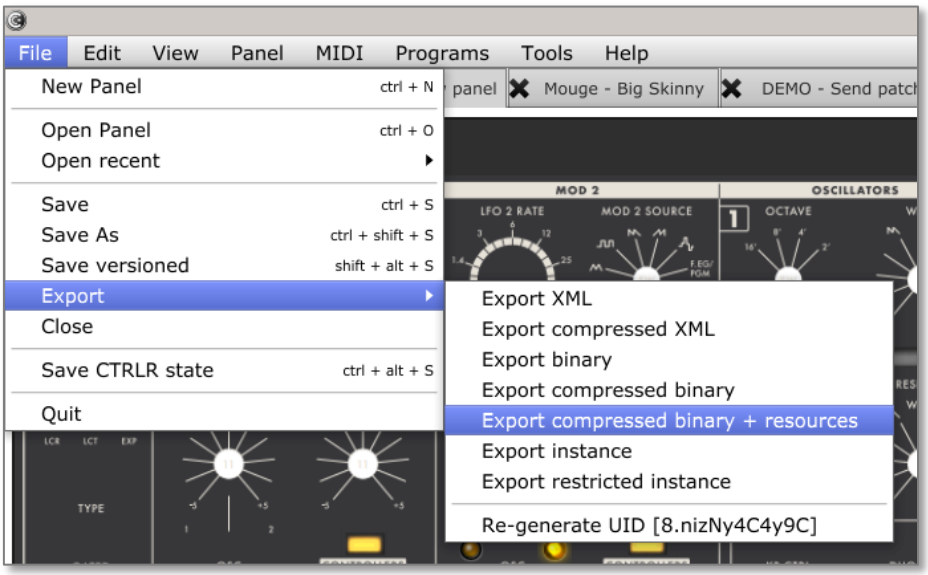


As always, it is better to perform too many saves than losing some work! Use the versioning proposed, *save* names are adapted and proposed automatically.

Exporting your panel

Ctrlr reads and can export its data in different formats:

– .panel – XML
– .panelz – compressed XML (gzip)
– .bpanel – Binary
– .bpanelz – Binary compressed (the best way to share on the net, smallest and loads quick)

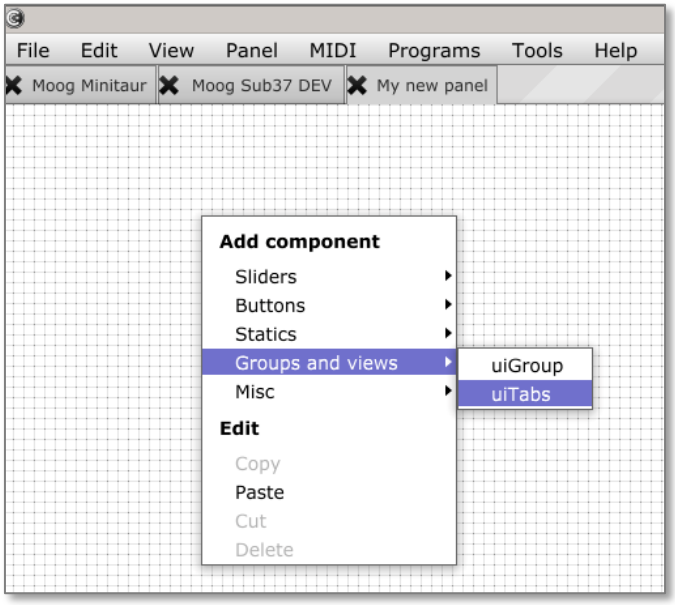


Step 2 – Adding some tabs and the main panel

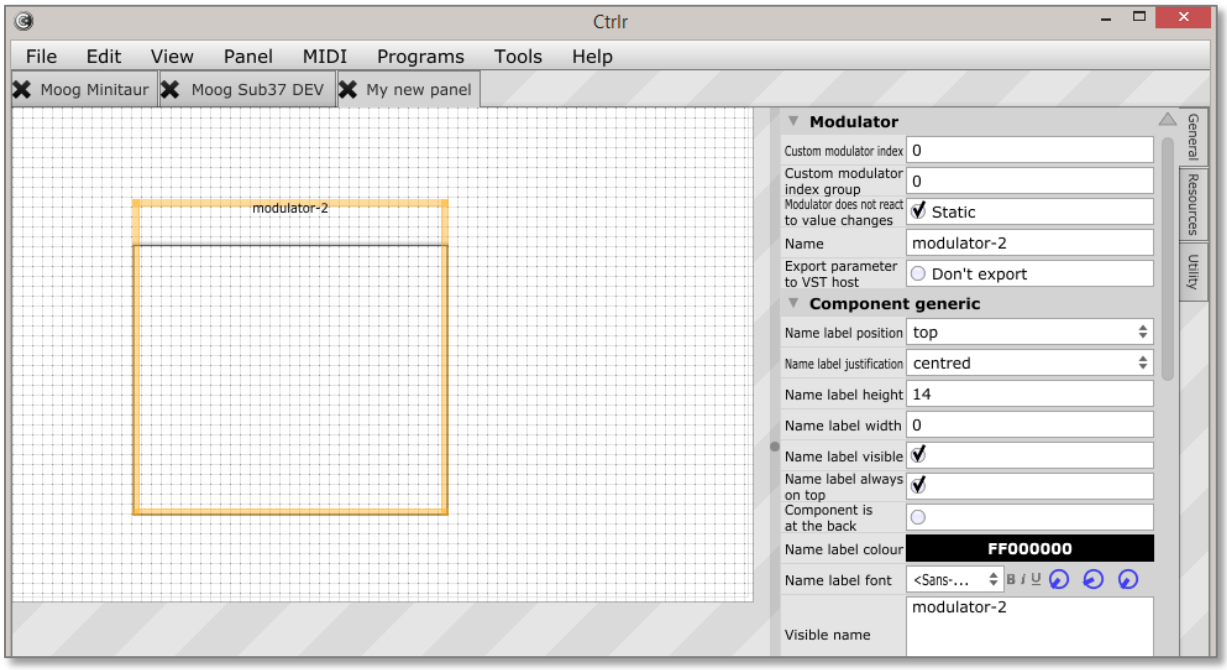
In Ctrlr all components have parent modulators (in fact a component is embedded within a modulator). Modulators have properties and components have properties.

The uiTabs component

Right click on the panel grid and select **Add component → Groups and views → uiTabs**



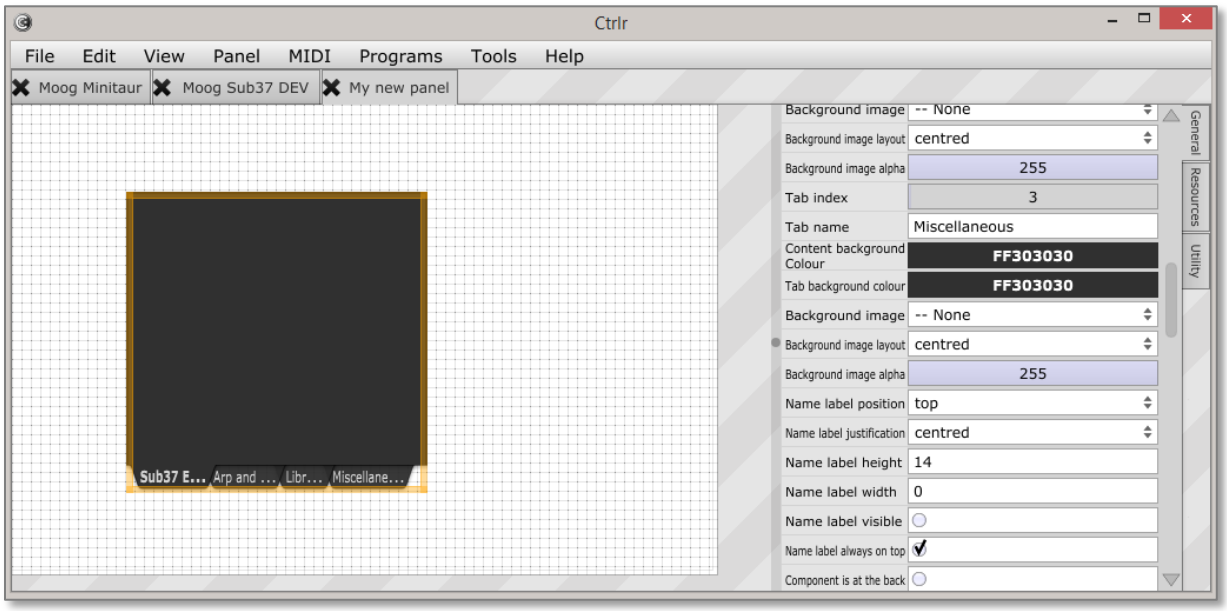
You should see a component called “modulator-2”. Click on it to select it. The property panel on the right side will now show the uiTabs properties.



Scroll through the property panel and read the different properties to have an idea of the many fine tuning possibilities.

We will now add 4 tabs and adjust a few layout parameters:

- Scroll down completely at the bottom of the property panel to see the Add tab button. Click 4 times on it in order to have 4 tabs. The component turns yellow and 4 tabs named Tab0, Tab1, Tab2 and Tab3 are created. To see the effect of your change, you need to click on an empty area of the panel then to reselect the uiTabs modulator.
- **Modulator – Name**: give a name to the modulator; for example *Tabs*
- **Component generic – Tab name**: *Sub37 Editor*, *Arp and Seq, Librarian, Miscellaneous* (you give a name to each tab)
- **Component generic – Content background color**: the background color of each tab window. Set it to *FF303030*. The 6 last digits in hexadecimal are representing the RGB values of the color. 00 is the lowest level and FF is the maximum level for each R, G and B. 00000 is black, FFFFFF is white. Look at http://www.w3schools.com/html/html_colors.asp for example to get the HEX values of colors. The 2 first digits are the alpha level (transparency setting). To see the effect of your change, you need to click on an empty area of the panel then to reselect the uiTabs modulator.
- **Component generic – Tab background color**: the background color of each tab heading. Set it to *FF303030*. Same explanation as above for the hexadecimal setting.
- **Component generic – Name label visible**: *OFF*. This will hide the name of the currently selected modulator.
- **Component generic – Visible name**: *Moog Sub37 tabs*. The label of the currently selected modulator.
- **Component – Tabs orientation**: *TabsAtBottom*. The positioning of the tabs labels.
- **Component – Tab text font (current)**: *bold ON*. The font of the selected tab labels.

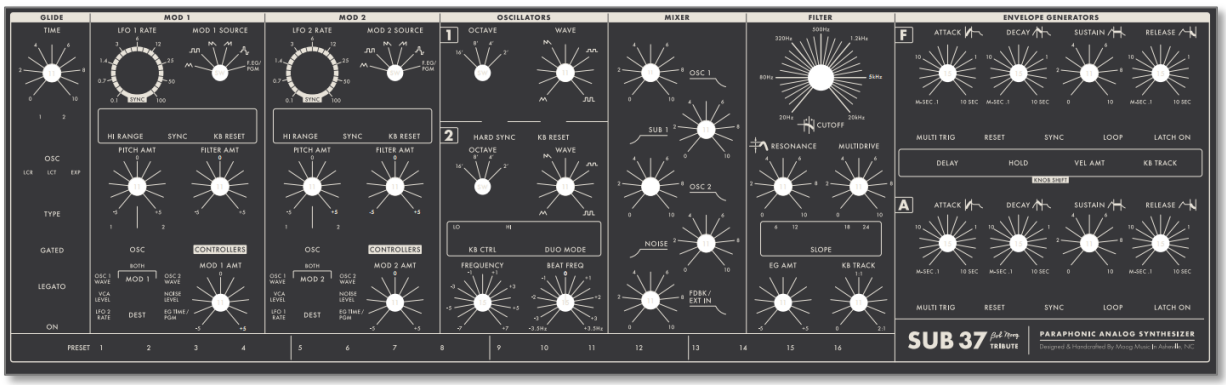


Background panel images

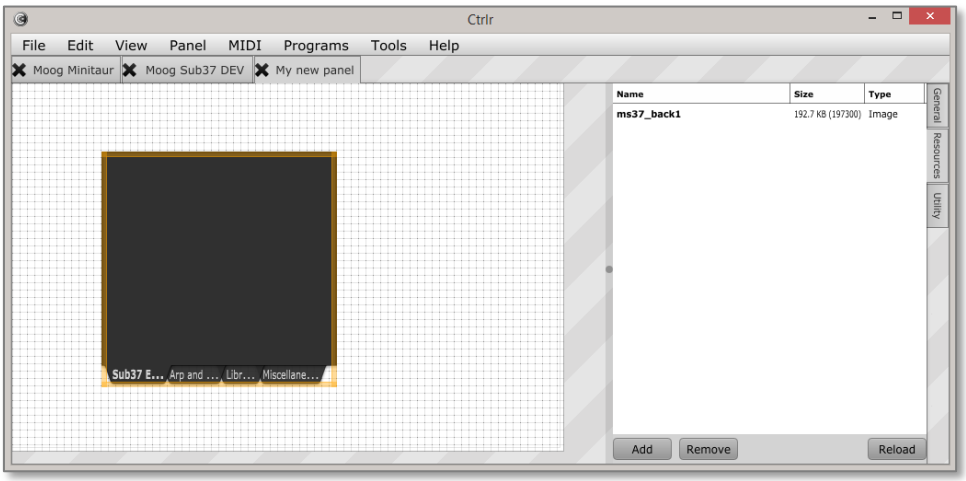
You may keep the panel background as plain color but another possibility is to find an existing panel layout (if possible without any button) on internet.

For the Sub37 I found the actual panel on the Moog Music site and also a panel made by a Sub37 owner to register his patches. I took the Moog Music panel and adjusted it by removing the

Programming and Arpeggiator sections from the left and the Output section from the right (this may evolve during the life cycle of this project ☺).



Before being able to use an image in the panel it is needed to upload it in **Resources**. On the right side of the property panel, vertically, click on the **Resources tab** then upload your panel image(s) by clicking on the **Add button**.



Now, go back to the **General tab** of the property panel and use the image(s) as background for tab(s):

– **Component generic – Background image**: select the background image for each tab. For me, the Sub37 back panel in the tab Sub37 Editor. I'll see later on for the other tabs.
– Resize the panel (white area)and the uiTabs modulator (orange border) by moving the mouse on their borders and dragging the border until you reach a good size for your tab/panel

Step 3 – Adding buttons

Button images concept

First of all, you should know that when you open a panel in Ctrlr, the images used are available in a temporary folder under C:\Users*your_user_name*\AppData\Roaming\Ctrlr*temp_folder_name* where *temp_folder_name* is something like *8.nTdcoD3iacC* (under Windows).

This means that you have the possibility to re-use existing nice button or panel images from the panels provided on the Ctrlr web site if you find some nice ones. For my Sub37 panel, I'm re-using the buttons used by Stoner in his Minitaur panel. Great feature and thanks Stoner!

Secondly, buttons, either rotary or toggle, do have different states. A toggle button will typically have only 2 states while a rotary button will have many to describe its different angular positions or values. Midi-wise, a button returning 127 values should have 127 positions.

This is rendered in the same "container" image by putting side by side – either horizontally or vertically – an image (frame) corresponding to each button position/value. Each frame should have the same size that will be indicated in Ctrlr properties.

Examples (the 3 first ones from Stoner's Minitaur panel):

The first image is an extract of a 127 frames 80x10160 pixels image (10160=127x80), the second one a triple state button, the third and last ones ON-OFF toggle switches.

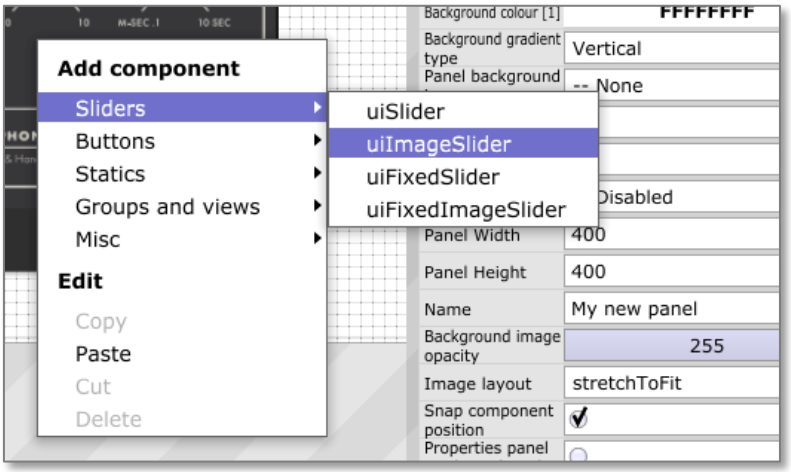
The last one has been done by taking actual pictures from my Sub37 then by tweaking with different image editing software. To be honest, the most difficult part was to size the two images to get exactly the same button size and to position them exactly at 25-75% of the height and 50% of the width (use rules!). This is required to have the frame concept working fine as you need to specify the frame size in the properties.

I think you can also design the buttons with a 2D or 3D CAD system and there is also a nice freeware called **Jknobman** that supports creating buttons and all their frames. You can download it at http://www.g200kg.com/en/software/knobman.html .

As for the main panel, before being able to use an image for a button it is needed to upload it in **Resources**. On the right side of the property panel, vertically, click on the **Resources tab** then upload your button image(s) by clicking on the **Add button**.

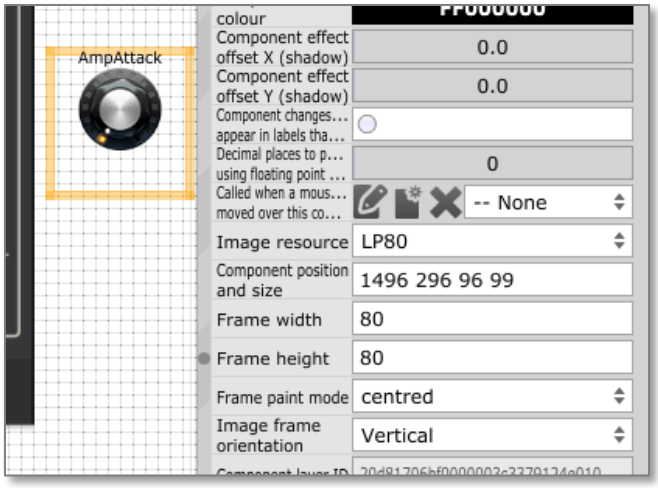
Adding a rotary button

Right click on the panel grid and select **Add component → Sliders → uiImageSlider**



As before, set different properties:

- **Modulator – Name**: give a name to the modulator; for example *btnAmpAttack* (I saw a post from Atom advising to keep names to max 8 characters in the case the panel is used as VST – to check)
- **Component generic – Visible name**: *Amp Attack*. In case of, we will hide it anyway.
- **Component generic – Name label visible**: *OFF*
- **Component generic – Image resource**: *your_rotary_button_image* (for me: *LP80*)
- **Component generic – Frame width**: *80* (the width of each frame)
- **Component generic – Frame height**: *80* (the height of each frame)
- **Component generic – Image frame orientation**: *vertical* (if you assembled your frames vertically)



- **Component – Slider style**: *RotaryVerticalDrag*
- **Component – Minimum value**: *0*
- **Component – Maximum value**: *127*
- **Component – Value position**: *NoTextBox*
- **Component – Display popup bubble when dragging**: *ON.* This is a nice feature showing the actual value of the button when you turn it. Optional of course…

With the rotary button modulator selected, right-click on it and select **Layout – Send to front** to secure the visibility of the button when we will move it on the panel.

Adjust the button size by pulling the orange border down or up then move the button to its position. Re-adjust size and position precisely.

Check that the following properties are set:

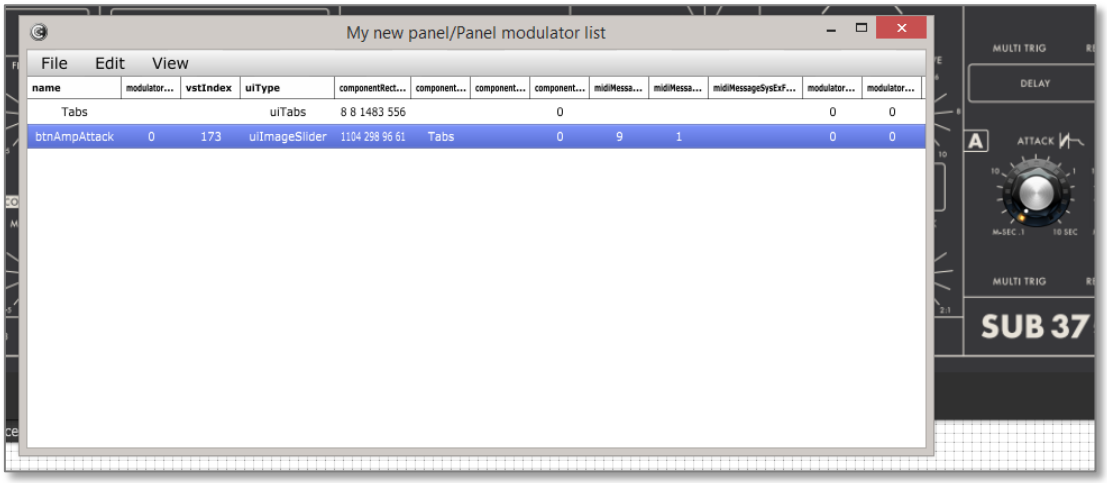
- **Component generic – Group**: *Tabs*
- **Component generic – Is component member of a group**: *Yes*

When moving the panel, the button will move at the same time.

Later on, it can be useful to set the property **Component generic – Component size and position is locked** to *ON*. I would advise not to do it directly as it makes selection and copy/paste more tricky. (Need to understand the behavior and interest of this property as once a modulator is part of a group it is moving together with the group so somewhere its position is locked…)

The modulator list window

If modulators are part of a group and with their position and size locked, it is not possible to select them directly. To be able to modify their properties and select them individually, select **Panel → Modulator list** from the top menu then click on the modulator you want to modify. The panel properties will show the corresponding properties that you can directly edit as in a table.



Copy/Paste buttons

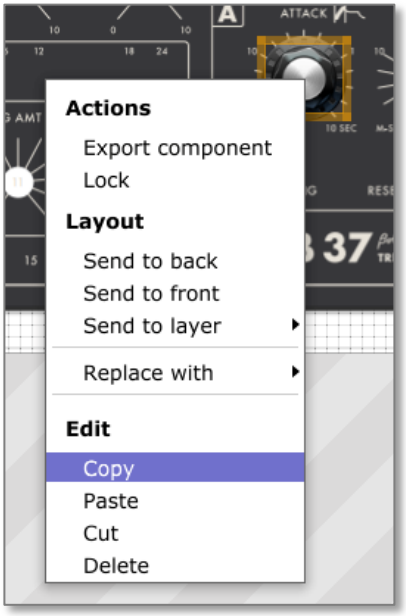
To quicker reproduce buttons, it is possible to use a classical **Copy/Paste**.

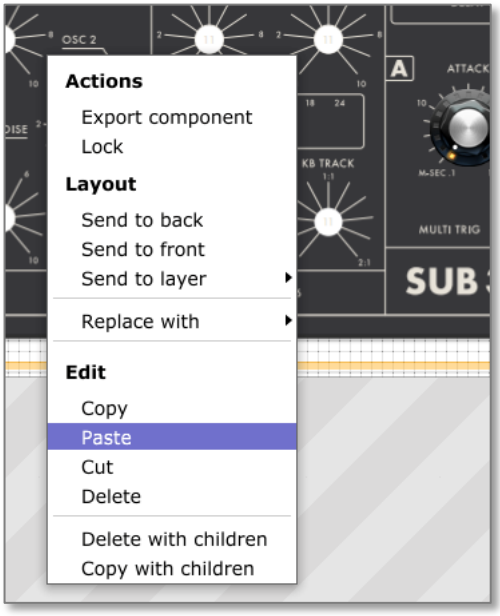
Select your modulator then right click and select **Edit→Copy**.

Select the Tabs modulator then right click and select **Edit→Paste**.

Position them where needed. Position and size is visible in the **Component generic – Position and size** property (xpos, ypos, horizontal size, vertical size).

Don't forget to adapt the **Modulator – Name** and **Component generic – Visible name** properties for each new button! Use different and topic related names.

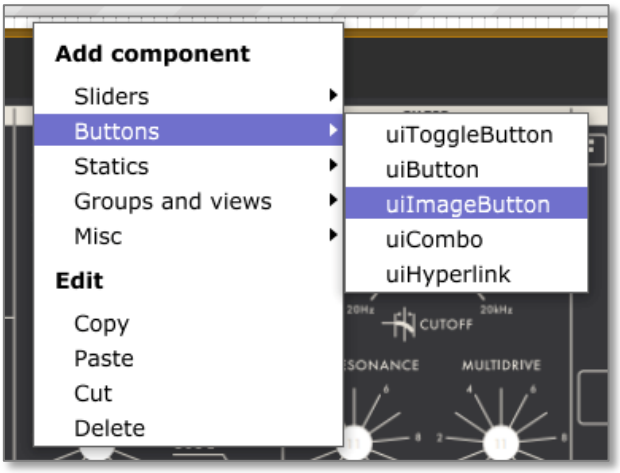




Adding a toggle button

A toggle button is for example a two frames button that will be dark or illuminated at each click.

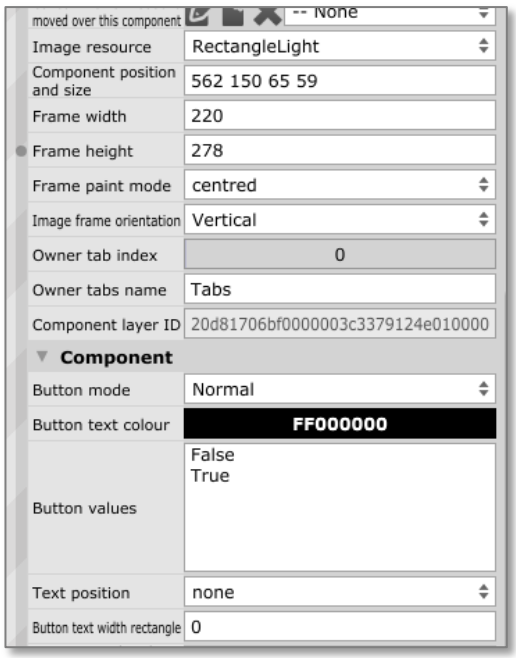
Right click on the panel grid and select **Add component → Buttons → uiImageButton**



As before, set different properties:

- **Modulator – Name**: give a name to the modulator; for example *btnOsc2HardSync*
- **Component generic – Visible name**: *OSC 2 HARD SYNC*. In case of, we will hide it anyway.
- **Component generic – Name label visible**: *OFF*
- **Component generic – Image resource**: *your_toggle_button_image* (for me: *RectangleLight*)
- **Component generic – Frame width**: *220* (the width of each frame)
- **Component generic – Frame height**: *278* (the height of each frame)
- **Component generic – Image frame orientation**: *vertical* (if you assembled your frames vertically)
- **Component – Button mode**: *normal* (switching between two or more frames)
- **Component – Button values**: *False True* (one value by row)

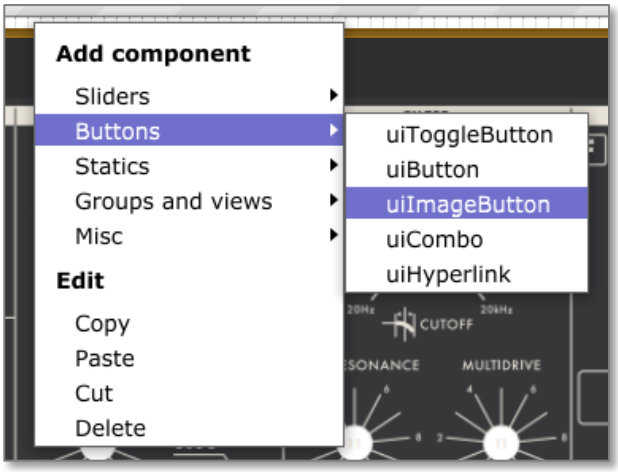
With a *normal* mode button, the button values are cycled row by row and the frames displayed one by one.



Adding a momentary button

A momentary button is for example a one frame button that will cycle through different values each time it is pressed.

Right click on the panel grid and select **Add component → Buttons → uiImageButton**



As before, set different properties:

- **Modulator – Name**: give a name to the modulator; for example *btnMod2Osc*
- **Component generic – Visible name**: *MOD2 Osc1/2 Sel*. In case of, we will hide it anyway.
- **Component generic – Name label visible**: *OFF*
- **Component generic – Image resource**: *your_button_image* (for me: *Rectangle*)
- **Component generic – Frame width**: *183* (the width of the image)
- **Component generic – Frame height**: *115* (the height of the image)
- **Component – Button mode**: *momentary* (works like a push button)

For the values, there are 3 things to consider: the text that will be displayed, the value to test in a Lua code (for some automation, if needed) and the value that is send by Midi.

The text to display (you need to have **Text position** different than “none”) is what’s come on the left side of “=” sign. The value sent by Midi is on the right side of the equal sign. The value to test in your Lua code is the rank of each row starting by 0

On the Moog Sub37, our momentary button indicates the destination of the pitch modulation (Osc1+Osc2, Osc1, Osc2). CC is 88 and Midi values to send are 0, 43 and 85. The destination is indicated on the synth by 2 leds (1 and 2).

So, for **Component – Button values** you need to have (one value by row):
Osc1+Osc2=0
Osc1=43
Osc2=85

A Lua code is used to set the corresponding LEDs ON depending on each button press (see Step 5).

With a *momentary* mode button, the button values are cycled row by row and the button is hidden when pressed without the need of a blank frame.

Step 4 – Sending Midi data

Understanding the Midi specifications of your device

Of course you are building this panel to interact with your device through Midi. Usually all devices' manuals do include a Midi specifications section, more or less well documented.

Typically you will interact through CC (Control Change), NRPN (Non Registered Parameter Numbers) or Sysex (System Exclusive) messages.

Control Change messages (CC) belongs to the Channel Voice family of messages. They are made of 3 bytes with n=channel number (0-15), c=controller number (0-127) and v=controller value (0-127) as:

Status byte	Data byte	Data byte
1011nnnn	0ccccccc	0vvvvvvv

Some controllers may use values 0-16383 instead of 0-127. They will then need 2 messages as above where the CC# of the first message will be 1 to 31 while the CC# of the second message will be 33 to 63. The value will be the combination of the value bytes of the two messages (value byte message 1 = MSB Most Significant Byte; value byte message 2 = LSB Least Significant Byte)

Example1: setting the Amp EG attack time on Midi channel 1 to 43 will require sending B0 1C 2B hex or 10110000 00011100 00101011 (Midi channel 1 has channel number 0)

Example2: setting the Modulation wheel on Midi channel 1 to 1234 will require sending B0 01 04 then B0 21 D2 hex or 10110000 00000001 00000100 then 10110000 00100001 11010010 (value is 00000100 11010010)

Non Registered Parameter Numbers messages (NRPN) are special Control Change subgroups used to control parameters not available by CC messages (there are usually more parameters to tune than the number of available CC's).

To identify the parameter to control it is first needed to use its NRPN number (first LSB in CC 98 then MSB in CC 99) then the value in a third (and fourth if needed) message.

System Exclusive messages (SysEx) allow manufacturers to create their own messages (such as bulk dumps, patch parameters, and other non-specified data).

A System Exclusive message begins with 11110000 (240 decimal F0 hex), followed by the manufacturer ID#, then by an unspecified number of data bytes of any ranges from 0-127) and ends with 11110111 (247 decimal or F7 hex), meaning End of SysEx message.

The manufacturer's Midi ID numbers are available at http://www.midi.org/techspecs/manid.php and http://sequence15.blogspot.be/2008/12/midi-manufacturer-ids.html for example.

For Moog it is 04.

More information on Midi and its messages on the internet at sites like
http://www.midi.org/index.php
http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI.shtml

Preparing modulators for sending a CC midi message

For the Moog Sub37, there is a parameter in the Midi Menu to select 7 bits (0-127) or 14 bits (0-16383) communication and it is the reason I'm explaining both.

Let's work with the Amplifier Envelope Generator:

- Attack time: CC# 28, 0-127 or 16383
- Decay time: CC# 29, 0-127 or 16383
- Sustain time: CC# 30, 0-127 or 16383
- Release time: CC# 31, 0-127 or 16383

Select each corresponding rotary button created in the Step 3 then set the following properties:

- **Component – Maximum value**: *127*
- **MIDI – MIDI message type**: *CC*
- **MIDI – MIDI Channel**: *1*
- **MIDI – MIDI controller number**: *28, 29, 30* and *31* for each button respectively (Ctrlr accepts the values in decimal)

Should you need 0-16383 values then this becomes:

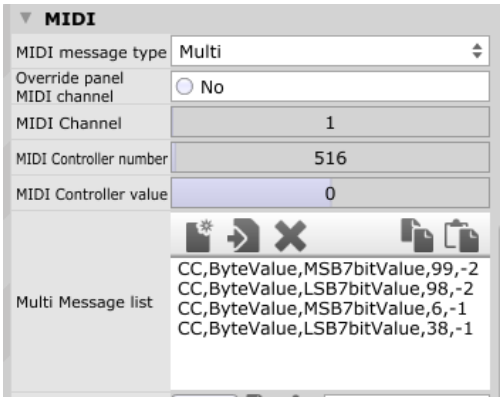
- **Component – Maximum value**: *16383*
- **MIDI – MIDI message type**: *Multi*
- **MIDI – MIDI Channel**: *1*
- **MIDI – MIDI controller number**: *28, 29, 30* and *31* for each button respectively (Ctrlr accepts the values in decimal)
- **MIDI – Multi Message list:** CC,ByteValue,MSB7bitValue,28,-1:CC,ByteValue,LSB7bitValue,60,-1

Preparing modulators for sending a NRPN midi message

With the Sub37, there are a lot of parameters and it is thus not possible to assign a standard MIDI message to every parameter. Therefore NRPN messages can also be used.

To use NRPN messages:

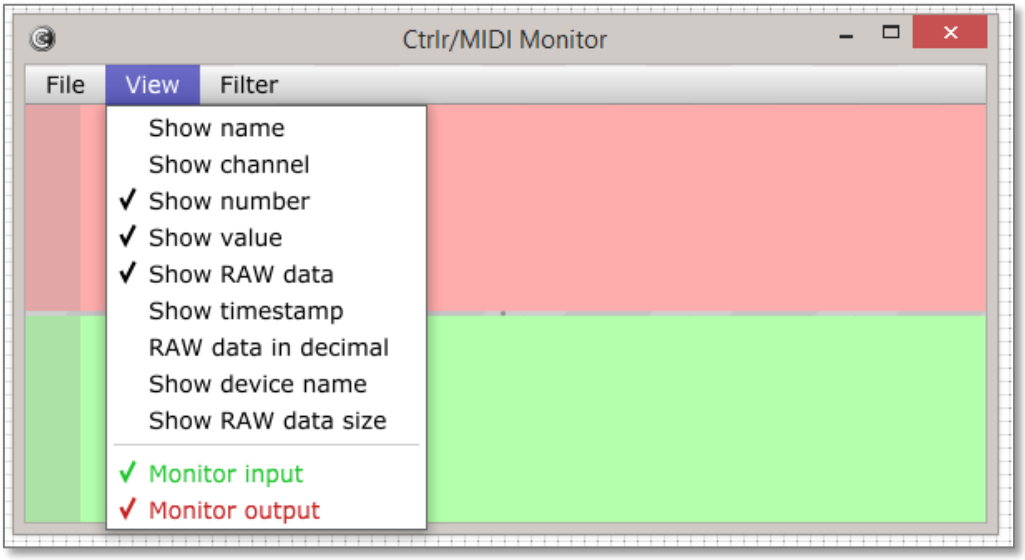
- **Component – Button values**: *0 1* (2 lines – case for a 2 values switch)
- **MIDI – MIDI message type**: *Multi*
- **MIDI – MIDI Channel**: *1*
- **MIDI – MIDI controller number**: the NRPN number
- **MIDI – Multi Message list:** CC,ByteValue,MSB7bitValue,99,-2:CC,ByteValue,LSB7bitValue,98,-2:CC,ByteValue,MSB7bitValue,6,-1:CC,ByteValue,LSB7bitValue,38,-1



The MIDI monitor

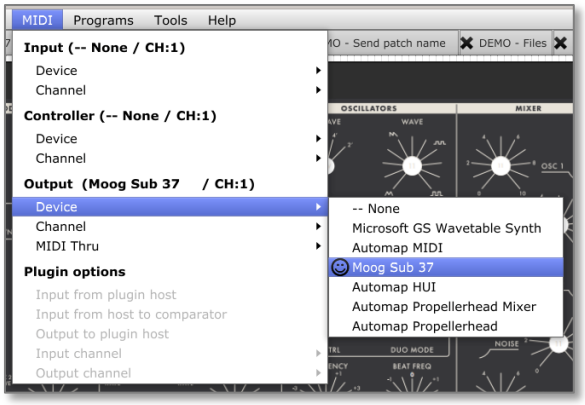
To analyze the Midi inputs/outputs it is valuable to display the Midi monitor available in the Ctrlr menu **Tools→MIDI Monitor**.

Under **View**, select what data you would like to monitor (Midi channel, Device, CC number, value…) and enable/disable either Midi output (red – from Ctrlr to your device) or Midi input (green – from your device to Ctrlr).



Setting Ctrlr to send data and testing

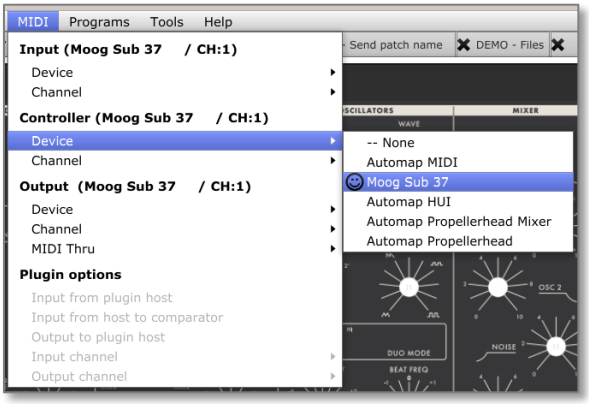
Start your device then select it and its MIDI channel with the Ctrlr menu **MIDI→Output→Device** and **MIDI→Output→Channel**. A green message will be displayed at the bottom of Ctrlr if the connection is established and this will be confirmed by a happy face in the menu as shown below.



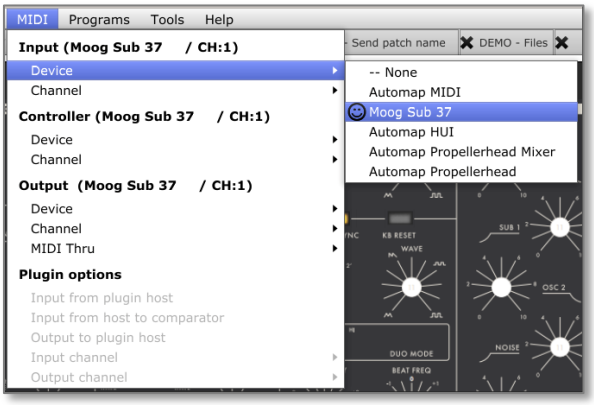
To have Ctrlr controlling your device, you must also set your device in the **Controller** section (need to understand why – read different things on the forum but not enough to be sure of what to explain)

http://ctrlr.org/forums/topic/midi-device-settings-controllerin-device/

For now, one of the Ctrlr users "Stoner" added this code to Ctrlr so that the Controller device does the same thing that the Input device does. MIDI In/Out will be used for communicating with the device the panel is created for, while the Controller device will be used to map "custom" MIDI events on the Controller device to modulators/components in the panel, it will come with a "MIDI Learn" function

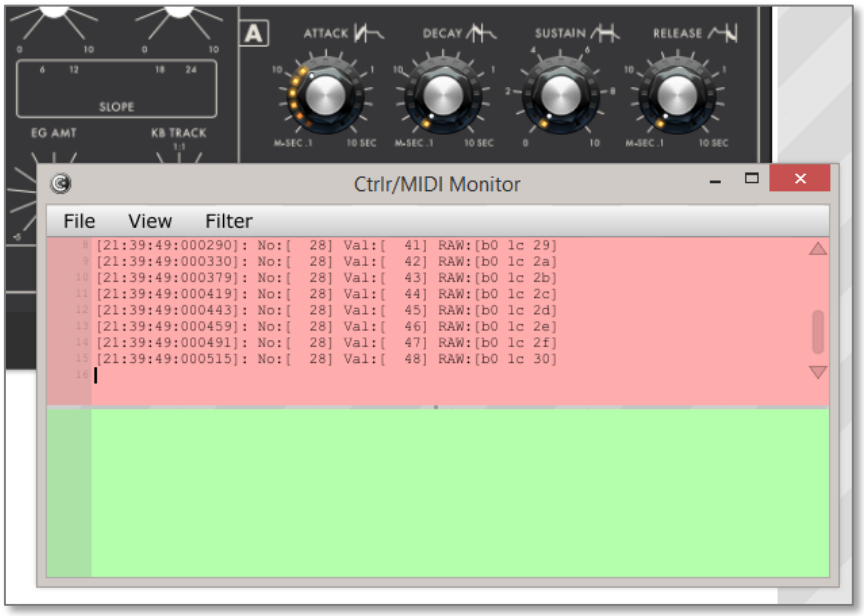


As you are busy and if wished/needed you do the same for the **Input** section.



Switch panel mode to leave the Editing mode with **Panel → Panel Mode**

Select the Attack time rotary button and move it. Messages should be sent and be visible in the MIDI monitor window:



You can also perform this test without an actual hardware device **but** a Midi output device and channel **must** be selected.

Of course, you can now check the sound of your synthesizer. It should have a different attack…

On the Sub37, there is an illuminated Midi switch that turns red when the synthesizer gets data.

Changing the attack on the device itself should trigger the display of the same kind of messages (in our case CC28, B0 1C *value*) in the green part of the Midi monitor window.

It is needed to create a Lua method (event handler) to have your Ctrlr panel buttons updated based on the value received. **This will be explained later…**

Step 5 – Introduction to automation with Lua

The panels can get some automation by using a script language called **Lua**. This can be used for even more complex developments like having a librarian, having graphs for envelope generators…

The goal of this section is just to give you an idea of some possibility. It will be developed more extensively in a coming version of this guide.

What is Lua?

From the lua.org web site, one can read:

Lua is a powerful and fast programming language that is easy to learn and use and to embed into your application. Lua is designed to be a lightweight embeddable scripting language and is used for all sorts of applications from games to web applications and image processing.

"Lua" (pronounced **LOO-ah**) means "Moon" in Portuguese. As such, it is neither an acronym nor an abbreviation, but a noun. More specifically, "Lua" is a name, the name of the Earth's moon and the name of the language. Like most names, it should be written in lower case with an initial capital, that is, "Lua". Please do not write it as "LUA", which is both ugly and confusing, because then it becomes an acronym with different meanings for different people.

Lua references

The bible: http://www.lua.org/home.html

Reference manual: http://www.lua.org/manual/5.3/

Some automation

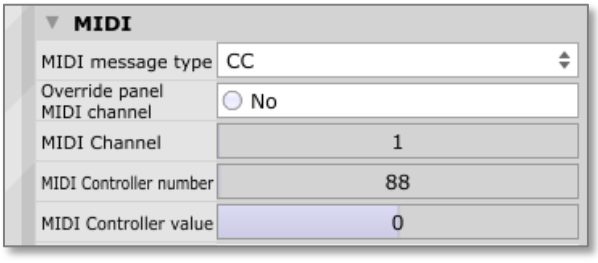
In the Step 4, we created a momentary button *MOD2 Osc1/2 Sel* with **Component – Button values** as (one value by row):
Osc1+Osc2=0
Osc1=43
Osc2=85

We want that, when clicking on the button, the value (0, 43 or 85) must be sent to the synth (CC 88) and that the corresponding LED are toggled ON or OFF accordingly (as on the hardware synth). If this is happening then we are good ☺

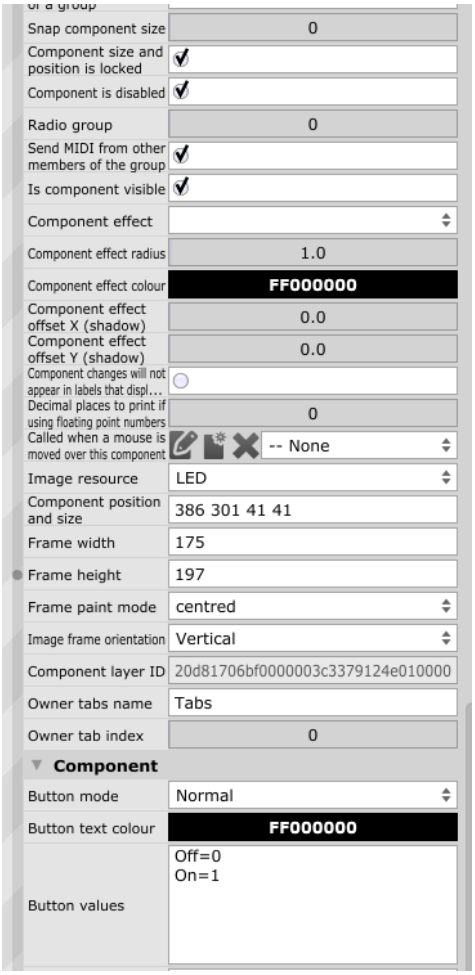
Step by step

Check that the *MOD2 Osc1/2 Sel* button has its Midi properties set as:

- **MIDI – MIDI message type**: *CC*
- **MIDI – MIDI Channel**: *1*
- **MIDI – MIDI controller number**: *88* (this is of course specific to our example)



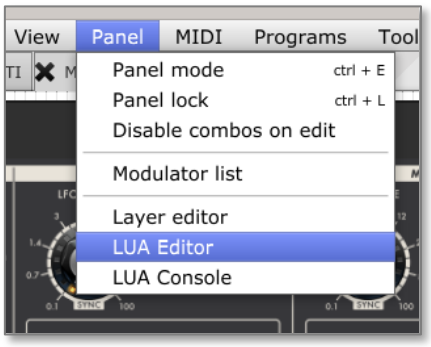
Create the LED for Osc1 and Osc2 with uiImageButton components named *ledMod2Osc1Pitch* and *ledMod2Osc2Pitch* as described in Step 4 (toggle button, a led image with 2 frames – off and on, value 0 and 1).

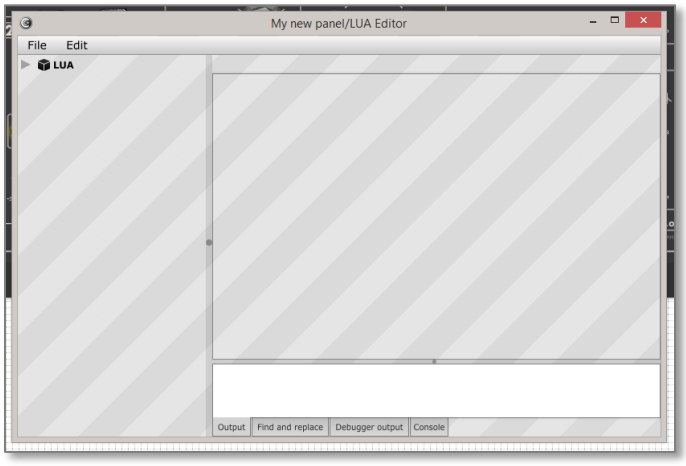


As leds shouldn't be activated by the user, it is good to set the following properties to ON:

– **Component generic – Component size and position is locked**: *ON*
– **Component generic – Component is disabled**: *ON*

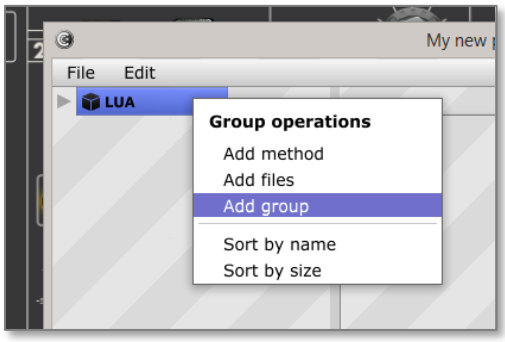
Select **Panel →Lua Editor** in the Ctrlr menu to open the window where you will be able to organize and edit your Lua programs.

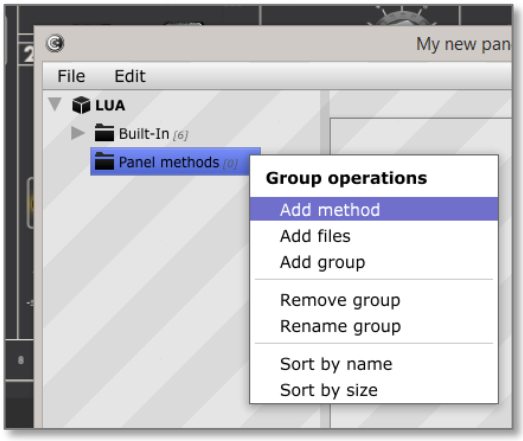




It is good to directly take the habit to well organize your programs. Right-click on the LUA folder and select **Add group** to create a new folder called *Panel methods* (for example).

Select the *Panel methods* folder and select **Add method** to add a new empty method called *Mod2OscSelect*





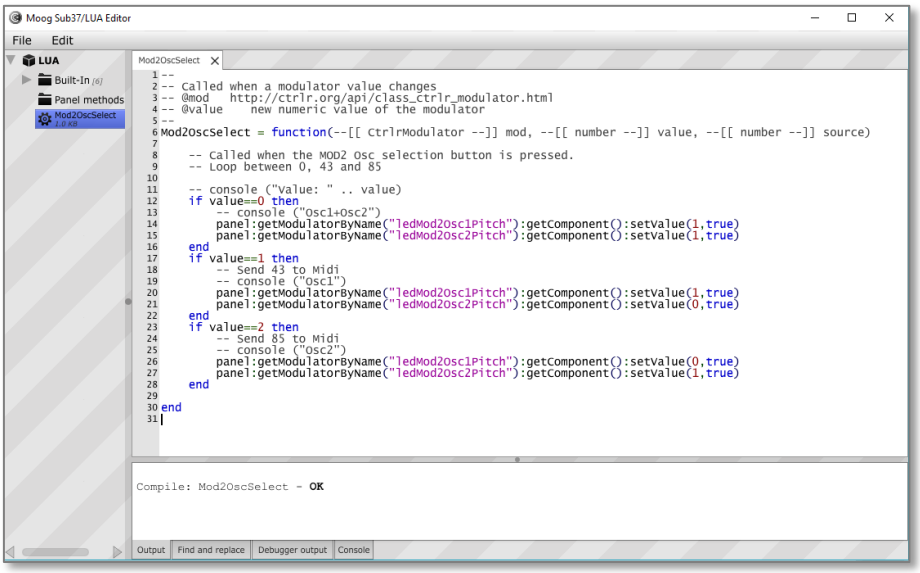
Select the *MOD2OscSelect* method then by double clicking display it in the editor window. Copy the following code then paste it by replacing the existing empty function.

```
--
-- Called when a modulator value changes
-- @mod   http://ctrlr.org/api/class_ctrlr_modulator.html
-- @value    new numeric value of the modulator
--
Mod2OscSelect = function(--[[ CtrlrModulator --]] mod, --[[ number --]] value, --[[ number --]] source)
	-- Called when the MOD2 Osc selection button is pressed.
	-- Loop between 0, 43 and 85
	-- console ("Value: " .. value)
	if value==0 then
		-- console ("Osc1+Osc2")
	panel:getModulatorByName("ledMod2Osc1Pitch"):getComponent():setValue(1,true)
	panel:getModulatorByName("ledMod2Osc2Pitch"):getComponent():setValue(1,true)
	end
	if value==1 then
		-- Send 43 to Midi
		-- console ("Osc1")
	panel:getModulatorByName("ledMod2Osc1Pitch"):getComponent():setValue(1,true)
	panel:getModulatorByName("ledMod2Osc2Pitch"):getComponent():setValue(0,true)
	end
	if value==2 then
		-- Send 85 to Midi
		-- console ("Osc2")
	panel:getModulatorByName("ledMod2Osc1Pitch"):getComponent():setValue(0,true)
	panel:getModulatorByName("ledMod2Osc2Pitch"):getComponent():setValue(1,true)
	end
```

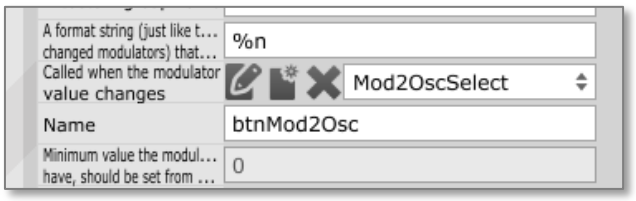
```
end
```

As you can see, the value of the modulator to test is the row number with the first row being 0. Depending on the value, the two leds are set On or Off. It is the setting of the modulator and the Midi CC settings that secures sending the corresponding Midi value 0, 43 or 85.

Save and compile your Lua method by selecting **File→Save and compile**. It should look like this:



Select the button and select the *Mod2OscSelect* method in the pulldown of the **Modulator – Called when the modulator value changes** property



Appendix A

Version history

Date	Version	Description	By
2015-07-01	1.0	Initial document. Only headers in the Librarian part.	goodweather
2015-07-04	1.1	Added Save and Export description. Explanation of first 2 digits in colors (transparency). Midi Controller and Midi Input. Popup bubble when dragging. Lua and not LUA.	goodweather Puppeteer
2015-08-23	1.2	More details on button images and toggle button. Momentary button explained. Step 5 "Librarian" with only a structure replaced by Step 5 "Introduction to automation" with Lua fully documented.	goodweather
2016-03-31	1.3	Corrections in Lua code (beginners' mistakes)	goodweather
2016-??-??	2.0	Will be completely re-written with more examples and Lua codes. Will be mainly based on DSI Pro2, Moog Sub37 and a generic one	goodweather

Things I didn't find

- How to make the bubble properties visible for uiSlider, uiImageSlider?
- Is it possible to refresh the list of MIDI devices or is it necessary to quit and restart Ctrlr to have the list refreshed when a new synth or controller is connected / switched ON?
- How to keep the MIDI monitor window on top while moving buttons in a panel?

Identified things to add in this manual

In this section I will list things that I identified to add and that I didn't have yet time to describe or that have not been used for the Moog Sub37 panel.

So:

- Grouping modulators
- Example of nice style buttons
- A few nice basic Lua programs
- Envelope graphs display
- Description and usage of each component
- Properties of each component
- Useful links chapter (Lua, Midi, Hexadecimal…)
- Oscilloscope (is this really possible to achieve?)

Identified possible bugs in v5.3.94

- uiTabs background image layout: centred → cent**e**red
- uiTabs "Content background **c**olour"
- Everywhere: color or colour (US vs British English ☺)
- uiTabs – Tabs orientation: TabsAtLef**t**
- The current tab label disappears sometimes (is this a bug?). My impression is that it is after some button copy/paste. I will try to identify better next time this will occur but I got it already a few times.
- Naming agreement: bubble or tooltip? At modulator level properties are referring to Bubble while at Panel level it is Tooltip. Is this the same object or something different?

- Bubble: Name text justification
- Change LUA to Lua in Ctrlr menu and all other places

Possible enhancements related to the topics treated in this manual

- uiTabs property panel to be refreshed each time that a tab is added by pressing the Add tab button. For the moment, it is needed to click on the panel then to reselect the uiTabs component to see the adapted properties
- Move the Add tab button at the top of the Properties panel
- Move the Remove tab at the beginning of the properties section related to each tab
- uiTabs content background color to be refreshed directly when the content background color is changed. For the moment, it is needed to click on the panel then to reselect the uiTabs component to see the new color applied

Appendix B

Ctrlr methods documentation

Here is some valuable info if you want to know more about the available methods in Ctrlr. I only found Atom's post recently and I copy the text as-is from the forum because it is an important reference.

http://ctrlr.org/forums/topic/documentation-functions-links/

There are a few ways to find out the list of Lua methods and functions.

Look at the bottom of:

- for panel methods: http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaPanel.cpp
- for modulator methods: http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaModulator.cpp
- for object methods (any modulator, panel, component is an object): http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaObject.cpp
- for timer methods: http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaMultiTimer.cpp
- for utility methods: http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaUtils.cpp

For the panel editor, canvas,layer:

- http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaPanelEditor.cpp
- http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaPanelCanvas.cpp
- http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/CtrlrLuaPanelCanvasLayer.cpp

For each object defined you can look at and list its methods for example in the Lua Console:

```
mod = panel:getModulatorByName("modulator1")
com = mod:getComponent()
what (mod) -- will print all the methods for the modulator
what (com) -- will print all the methods for the component
```

To get a list of ALL classes bound to Lua, use the how() method:

```
how()
```

The documentation for classes that do not start with Ctrlr prefix is on the Juce website: http://rawmaterialsoftware.com/juce/api/classes.html

Always check if all methods are available. You can do that by looking at the class file in the http://sourceforge.net/p/ctrlrv4/code/1511/tree/nightly/Source/Lua/JuceClasses/ directory (each file .cpp/.h file pair represents one class and you can see what methods are bound to Lua and how).