Native Instrument Komplete Kontrol MK2

MC Custom Script

Preinstallation Requirements and Instructions

There are two options in this script, the **FULL** and the **Stripped-Down** one.

In the **Stripped-Down** option, the following is missing:

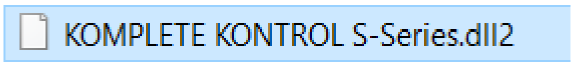
- No browsing of instruments and effects plugins available via the Shift + Jog Wheel turns. This is replaced by the default Cubase's presets browsing functionality.

Apart from the above, all other functions are the same.

WARNING!

You really have to disable by renaming or moving to another folder, the file C:\Program Files\Common Files\Steinberg\Shared Components\KOMPLETE KONTROL S-Series.dll which includes the original Native Instrument's Cubase implementation, upon testing my script. Otherwise, there will be crashes, conflicts, and erratic controller's behavior.

I have just renamed this file by adding the number 2 after the dll extension and it's all good.

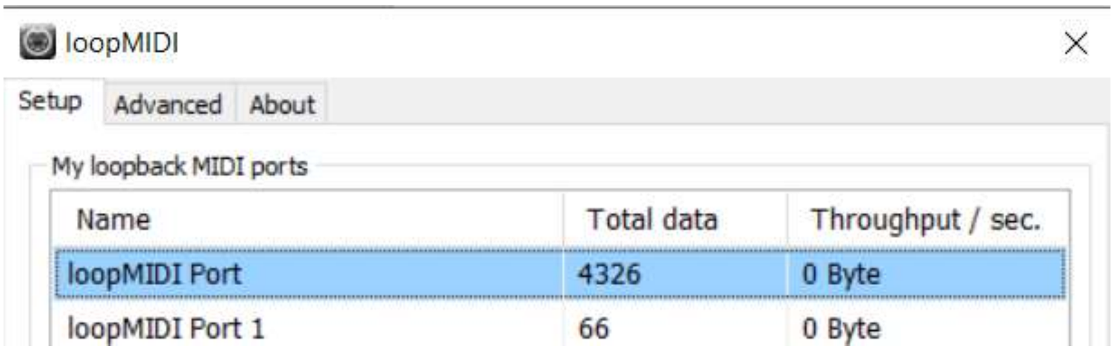

KOMPLETE KONTROL S-Series.dll2

Note that you can always revert to the original NI's Cubase integration by renaming this file to its original name and by disabling my script.

[Applies to the FULL implementation. If you're going to use the stripped-down version, please skip this step]

If you're going to use the full implementation of this script, you will need 2 virtual MIDI ports (apart from the default ports of our Komplete Kontrol).

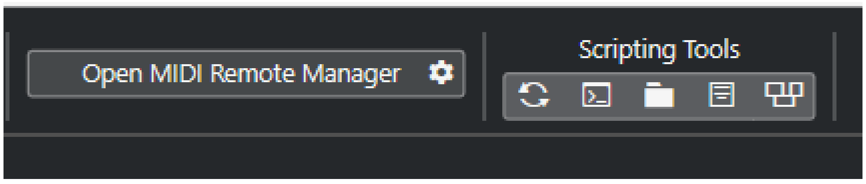
For this, I'm using **loopMIDI** (<u>you can obviously use other virtual ports if you have</u>) and my ports are setup like this:



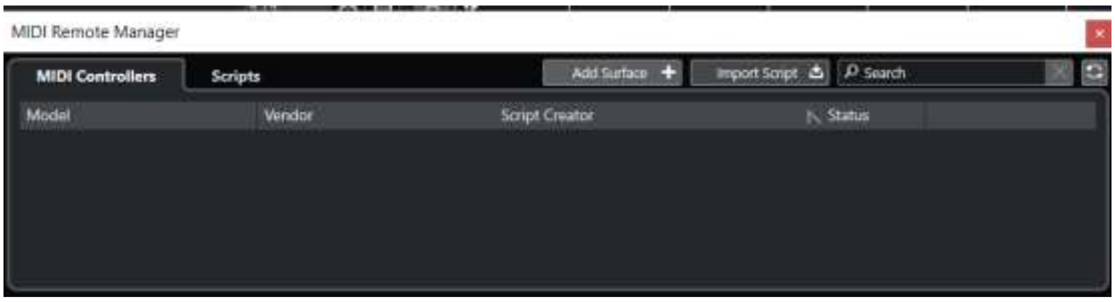
Name	Total data	Throughput / sec.
loopMIDI Port	4326	0 Byte
loopMIDI Port 1	66	0 Byte

[Applies to BOTH implementations]

Open the **MIDI Remote TAB** in Cubase and then click on **Open MIDI Remote Manager**:



In the Midi Remote Manager window, click on **Import Script:**



In the File Dialog, choose the script's file you've downloaded, and Cubase will install the script and its components. **The next step is to make some adjustments <u>when needed</u>.**

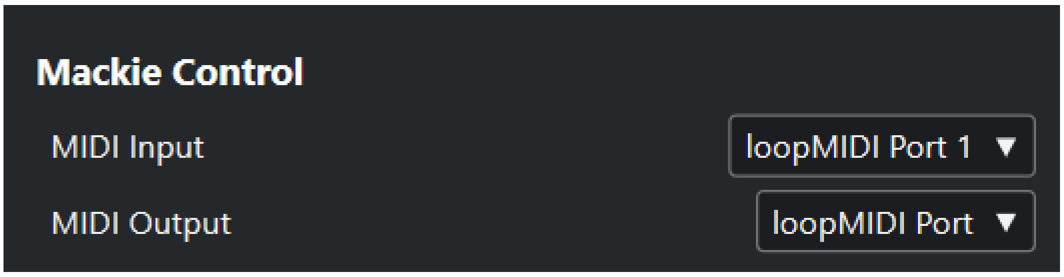
[Applies to the FULL implementation. If you're going to use the stripped-down version, please skip this step]

If we do want to use the full implementation's functionalities, there is another step involved, apart from setting **generalSettings.strippedDown =0** in the **mapOfGeneralSettings.js** file.

Setup for allowing changing plugins using our controller:

Open **Cubase's Studio->Studio Setup**, click on **Add Device** and select **Mackie Control**.

In the ports of this new item, assign **loopMIDI Port 1** and **loopMIDI port** respectively, as shown in the screenshot below, and click **Apply**.



Mackie Control

MIDI Input	loopMIDI Port 1
MIDI Output	loopMIDI Port

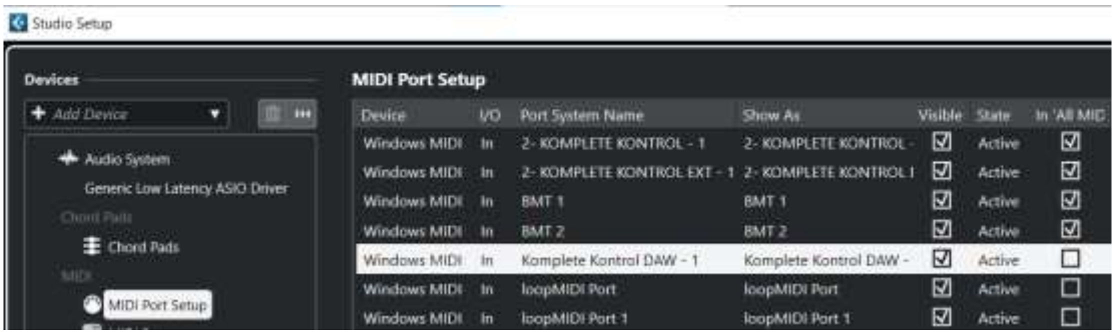
[Applies to BOTH implementations]

SETUP

MIDI ports setup

From inside Cubase, go to menu **Studio-> Studio Setup → Tab Midi Port Setup** and **uncheck** the entry **Komplete Kontrol DAW – 1** (this is its default port for DAW control) from the "**In 'ALL MIDI inputs'**" column.

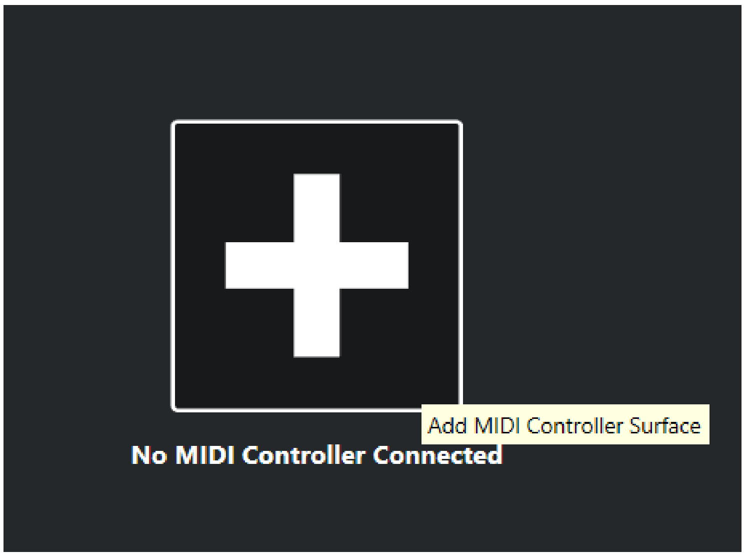
If using the full implementation, please uncheck the following ports too: **loopMIDI Port** and **loopMIDI Port 1**.



MIDI Port Setup

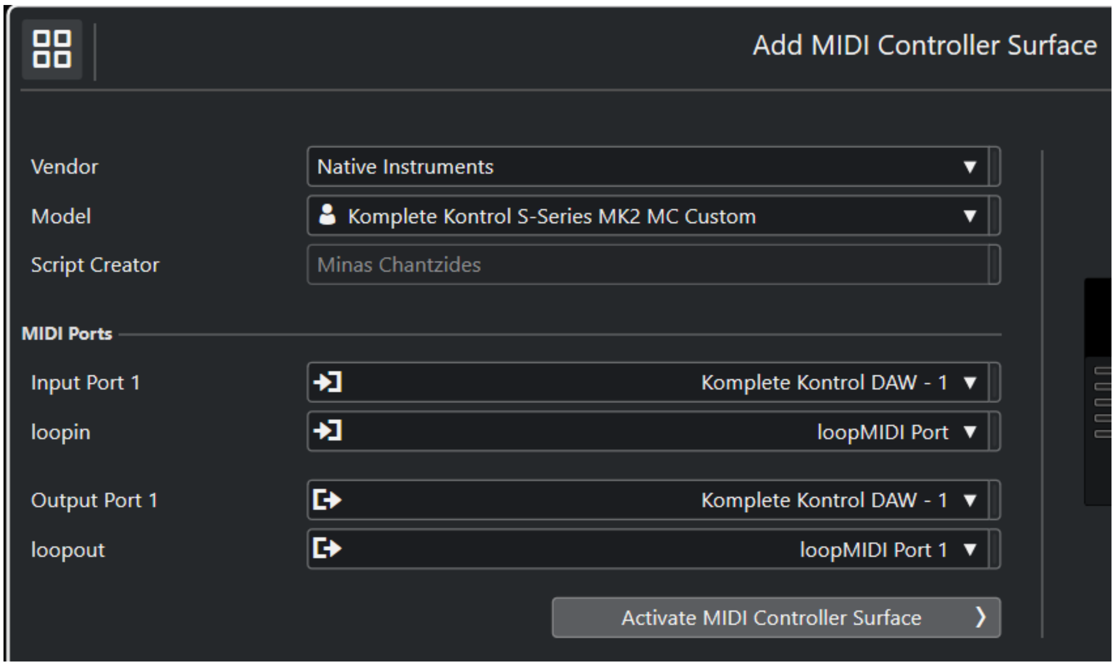
Device	I/O	Port System Name	Show As	Visible	State	In 'All MIC
Windows MIDI	In	2- KOMPLETE KONTROL - 1	2- KOMPLETE KONTROL -	✓	Active	✓
Windows MIDI	In	2- KOMPLETE KONTROL EXT - 1	2- KOMPLETE KONTROL I	✓	Active	✓
Windows MIDI	In	BMT 1	BMT 1	✓	Active	✓
Windows MIDI	In	BMT 2	BMT 2	✓	Active	✓
Windows MIDI	In	Komplete Kontrol DAW - 1	Komplete Kontrol DAW -	✓	Active	☐
Windows MIDI	In	loopMIDI Port	loopMIDI Port	✓	Active	☐
Windows MIDI	In	loopMIDI Port 1	loopMIDI Port 1	✓	Active	☐

Go back to Cubase and after selecting the **MIDI Editor Tab** click on the big **PLUS** button:



Fill the form as follows and click on the **Activate MIDI Controller Surface** button.

If you're going to use the **full implementation**, you must fill in the form as follows:


Add MIDI Controller Surface

Vendor	Native Instruments
Model	Komplete Kontrol S-Series MK2 MC Custom
Script Creator	Minas Chantzides

MIDI Ports

Input Port 1	Komplete Kontrol DAW - 1
loopin	loopMIDI Port
Output Port 1	Komplete Kontrol DAW - 1
loopout	loopMIDI Port 1

Note (again) that this is the full implementation of the script. In this one we are using apart from the Komplete Kontrol's DAW Midi Ports, another one set, created by using the loopMIDI utility, available from Tobias Erichsen's website here: https://www.tobias-erichsen.de/software/loopmidi.html .

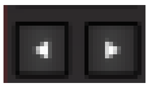
If you have selected the **stripped-down version (this is the default)**, Cubase most probably will immediately recognize the connected device and set it up properly. However, in case it does not, you must follow the previous screenshot, but this time, you have to simply select the "**Komplete Kontrol Daw-1**" ports, the other two will not be visible.

Pages

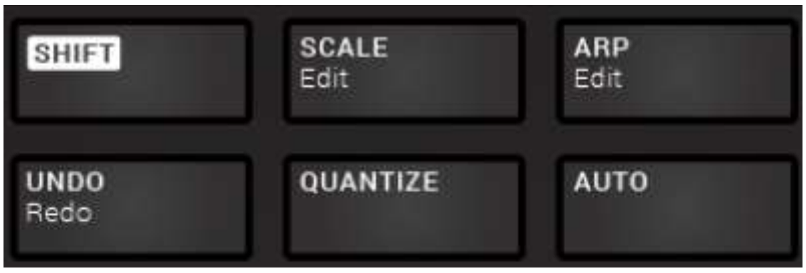
The script creates the following pages:

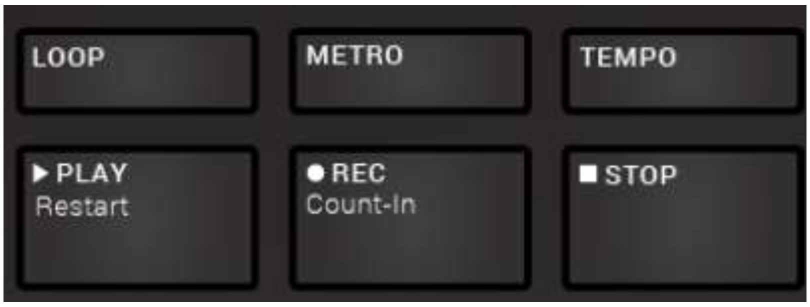
- **Mixer**
- **Focused Quick Controls**
- **Quick Controls (This refers to the instrument's controls)**
- **Channel Strip (EQ, Input filter, Gate, Compressor, Tools, Saturation and Limiter)**
- **Insert Effects**
- **Send Effects**
- **Commands Set 1**
- **Commands Set 2**
- **Commands Set 3**

We can navigate through the various pages using the **Left** and **Right** arrow buttons of the Komplete Kontrol:



Transport Buttons Section





Here we have the usual assignments, i.e., Undo (and Redo), Quantize, Auto Write, Loop, Metro, Play (and Restart), Rec (and Count-In) and Stop.

However, in this script, I've made some additions, based on **double-click** functionality.

So, if we double click

- Quantize: We turn on/off the Auto Quantize mode

- Auto: We turn on/off the Auto Read mode

- Loop: We set Punch-In/Out locators, based on the direction we previously turned our jog wheel (See **Note about actions based on jog-turn direction**)

- Metro: we can change the metronome's click level by turning the jog wheel. After 2 seconds of inactivity (i.e., no longer turning the jog wheel after Metro's double click, jog wheel gets back to its original assignment)

- Play: We toggle Punch-In

- Play (while holding Shift): We toggle Punch-Out

- Clear: We cut head/tail of the selected event, based on the direction we previously turned our jog wheel (See **Note about actions based on jog-turn direction**)

- Jog: We show/hide our plugin's window

  Jog (while holding Shift): We show/hide our track's channel settings' window.

The Jog Encoder Section

The Jog Encoder can be of great use, since we can literally have 13 assignments out-of-the box, plus 2 implemented in this script by using double push.

So, we have:

nudge up, nudge down,

nudge left, nudge right,

turn left, turn left with Shift held,

turn left in state2 (that is when we previously pressed Shift + nudge up)

turn left with Shift held in state 2,

turn right, turn right with Shift held,

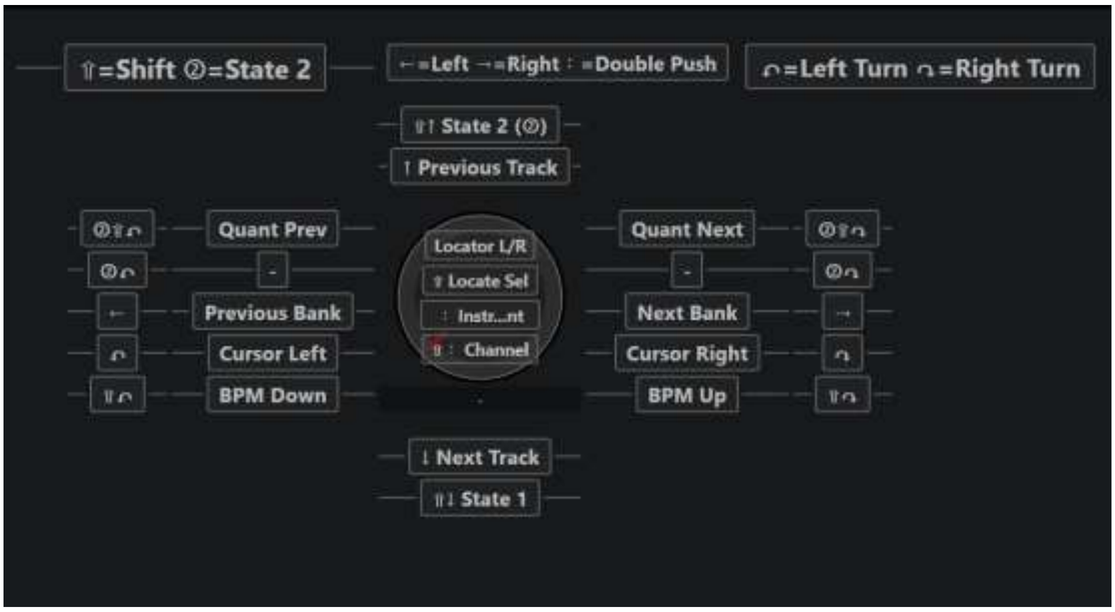
turn right in state2 (that is when we previously pressed Shift + nudge up)

turn right with Shift held in state 2,

push, Shift + push, and finally double push and Shift + double push.

For each of these actions, I have made a graphical representation in the MIDI Remote UI's window.

Let's see how these actions are represented in our mixer page:



At the very first line, we see the notation followed in the next lines. This line tells us the characters used in the other lines to denote a “state”.

The ⇧=Shift tells us that we have to use the Shift button when we meet the ⇧ sign.

The next one, ②=State 2 tells us that we have to first turn our jog to state 2 before executing the action we want to. State 2 is turned on by holding Shift and nudging our jog up. State 1 is turned on by holding Shift and nudging our jog down.

The other symbols

←=Left →=Right ⁚ =Double Push ↶=Left Turn ↷=Right Turn

refer to the actions that can be taken when we nudge left/right, double push or turn left/right our jog.

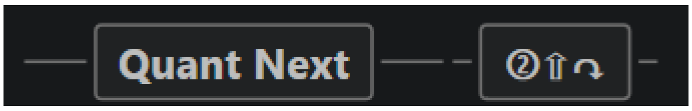
Now, let’s see two examples:

1. The sequence ②⇧↶

   Remembering our previous definitions, this means: Enable State 2, hold Shift and Turn Left.

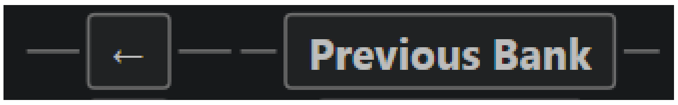
2. ⇧ ⁚ stands for: Hold Shift and Double Click (Push) the jog.

Hope it now makes sense. Example:



This means that if we go to state 2 (by holding Shift and then nudging up) and then hold Shift and turn our jog right, we select the next quantize preset.

Another (simple this time) example:



This means that if we nudge left, we select the previous bank (be it a mixer bank or a plug-in quick controls bank etc.)

I'm using this way of representing actions for each page of my script.

With a little bit of playing around with the assignments, I really think that most of them – if not all - will come out naturally while moving/pushing the jog.

Note that, in each page, there may be different assignments to the jog functions, so at least at the beginning, it's advised that you read them on our midi remote's UI.

Note about actions based on jog-turn direction

A short note about how the assignment Locator L/R to the simple push, works:

When we turn our jog left or right, the script memorizes the direction. So, if we now **push** our jog down, depending on the direction, the script will set our left or right locator to the current cursor's position.

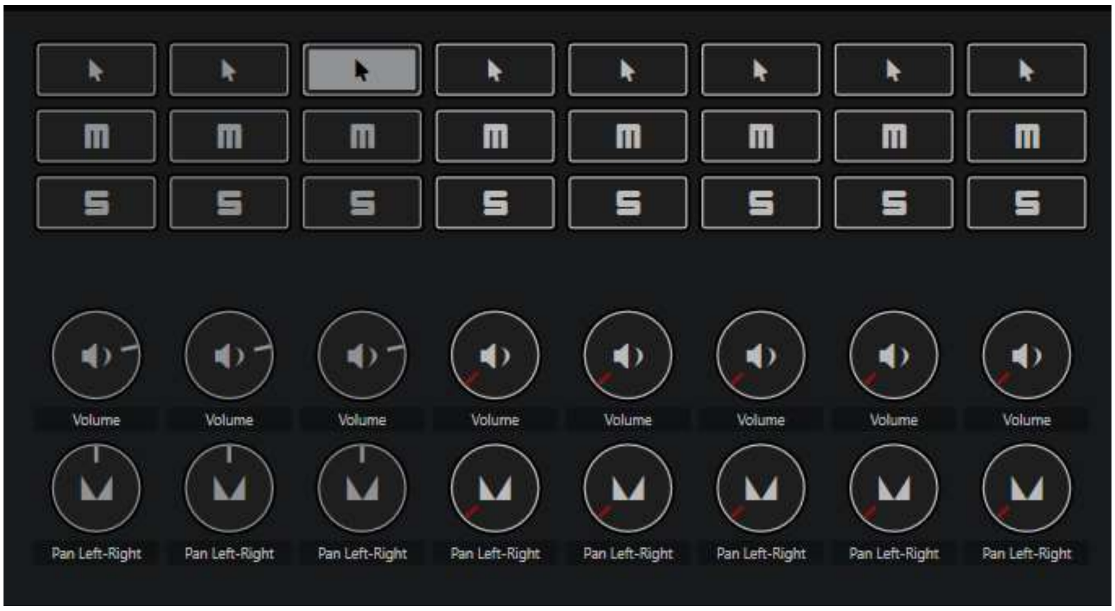
The same method is used when we double click **Loop** and **Clear** buttons:

When we double click **Loop**, based on the direction saved, the script sets our **Punch-In** (left) or **Punch-Out** (right) locator to the current cursor's position.

When we double click **Clear**, again based on the direction saved, the script performs either **Cut Head** (left) or **Cut Tail** (right).

The “Main” Knobs and Button Section

I’ve tried to show in the UI all the assignments made per mapping page. For example, let’s have a look at our Mixer page:



Here, we can see some differences from our real controller’s surface. So, instead of one, we have three buttons for each column, and two knobs instead of the real one.

Buttons:

The very first row of buttons stands for the assignments made to our buttons.

The second row is for the assignments made to our buttons with the Mute button pressed.

The third row is for the assignments made to our buttons with the Solo button pressed.

So, in the mixer page, we see the (default in the NI’s original implementation as well) Select, Mute and Solo assignments for each of the eight tracks of the currently selected mixer bank.

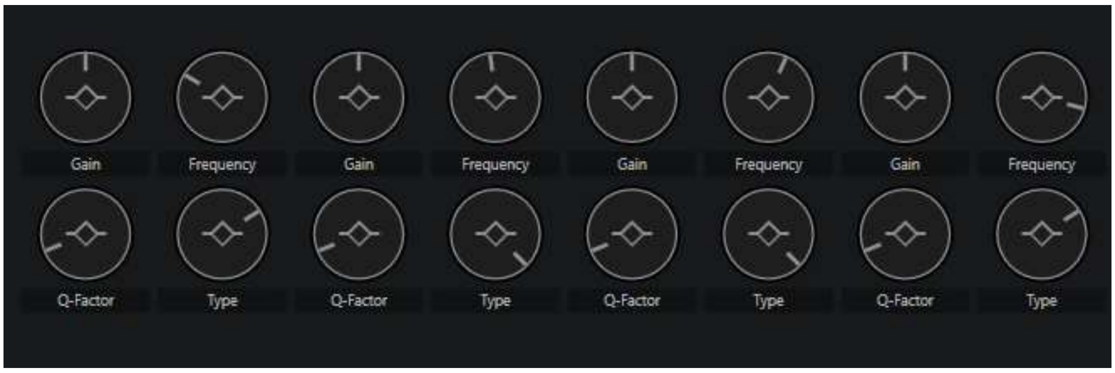
Knobs:

The first row is for the assignments made to our knobs,

The second row is for the assignments made to our knobs when we have set our controller to its second state (**State 2**). This second state, as we know from the original NI’s implementation, is activated by holding **Shift** and nudging our **jog up**. Returning to the first (initial) state is done by holding **Shift** and nudging our **jog down**.

So, in the Mixer page, in state 2 as we see in the screenshot, this second row refers to the pan attribute of our tracks.

Let’s see another example, from the EQ subpage of the Channel Strip Page:



Here, the first row of our knobs is for controlling Gain and Frequency of our four EQ bands.

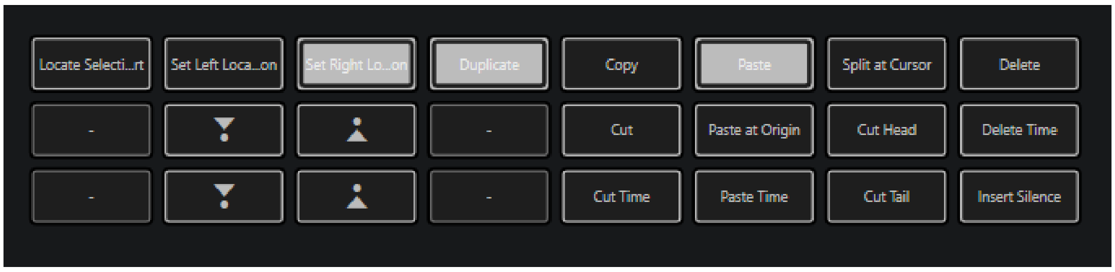
The second row is for controlling the Q-Factor and Type of these bands.

Specifically in the case of this subpage, I have another set of assignments for the Q-Factor and Type, without the need for moving to State 2.

If we hold Mute and click on On/Off button, we can change Q-Factor by using our knob. We can return to controlling Gain by holding Mute and clicking on On/Off again.

Furthermore, we can select Type by pressing the corresponding buttons. The Type value is increased with each press.

Another example, this time from the first Commands Page:

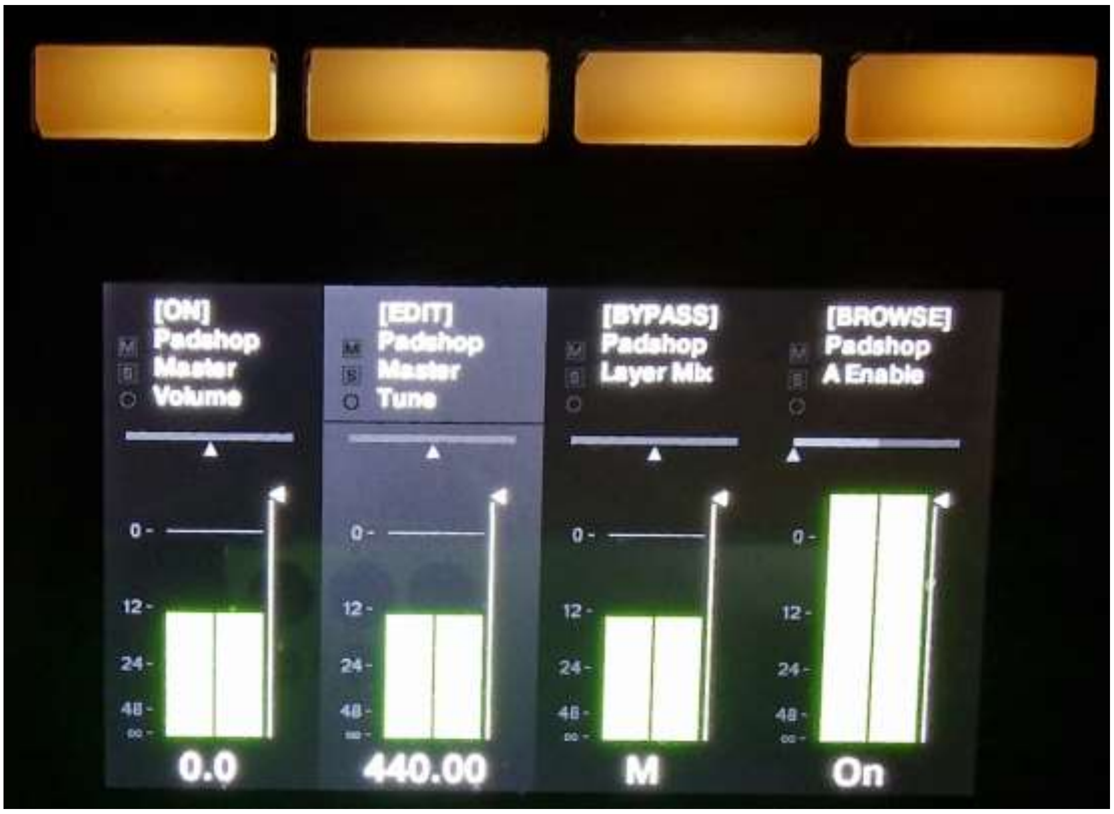


Here we see different assignments for our **Select**, **Mute** and **Solo** combinations.

For example, by holding **Mute** and pressing the fifth **button**, performs **Cut**, while by holding **Solo** and pressing the same **button**, performs **Cut Time**.

Finally, please note that whenever we see text in brackets right below from our buttons, this is to denote the assignment for this button. This is **not** showing on the UI, **but** on our controller's display.

Here's an example from our Instrument's Quick Controls' page:



We see the ON, EDIT, BYPASS and BROWSE functions, all in brackets to denote that they are assigned to our buttons.

General Settings

In the file **mapOfGeneralSettings.js**, we have the general settings which are used by the script.

You can modify any entry you wish.

I’ve included comments to show what every setting does. You can alter their values to 0 or 1, based on your preference.

Some of the functionalities described earlier, can be disabled/re-enabled by changing the corresponding variables’ values (from 1 to 0 for disabling, or 1 from 0 for re-enabling).

Commands Sets

As written earlier, I have three pages of commands’ sets. You can alter the commands bound, by editing the file **mapOfCommands.js**. Please follow the instructions provided in the file’s comments.

<u>Since this is a script which was built based on my personal workflow, I have setup some Macros and Logical Editor Presets, that are used in these command sets, as well as factory commands.</u>

You can obviously edit these entries and setup your very own.

However, if you want these commands, you have to add them manually, here’s how:

Inside the scripts folder, I have placed two subfolders **Logical_editor_presets** and MIDI_Logical_editor_presets, containing the logical presets xml files, I use in my implementation. Feel free to import these to your own presets folder if you like them and/or wish to alter them.

But, of course, you then have to create the corresponding macros. This means that you have to manually edit your commands xml file unfortunately, but it surely can be done with not too much effort (hopefully).

You can check the instructions inside the file **“Instructions for importing my own logical editor presets.pdf”.**