Cubase 12 InControl Driver for Novation SL MKIII

1 Introduction

With the introduction of Cubase 12, I was quite curious to discover that the new MIDI Remote feature was part of the release. I experimented with the Mapping Assistant but quickly discovered several limitations to what I was hoping to do.

Thankfully, Benjamin Bajic was motivated enough to create an experimental framework from one of the MIDI Remote examples of writing a driver in JavaScript. I downloaded his beginning version and started exploring. While it was quite limited, it did illustrate some of the capabilities of doing the driver in JavaScript.

There was a substantial learning curve across a variety of technologies (SL MKII, Cubase, JavaScript, MIDI Remote, etc.), but a working sample, however limited, was invaluable to get some motivation to work on this.

Once I could master enough functionality and JavaScript syntax, I started adding some samples of obvious things missing from the initial sample. Some examples are:

- Poor support for updating the displays.
- No support for LEDs.
- Partially functioning Mixer page, providing mixer surface functionality to Cubase.
- Mostly unmapped “Selected Page”, complete with shared controls, illustrating access to Mapping pages.
- Some functioning navigation controls. Somewhat limited but a good introduction.

2 SL MK3 InControl Controls

I have not discovered why yet, but it seems that some mapping capabilities in normal operation are not available when using InControl mode. This will need to be investigated. It appears that mapping is available in normal mode to control VST instrument properties. For InControl mode, the mapping seems limited to the functions listed in the Cubase Functions portion of the mapping assistant.

3 Surface Control Elements

The first thing required to define a MIDI Remote driver script is to define the components that are available to be assigned so that you can construct the virtual controls. You are building a virtual representation of the programmable controls as they are on the physical hardware. In the case of a simple driver, you will potentially define a single page that contains all the programmable controls. In driver terminology this is called a Mapping Page. The mapping page is explained in the following section.

Regardless of the complexity of your driver/device, the control components you require must be created with a parent of either page or subpage.

3.1 Page

A simple driver starts with a single mapping page. You can think of a page similarly to how you use a page in a notebook. All controls created while a page is active exist only on that page. Similarly, when these controls are assigned bindings, these bindings are also confined to the active page.

For more complicated drivers, additional pages may be created. Since the overall goal is to emulate the control surface, every page will ultimately tend to represent the same control layout, but the layout of a page is independently created and placed. Note that there is no restriction indicating that all pages contain all controls. However, in practice it’s probably best to create the layout of each page so that all the controls exist at the same location on every page.

Essentially, you are creating the control layout matching your controller on every page, while providing the control mapping on each page for different purposes.

Callbacks provided by the Page:

- On Activate – Called when a page becomes active.
- On Deactivate – Called when a page becomes inactive.

Pages are a part of the DeviceDriver object.

3.2 Subpage

The subpage behavior is very similar to the page. One or more subpages may be allocated on a single page. Like pages, controls are associated with a subpage when they are defined and must be defined for each page where they are expected.

It is important to understand that the Cubase MIDI Remote mapping assistant currently does not use subpages. If you require subpages, currently the control mapping must be done in the JavaScript ES5 code.

Callbacks provided by SubPage:

- On Activate – Called when a page becomes active.
- On Deactivate – Called when a page becomes inactive.

3.3 Controls

4 Binding

Controls that interact with the SL and Cubase must bind their callback functions. The type types of callback provided by the MIDI Remote API are:

- Bind Cubase functions to visual surface controls. This is how Cubase notifies the driver when something changes.
- Bind visual surface control to MIDI CC/Note. CC assignments in InControl mode are FIXED. This is how the SL communicates with the driver when controls are changed.

The MIDI Remote API via Cubase also provides services to the driver. For example, in addition to the actual control value (e.g. 0x00-0x7f), Cubase also notifies the driver what the display value should be (LR pan, db, etc.) There is no assurance that the display values will fit in the available control windows. This implies that text translation and/or configuration of short names.

4.1 Cubase (Host) Binding

When you change something in the Cubase GUI that is bound to surface/driver function, Cubase will call the associated callback function. This is a notification to the surface/driver that something of interest just occurred. Cubase is a host to the driver and the communication is via direct callback functions. Likewise, these callbacks are used to update the controls on the SL.

The Host binding connects a Cubase function to the surface control. The types of binding available are:

- Value Binding
- Command Binding
- Action Binding

4.2 SL (MIDI) Binding

When you change a control on the SL that has been bound to a surface control, the SL sends a MIDI CC message indicating the change. The SL binding indicates to the driver the new control value. The change in control value then notifies Cubase of the change. The following MIDI binding types are provided by the Remote API, with the ones used by the SL indicated in green:

- MIDI Note
- MIDI CC
- MIDI CC 14-bit
- MIDI CC 14-bit NRPN
- MIDI Channel Pressure
- MIDI Pitch Bend