
5
6
7
8
21
22
23
30
31
32
33
ELECTRA
ONE

Contents

User Guide 1

Quick Start Guide 3
Hardware overview 7
- Panel layout and connectors 8
- Connecting Electra 10
- Rear panel buttons 12
- Switching Electra One on and off 12

The Concept of Control 13
- Control types 14

Devices 17
- Level of device integration 17

Electra's User Interface 21
- Elements of the user interface 21
- The display 21
- Right side buttons 24
- Interacting with controls 27

MIDI message routing 31
External MIDI control 37
- Configuring a USB Host device 38
- MIDI message assignments 38

Electra Editor 39
- Electra App account 39
- Preset Editor 42

ElectraOne Console 59
- Firmware update 60

Troubleshooting 61

Firmware recovery 63
- Recovering from a system freeze 63

USB connection issues 67
- Electra USB MIDI ports 67
- Browser compatibility 67
- Verifying USB MIDI Ports 68

Tutorials 71

AU/VST control in Ableton 73
External control with LaunchPad 81
Writing SysEx templates 85

Developers 93

SysEx implementation 95
- The management port 95
- Manufacturer SysEx Id 95
- Get Electra info 95
- Upload a preset 96
- Download a preset 97
- Upload a configuration 98
- Download a configuration 98

Preset format description 101
- Why a new version? 101
- Preset JSON format 101

Control types 121
- Faders 121
- Scrollable lists 121
- Lists with bitmap data 122
- Pads 122
- Vertical faders 122
- Dials 123
- ADSR envelope 123
- ADR envelope 123
- DX7 (multi-stage) envelope 124
- Macro 124

Configuration format description 125
- Preset JSON format 125

Parsing SysEx messages 131

A simple example 131

The Patch element 134

The parsing rules 134
- The Instrument file inspector 136

User Guide

ELECTRA ONE

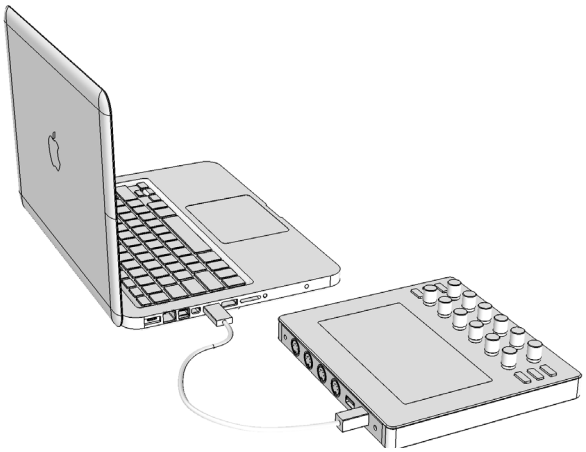
Quick Start Guide

Thank you for buying Electra One!

Electra One is a MIDI controller, a USB host, and a MIDI USB interface capable of streamlining your entire music production process. We hope Electra One will do the job you bought it for!

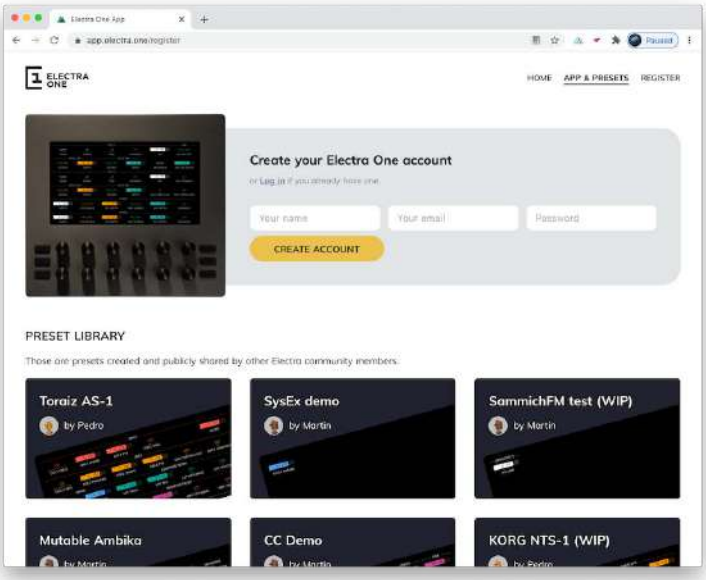
1. Connect your Electra One to a computer

Use the bundled USB cable to connect Electra One to your computer's USB port.



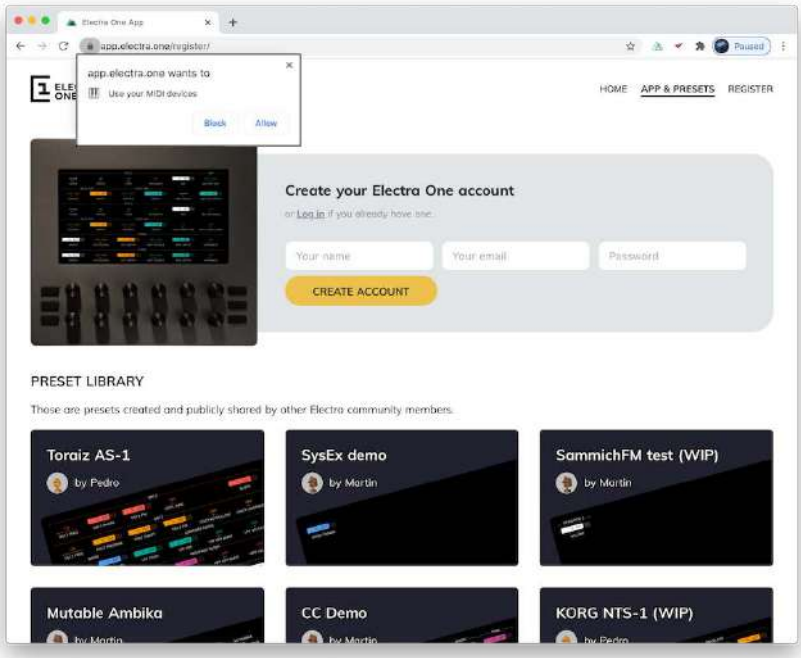
2. Create your Electra One account

Open Chrome or Edge browser and go to https://app.electra.one/register to create your Electra One account. An account will grant you access to the Preset library and the Preset editor.



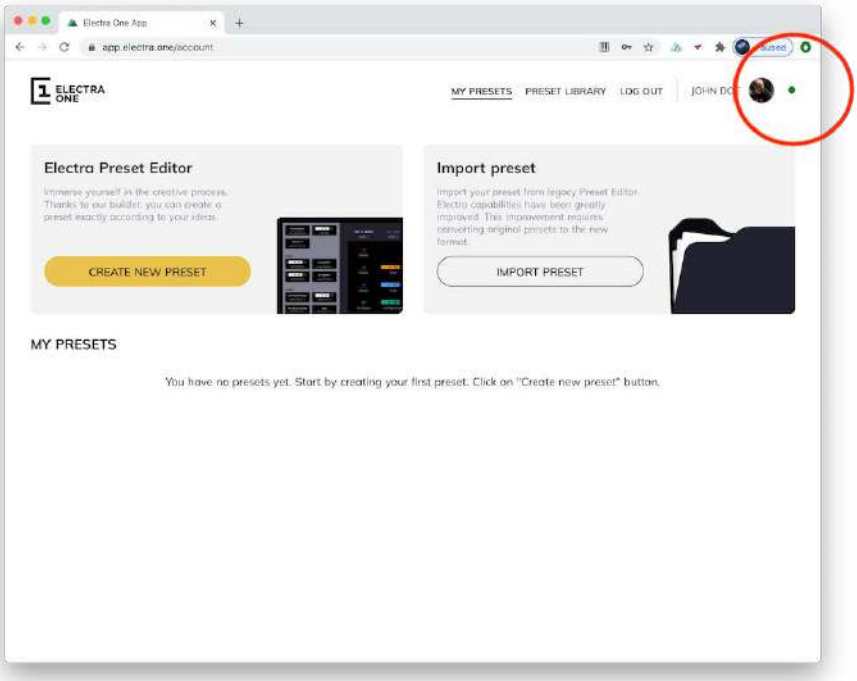
3. Allow use of MIDI devices

If you get a prompt asking about the use of MIDI devices, accept it by clicking ALLOW button.



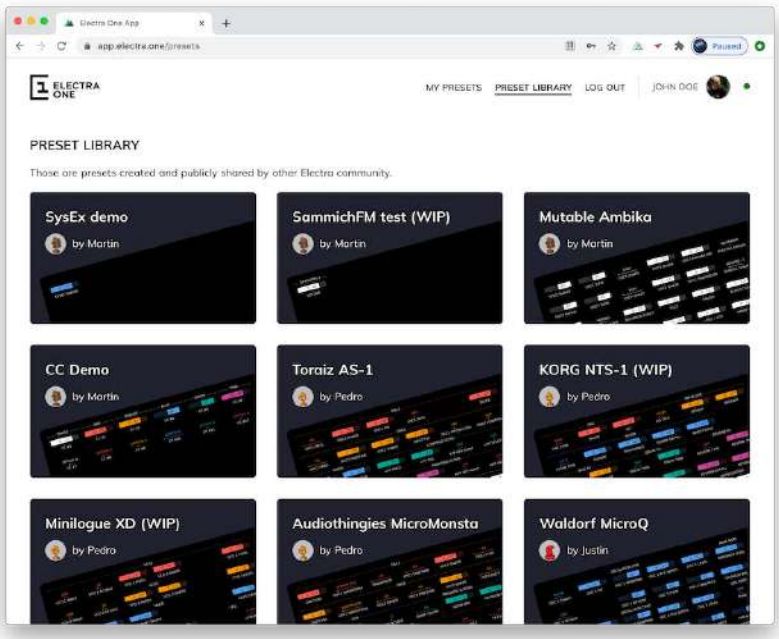
4. Verify that your Electra One is connected

Check the Connection indicator at the top right corner of the screen to see that Electra One is successfully connected. If the indicator is green, you are good to go. If it is grey, refer to The Connection Troubleshooting Guide



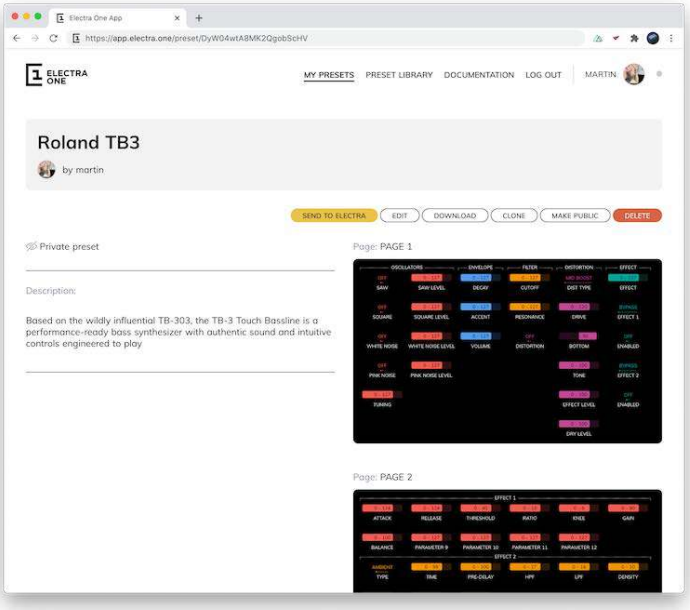
5. Go to the Preset library

Click on the **PRESET LIBRARY** link in the menu to pick one of many available presets. Click the preset you would like to send to your Electra One.



6. Send the preset to your Electra One

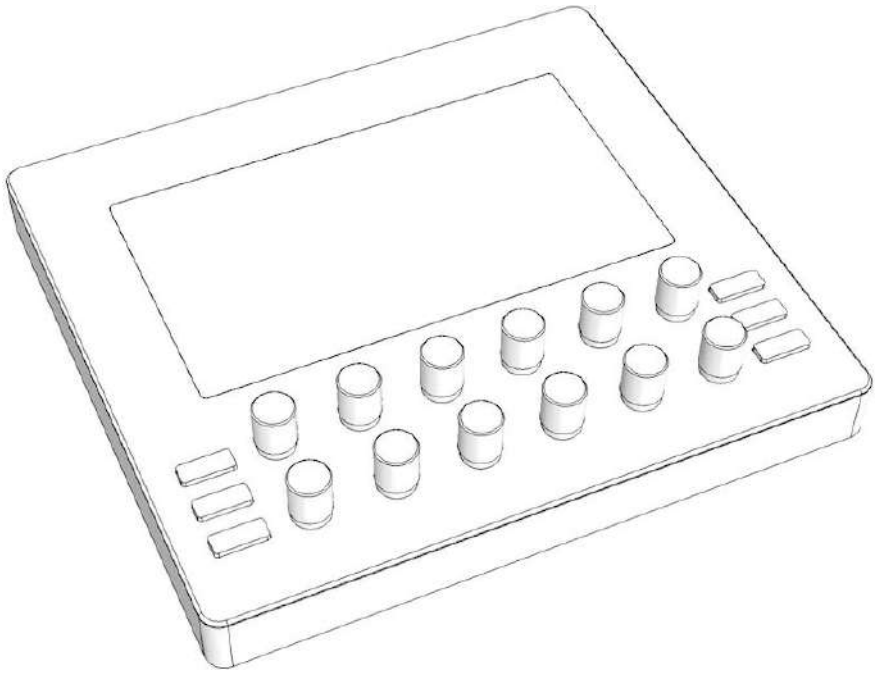
Hit the **SEND TO ELECTRA** button. Et voilà, the preset you picked is now loaded and ready to be used in your Electra One Midi Controller!



If you twist the knobs or use the LCD touch, your Electra One will send MIDI data according to the preset definition. To get more familiar with your new midi controller, proceed to the next chapter of the User Guide - Hardware overview.

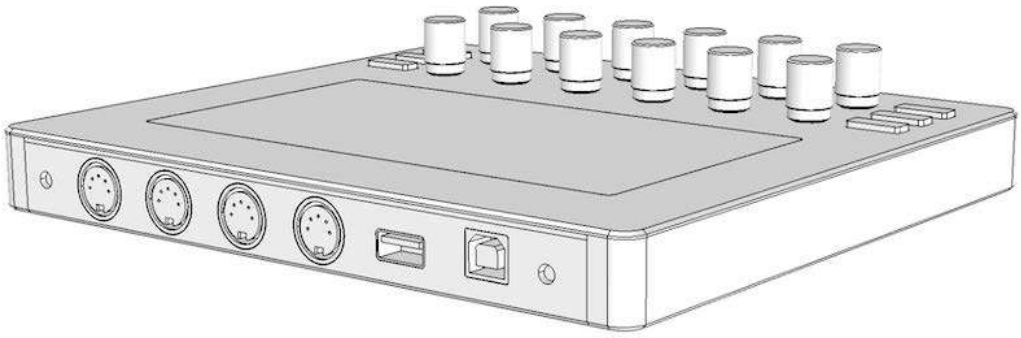
Hardware overview

Electra One is a MIDI controller, a USB host, and a MIDI USB interface capable of streamlining your entire music production process.



You can modify sound parameters by turning 12 smooth 360-degree knobs or with the touchscreen display. If you've ever wanted to control the cutoff filter of 10 synths at once then this controller is the answer. With 12 pages of controls and over 400 MIDI parameters within, Electra can satisfy all your MIDI requirements across multiple devices and synthesizers.

The controller has a USB MIDI Host port for plugging in other USB devices and two MIDI IN and OUT ports for connecting your MIDI gear.



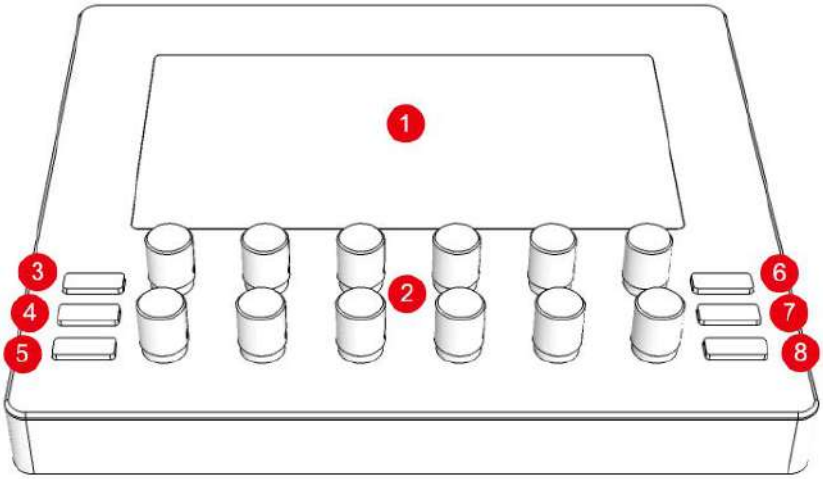
The controller does not only send MIDI messages to your instruments but it also visualizes all inbound data and it can even parse incoming SysEx messages with the patch settings. This gives you an insight into your gear's full capabilities.

The controller is programmed with a web based editor. The editor provides an intuitive way to create, edit, and share presets. Electra One's sheer versatility makes it a powerful tool in both live and studio settings.

Panel layout and connectors

Front panel

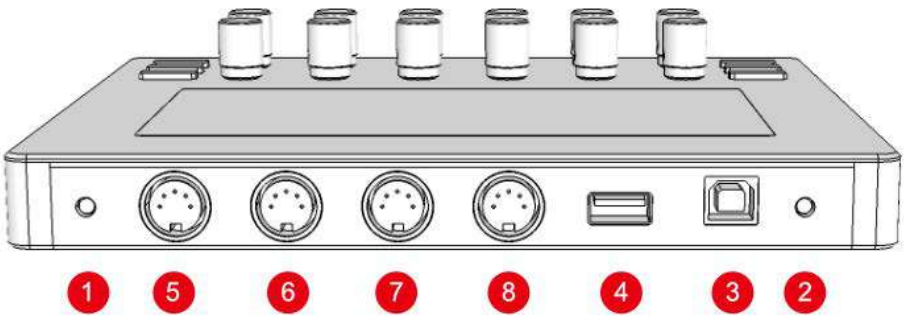
The front panel is the place where you interact with Electra One. A large touch-sensitive LCD displays all the controls and allows you to modify their values with a touch. 12 touch-sensitive knobs are assigned to on-screen controls and allow you to change values as well. The rubber buttons are here to navigate within Electra's pages, presets, and configuration.



1. Touch-sensitive display
2. Touch-sensitive 360-degree knobs. They are referred as KNOB1 (top-left) to KNOB12
3. `[SECTION 1]` button to make the upper section of controls active
4. `[SECTION 2]` button to make the middle section of controls active
5. `[SECTION 3]` button to make the bottom section of controls active
6. `[PATCH REQUEST]` button to download current patches from connected devices
7. `[SAVE]` button
8. `[MENU]` button

Back panel and connectors

The back panel serves two purposes. It is the place to connect your gear and the computer. There are also buttons to reset Electra and initiate the firmware update.

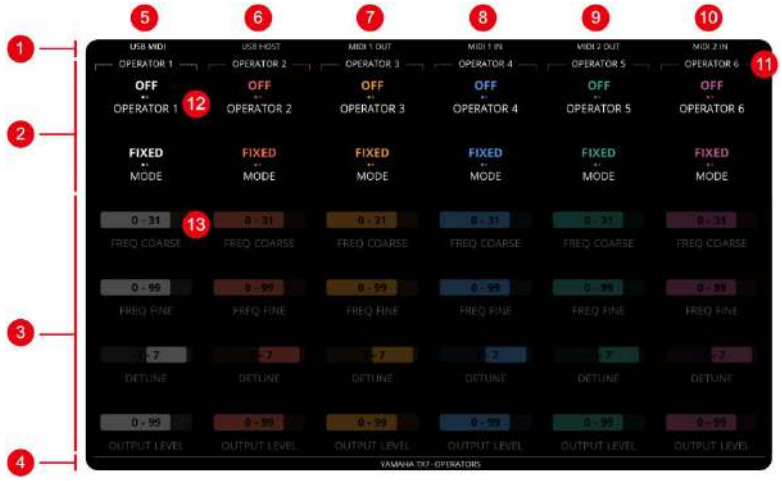


1. `[UPDATE]` button
2. `[RESET]` button
3. `<USB DEVICE>` port
4. `<USB HOST>` port
5. `<MIDI 2 IN>` port
6. `<MIDI 2 OUT>` port
7. `<MIDI 1 IN>` port
8. `<MIDI 1 OUT>` port

Ports `<MIDI 1 OUT>`, `<MIDI 1 IN>`, `<MIDI 2 OUT>`, `<MIDI 2 IN>` are often referred as to `<MIDI IO>`

Display layout

The display is the place where all the magic happens. It displays controls assigned to the synthesizer parameters as defined in the currently loaded preset. The display also provides information about activity on the ports, current preset, and page. The controls are organized in 6 x 6 grid, giving you access to 36 controls per page.

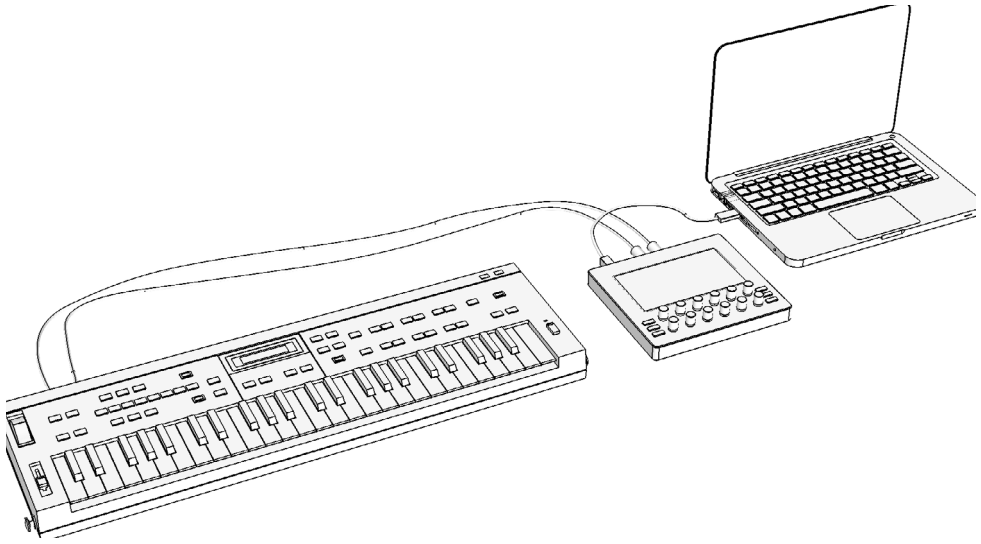


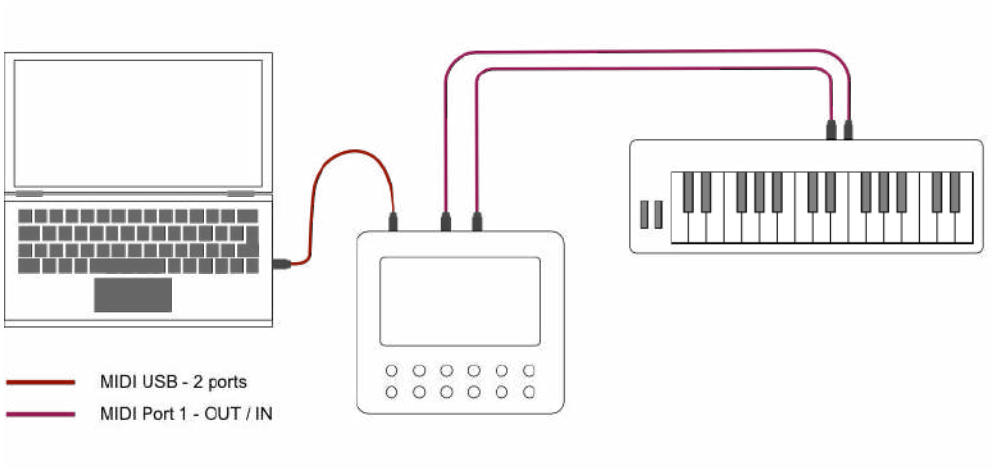
1. Status bar with MIDI port activity indicators
2. Active section of controls
3. Inactive sections of controls
4. Bottom bar with the preset name and current page name
5. Indicator of activity on `<USB DEVICE>` port
6. Devices connected to `<USB HOST>` port and indicator of activity
7. Indicator of activity on `<MIDI 1 OUT>` port
8. Indicator of activity on `<MIDI 1 IN>` port
9. Indicator of activity on `<MIDI 2 OUT>` port
10. Indicator of activity on `<MIDI 2 IN>` port
11. Group of Controls
12. List Control
13. Fader Control

Connecting Electra

A basic setup

Electra can be connected to your studio setup in many ways. For the start, the following configuration is ideal:





In this setup, Electra can control sound parameters of the synthesizer connected to `<MIDI IO>` port. You will see Electra' USB Device ports on the connected computers, we refer to them as `<USB DEVICE>`. These ports can be used to send and receive MIDI messages to and from the synthesizer. It means Electra acts here as a MIDI controller as well as a MIDI USB interface.

There are three USB device ports:

- `Electra Port 1`
- `Electra Port 2`
- `Electra CTRL`

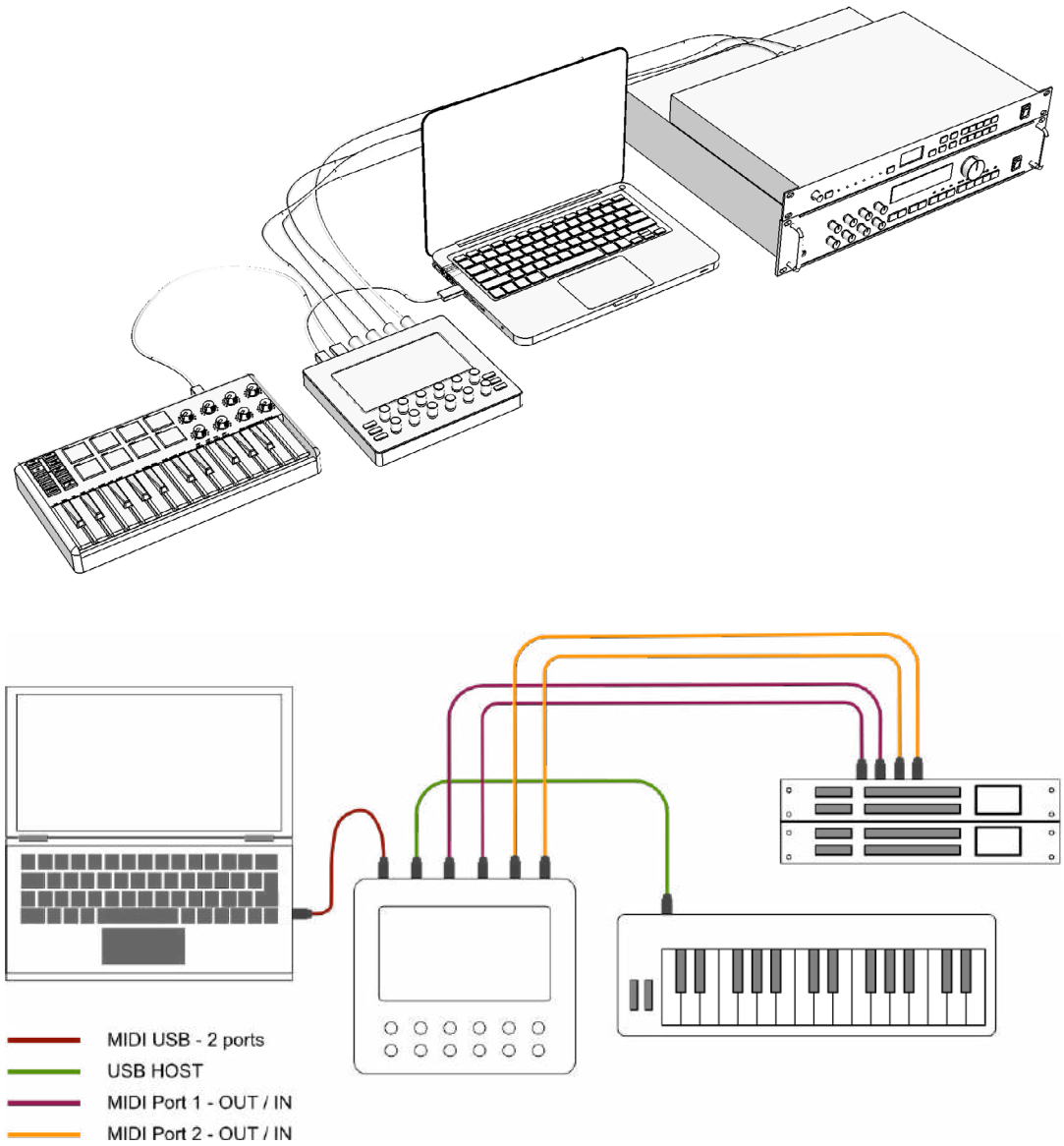
Any MIDI message sent to USB device Port 1 OUT, will be forwarded to `<MIDI 1 OUT>` port. Any message received on `<MIDI 1 IN>` port, will be forwarded to USB device Port 1 IN. Ports 2 work in the same way.

Electra CTRL is dedicated to communication between Electra and Electra Editor.

> TIP
>
> If you do not see Electra CTRL, it might be called Port 3. Some versions of operating systems do not read the port name correctly. Please review The Connection Troubleshooting Guide.

A complex setup

To illustrate Electra's capabilities let's take a look at a more complex setup:



Here we connect more gear to the `<MIDI IO>` ports. Next to that, the Master keyboard is connected to Electra's `<USB HOST>` port.

In this setup, you can have full bi-directional control over two hardware synth modules. The MIDI messages generated by the Master keyboard are automatically forwarded to both `<MIDI IO>` ports and to the `<USB DEVICE>` ports.

MIDI messages generated by turning the knobs are being merged to the flow of MIDI data according to Electra's settings. These messages will be sent to both `<USB DEVICE>` port as well as to `<MIDI IO>` ports.

Rear panel buttons

The Reset and Update buttons are here to help you out when you get into a tricky situation. As these should be very rare events, the buttons are recessed and can be reached with a pen or with another thin object.

Reset button

`[RESET]` button will power recycle the controller. The effect of pressing the preset button is the same as disconnecting and reconnecting the USB cable. Note, resetting Electra will not delete any of our presets or settings stored in the device.

Update button

`[UPDATE]` button is used to initiate a forced firmware update. Under normal circumstances, firmware updates can be initiated from the Electra One Console application without pressing the `[UPDATE]` button. If, however, your firmware update fails or if you upload a corrupted firmware file to Electra and it becomes bricked, pressing the `[UPDATE]` will allow you to restore the controller to the working state. Full description of the recovery procedure is described at Firmware recovery.

Switching Electra One on and off

Electra One starts up immediately after it is connected to the USB power. It does not matter whether it is connected to a computer or a USB power adapter.

The controller can be put on standby mode by pressing and holding the `[MENU]` button and pressing the `[SECTION 3]` button shortly after the `[MENU]` button is held down at least for 2 seconds.

Subsequent pressing of the `[MENU]` button will activate the controller.

Electra turns off the display, knob reading, and touch sensors in the standby mode. It does still, however, process and routes incoming MIDI data on all interfaces.

The Concept of Control

Electra can be described as a MIDI device that:

1. **Generates** MIDI messages according to user actions, such as turning the knobs or using the touch
2. **Forwards** MIDI messages between different ports, (MIDI IO, USB host, and USB device)
3. **Merges** messages (1) to the flow of messages (2)

A Control is an instance of a synth parameter. It visualizes the value of the parameter and it generates and sends corresponding MIDI messages when the parameter value is changed. Control also reflects any change of the value according to the incoming stream of MIDI messages.

The picture below shows a Fader Control representing Filter cutoff with the value currently set to 8.

Each Control has several attributes that define its function and appearance. The most obvious attributes are:

- the type of the Control (fader, list)
- the associated MIDI message type (CC, NRPN, SysEx)
- parameter number
- minimum and maximum value.

Electra users create Presets by organizing various Controls to layouts that suit their needs. Presets are created, edited, and managed in Electra One App and uploaded to Electra One controller over the USB. The picture below is an example of one Page of a Preset:

In this example, the Preset consists of a number of Controls used to edit patch parameters of one specific synthesizer. Presets may, however, consist of Controls that are linked to many synthesizers. This allows Electra not to be only a synth programmer but gives you the possibility to be in control of the whole rig of gear.

Control types

There are two types of controls currently:

- faders
- lists
- pads
- envelopes

The type of control tells Electra how the Control will look like. Also, it defines if the Control will work with continuous data ranges of values or with discrete values.

Fader

Faders are ideal for showing and sending continuous ranges of parameter values. A good example of a fader is a cutoff frequency of a filter or an oscillator detune. Turning the knob smoothly changes the current value of the assigned parameter. As you turn the knob corresponding MIDI messages are sent and the fader value is updated on the display.

The value may change within a given minimum and maximum.

If the value range minimum is negative, the zero value is placed proportionally within the length of the fader. Positive values are then represented by a fader bar extending to the right, while negative values extend to the left.

Optionally, faders may have Overlay assigned. Overlay replaces specified values with a text label. More on info on overlays is provided below.

Lists

The lists are one of the key features of Electra. They help to overcome the problem when discrete MIDI values have a specific meaning. Imagine a MIDI parameter VCF Envelope Mode that has the following four values:

MIDI value	Text label
0	Normal
16	Inverted
32	Normal with Dynamics
48	Inverted with Dynamics

The fader would not be very helpful here. When the list is used for this Control, the behavior of the knob changes to act pretty much as an encoder. Turning the knob switches the values of the parameter.

Each list value is represented as a dot. The currently selected one is highlighted and the label is shown. The lists may have up to 255 values. If the number of list items exceeds 16, the visual representation of the list slightly changes to a more fader like style. The text labels are still, however, shown instead of numeric values.

The lists do not have minimum and maximum values defined, they work only with values defined in the overlay.

Should the list receive a MIDI value that is not defined in the overlay, the numeric value will be displayed and the list will be highlighted with a red background.

Pads

The pads are controls that can be used to switch between two states / values of given parameter or to trigger transmission of MIDI messages.

The pad will transmit MIDI messages when its state is changed. It is up to the user to specify if a message will be sent on a change from **Off** to **On** state, or from **On** to **Off** state, or both. When going from **Off** to On state, **On Value** will be sent. When going from **On** to **Off** state, **Off Value** will be sent.

The pads support the following MIDI message types:

- cc7
- cc14
- NRPN
- RPN
- SysEx
- Program change
- Note
- Start
- Stop
- Tune request

If the type of the associated MIDI message does not support value, eg. Start MIDI message, the value cannot be specified.

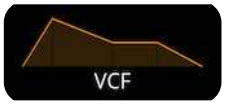
Program and Note numbers can be specified too.

The pads can act either as toggles or as momentary switches. A toggle stays switched in the new state, while a momentary switch always returns to the off state.

Pads are operated with display touch only.

Envelopes

The envelopes are multi-value controls that greatly improve the visual representation of ADSR and ADR envelopes. They allow users to map their MIDI parameters to individual segments of envelopes. Electra can then visualize the shape of the envelope whenever the incoming MIDI data modifies it or when the user changes the values with knobs.



The following types of envelopes are supported:

- ADSR
- ADR
- DX7 multi-stage

It is possible to specify which value will be assigned to the knob by default. For example, you may choose Attack being controlled by the knob. It is possible, however, to change this assignment on the fly when working with the preset. It means one knob can be used to change all values of the envelope.

The detail window of an envelope control provides access to all parameters with knobs. The knobs are remapped so that all parameters can be changed with knobs.

ELECTRA ONE

Devices

With Electra One a MIDI message is always send to or received from some sort of MIDI device. It can be any type of device capable of communicating using the MIDI protocol either using MIDI IO or USB:

- hardware synthesizer
- sampler
- DSP
- software plugin

A Device represents a musical instrument connected to one of Electra's hardware ports and listening on a particular MIDI channel. You will not be able to communicate with your instrument unless you register it as a device. A Device effectively tells Electra what instrument is connected to a particular port and channel. If you have two Waldorf Microwaves connected to your Electra One, you will need to have two devices configured.

One preset may have up to 16 devices configured. All of them can be used at the same time.

Controls do not refer to MIDI port and channel directly, instead they are linked to a Device. This makes it easy to do things such as changing a port of a Device. No changes to Controls are needed. Once you adjust the settings of the device, all Controls linked to this device will use the new settings.

Another important aspect of Devices is that they tell Electra what synthesizers are connected. This makes it possible for Electra to request and fetch patches from connected devices.

Level of device integration

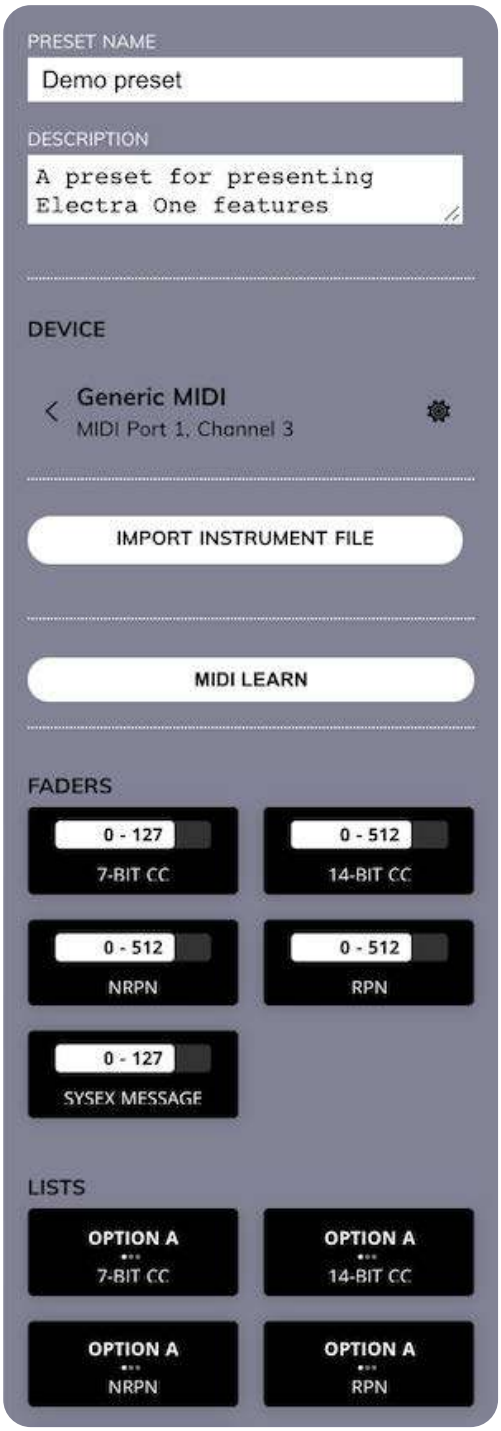
When you are setting up a device, you can use two levels of device integration:

- the generic MIDI devices
- the specific models of MIDI instruments
- the specific models of MIDI instruments with a patch reading

Generic devices

A generic Device tells Electra that there is some MIDI device connected given port and channel. Generic devices offer a palette of common predefined MIDI Controls that let users configure them as they wish.

Using a generic device is a very common practice, especially when you are building presets for your own use.



Integrated device

Integrated Device types on the other hand provide the implementation of particular models of musical instruments, such as, for example Yamaha DX7, Roland MKS-50, etc. Such devices

are implemented by so-called Electra Instrument Files (EIF). These are configuration files that provide information about all parameters, overlays, supported MIDI messages.

An integrated device provides you with predefined controls in the Editor sidebar. Your work on the preset is then minimized to making the layout of the preset, the hard work on the MIDI messaging is hidden with the device.

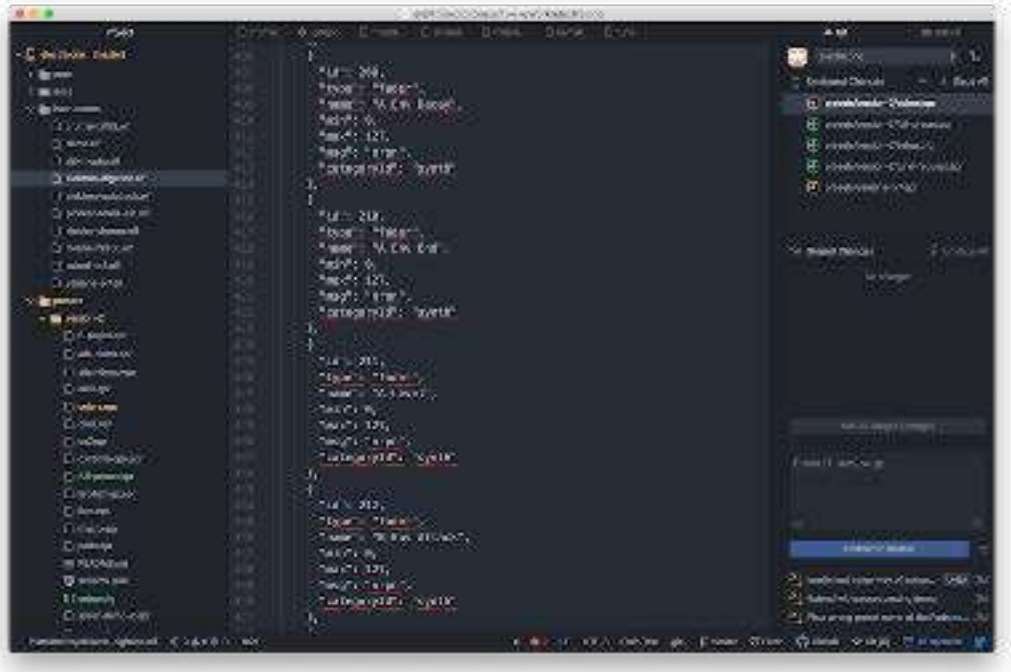
Of course, you still need to let Electra One know what port and channel the device is connected to.

Integrated device with patch parsing

An Integrated device with patch parsing is an extension of the Integrated device. Its instrument file consists of extra information about patch requesting and patch parsing SysEx definitions.

Such a device allows you to read the current settings of your instruments. This gives you perfect control over the instrument as you can read and visualize patches stored in instruments.

To create Integrated device users must have good knowledge of MIDI protocol and work with files in JSON format. The format of instrument files is described in the Developers section of the documentation.



ELECTRA ONE

Electra's User Interface

Electra One user interface is designed so that a computer is not needed once the presets are uploaded to the controller. All information that a user might need is available and accessible directly from the hardware. Of course, you may leave Electra One connected to a computer while using it. If you want to work with software VST plugins or DAW, you even have to keep it connected.

Elements of the user interface

The user interface relies on four important elements:

- the color display that shows information about presets and displays navigation means
- the display touch, that allows working with Controls and navigate between pages, presets, and configurations.
- the buttons to switch between sections, pages, as well as to load and save your data
- the touch-sensitive knobs that are used to improve orientation on the pages of Controls and to make choices on the selection screens.

The display

The status bar

The Status bar is located at the top of the screen. The status bar is meant to provide information about the USB device port, devices connected to the USB Host port, and MIDI IO ports.

The status bar is updated dynamically in real-time. The location of the status bar labels reflects the physical location of the ports on the rear panel.

USB Device port

The rightmost item of the status bar. It indicates the physical position and MIDI activity of the USB device port `<USB DEVICE>`.

USB Host devices

USB Host devices item provides information about MIDI USB devices connected to Electra's USB Host port. The item is also an indicator of MIDI activity on the MIDI USB host port `<USB HOST>`. The indicator is flashing whenever there is a flow of MIDI data. The MIDI USB devices are identified by their "Product name"

MIDI IO ports

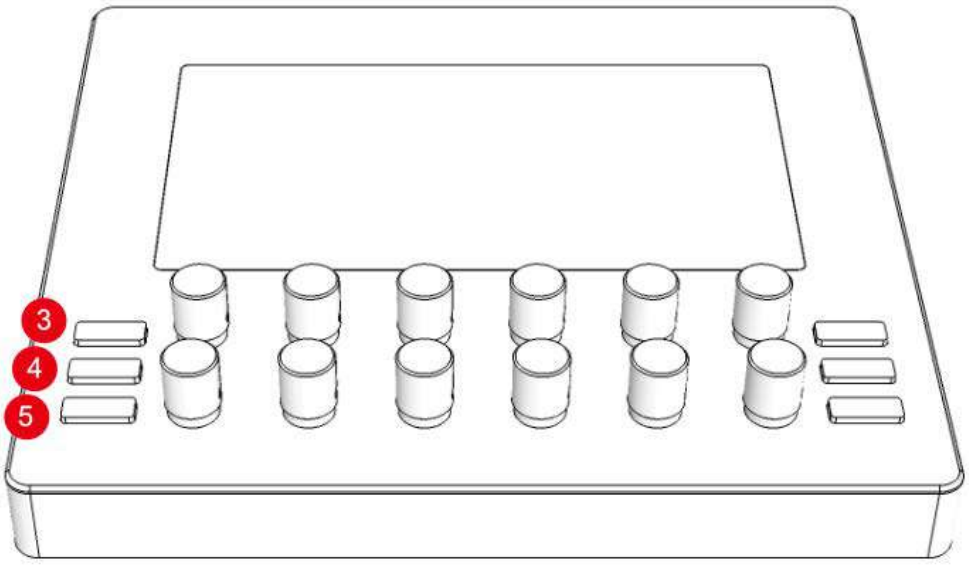
Indicators of the MIDI activity on the MIDI IO DIN-5 ports are referred to as `<MIDI IO>`. The indicators are flashing whenever there is a flow of MIDI data.

Active section

Electra's display can show up to 36 Controls on one screen. There are, however, only 12 knobs. To allow the user to easily reach all 36 Controls, the screen is divided into three sections of 12 Controls.

Only one section can be active at the time. The Controls of the active section are connected to the 12 knobs. Turning a knob will cause a change of the value of the corresponding Control.

The Active section is selected with the three buttons on the left side of Electra.



- Button 3, referred to as `[SECTION 1]`, makes the top section active
- Button 4, referred to as `[SECTION 2]`, makes the middle section active
- Button 5, referred to as `[SECTION 3]`, makes the bottom section active

The Active section is always highlighted, while the inactive sections are dimmed. The section highlighting helps users to get oriented on the screen and make a visual connection between the knobs and the controls.

The Active section can be also switched by clicking a Control on the display. The click will activate the section where the Control is located.

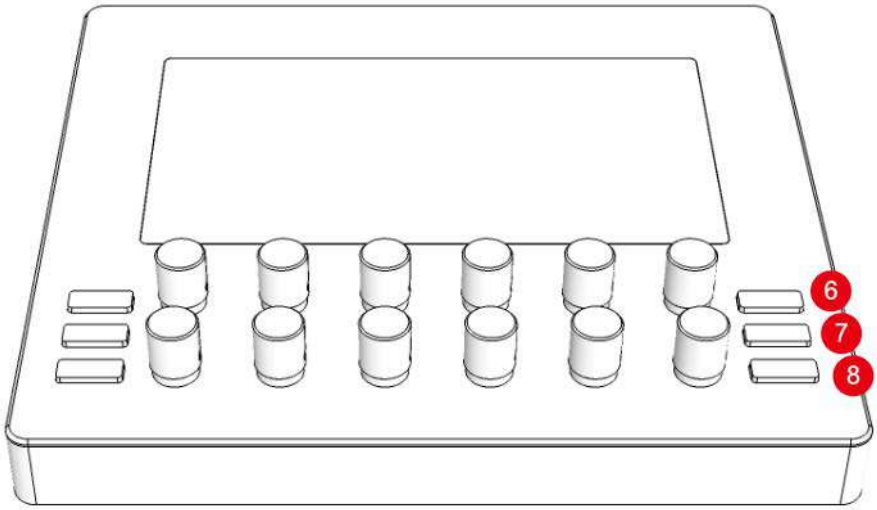
Top section active

Middle section active

Bottom section active

Right side buttons

Right side buttons provide access to preset synchronization operations and the Electra general menu.



- Button 6, referred to as `[PATCH REQUEST]`
- Button 7, referred to as `[SAVE]`
- Button 8, referred to as `[MENU]`

Patch request

If preset consists of instructions to request and parse device patch data, the requests are sent when the `[PATCH REQUEST]` button is pressed. Electra One controller will update the values of the Controls according to their patch settings as soon as the Device responds with the patch dump MIDI data.

Save

Pressing the `[SAVE]` button will initiate a SysEx dump of the current preset to Electra CTRL port.

Menu

Pressing `[MENU]` switches Electra to the menu window mode. The menu window can be used to:

- switch between preset Pages
- switch between Preset banks and Presets stored in Electra One's internal storage
- configure assignment of USB devices connected to USB Host port `<USB HOST>`

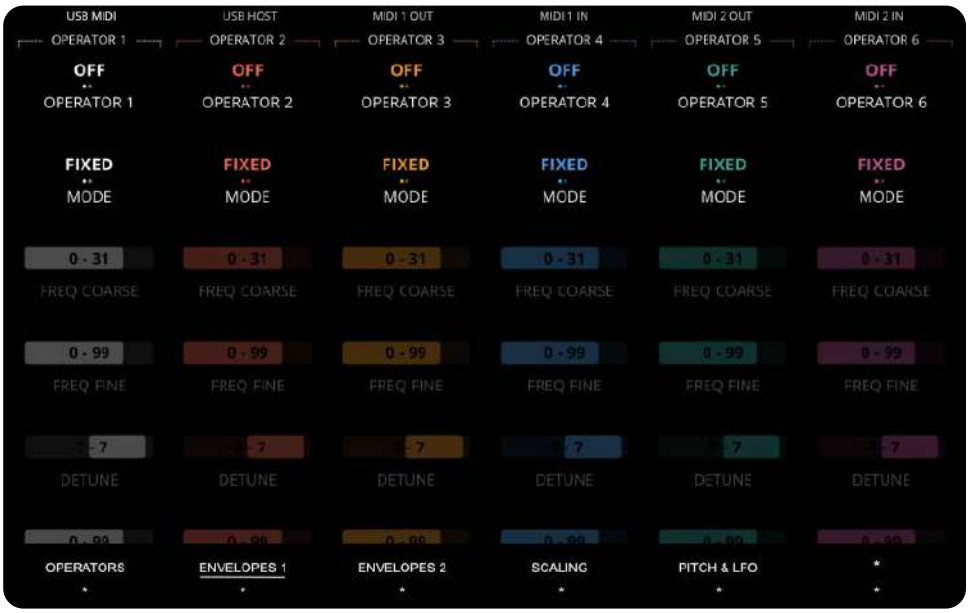
Page selection

Each Electra preset may have up to 12 pages of Controls. There are two ways to switch between the pages:

- Quick page browsing
- Page selection menu

Quick page browsing

When [MENU] button is pressed and held pressed a list of all pages will appear at the bottom of the screen.



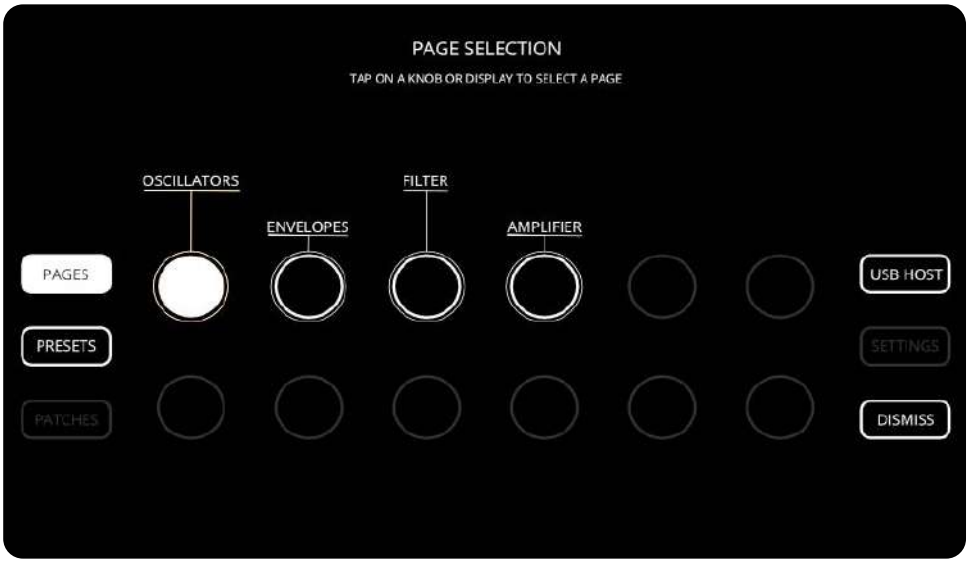
The currently selected page is marked with an underlined label. The page names are displayed so that their position corresponds to the knobs. The knob touch can be used to preview the pages. Pages marked as * are unused pages, these cannot be previewed or switched to.

If you change the selected page with knob touch, the Quick page view will disappear when the [MENU] button is released. If the [MENU] was just clicked without making a page selection, the Electra menu will be shown.

Page selection menu

The Page selection window is shown upon releasing the [MENU] button. It provides an overview of all pages and allows users to change the current page.

The page can be switched either by tapping the corresponding knob or touching the page name on the display.



The Page selection is displayed immediately after pressing the Menu button. It is done that way to provide a fast way to page switching.

The dimmed page icons represent empty pages. It is not possible to switch to them.

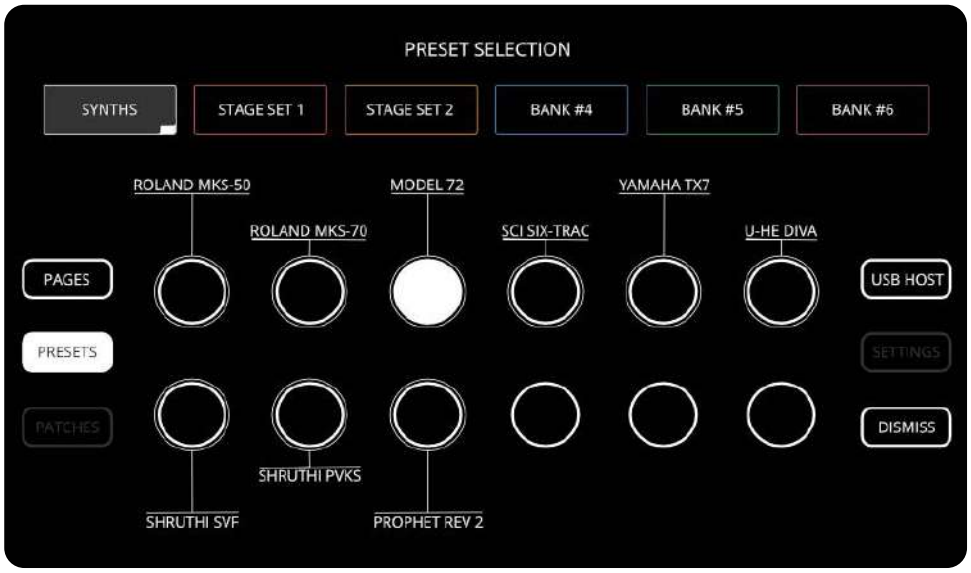
Preset selection

Not only that each preset has 12 pages, Electra One gives you a very fast way to switch between 72 presets, organized in 6 banks.

Changing the current preset

To reach the Preset selection, press the `[MENU]` and then tap the `[PRESETS]` on-screen button or press the left-middle hardware button that is temporarily assigned this function. The Preset selection window with 12 preset slots will appear.

The selection is done in the same way as selecting pages. You can use either tapping the knobs or with the touch on the display.



The preset is loaded immediately after it is selected. If the preset slot is empty, an empty preset will be loaded and shown.

The Electra App and the Electra Editor always send presets to the currently selected preset slot. If the selected slot contains a preset, sending a new preset from the editor will overwrite the original preset.

Changing the current preset bank

The banks can be switched by tapping the on-screen bank buttons

Each bank may have up to 12 presets. Once the preset bank is changed, you can choose the preset as described above.

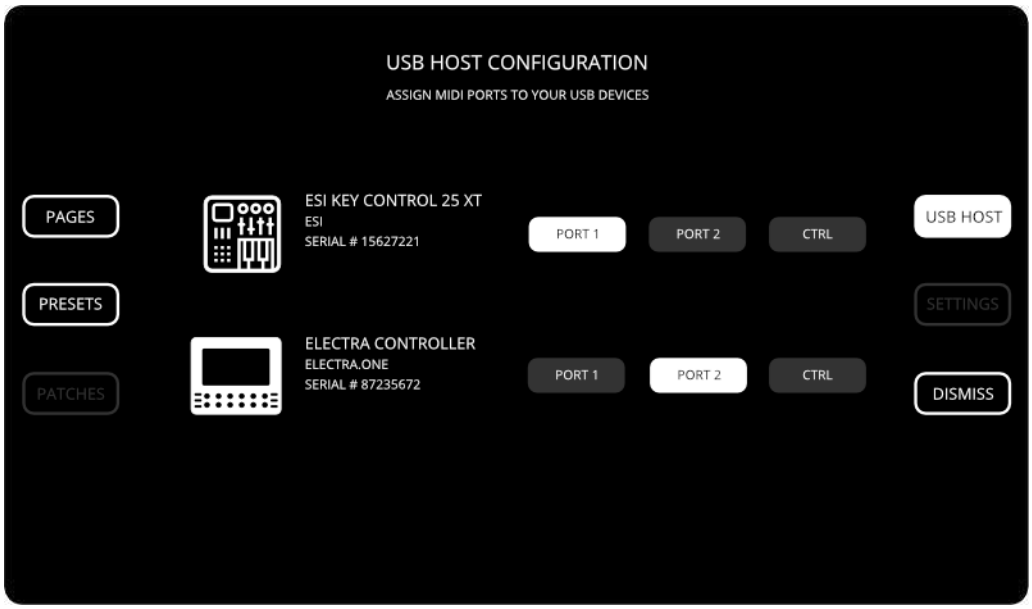
The active bank is highlighted with a color background:

The bank that consists of the currently loaded preset has a special indicator at the right-bottom corner:

USB Host configuration

Electra features a `<USB HOST>` port to connect USB MIDI devices and Electra accessories. Although there is only one USB port, it is possible to connect a standard USB Hub to increase the number of USB ports.

Any USB MIDI device connected to Electra's `<USB HOST>` port can be assigned to PORT 1, PORT 2, or CTRL.



If the USB device, for example, a master keyboard, is connected to PORT 1, the messages it sends will be forwarded to `<USB DEVICE>` port Electra Port 1 and `<MIDI 1 OUT>` port. Any MIDI message sent to either `<USB DEVICE>` port Electra Port 1 or `<MIDI 1 IN>` port will be automatically forwarded to the USB device connected to the `<USB HOST>` port 1. PORT 2 uses the same principle.

The CTRL port is used for Electra's external MIDI control. External control allows you to switch pages and presets by sending MIDI messages to Electra's `<USB HOST>` port.

Interacting with controls

Active control

To make orientation even easier, the touch on knobs marks corresponding controls underlined. It means you can see what Control is active even if you do not turn the knob.

Switching the Active section

A single tap on a Control will switch to the Active section where the control is located.

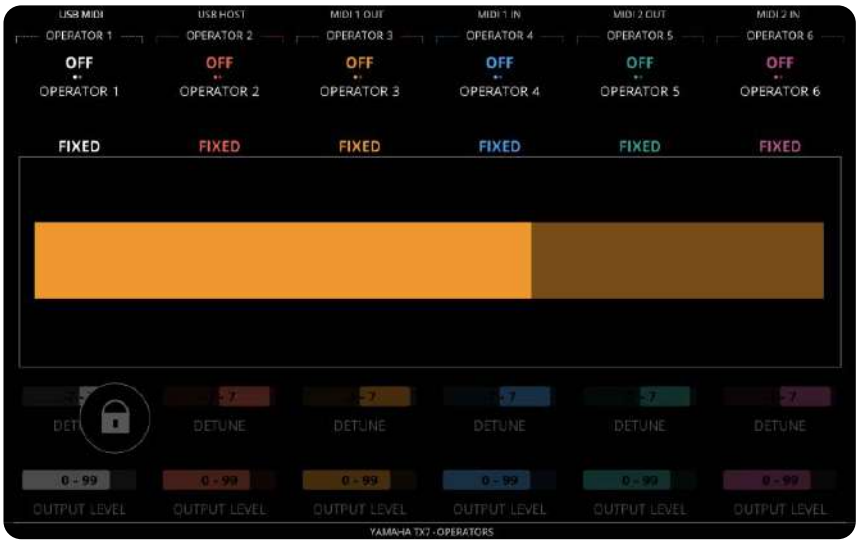
Control detail

Touch and prolonged hold of the Control will display a Control detail window with a full-size version of the Control. The Control detail provides you with fine control over the value. The detail window can be also revealed by holding the knob and pressing the `[SECTION 1]` button.

Each type of Control has its special detail window:

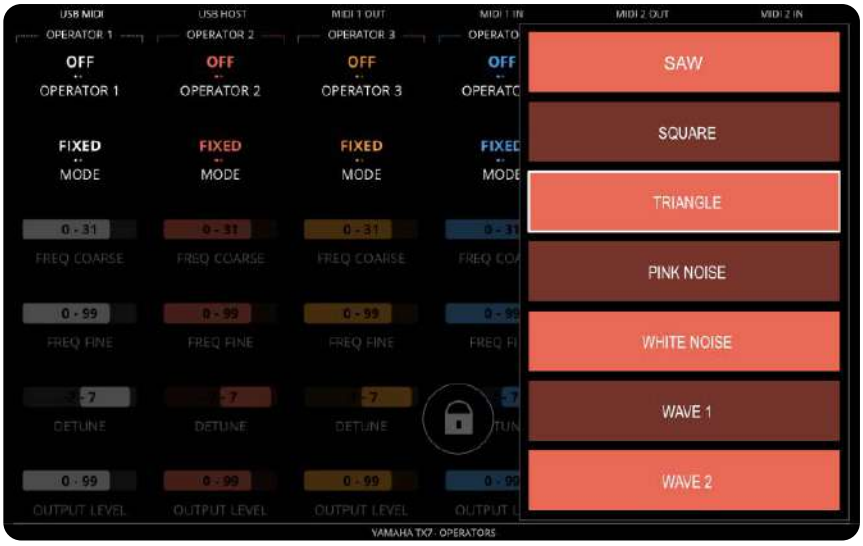
Fader detail

The fader detail is meant to allow maximum width sweeps of the Control value. The fader strip is wide and high enough to provide adequate room for your finger.



List detail

The list detail is meant to provide a tablet like swipeable list of values. The values can be browsed by swiping the items up and down. The item is selected with a single tap on the display.



Envelope detail

The envelope detail is meant to allow users to change all envelope values from one place. When the detail window is shown, Electra re-assigns knobs to individual envelope values. This temporary knob assignment is available only as long as the detail window is shown.

Locking the detail window

Normally, the detail window is closed immediately after changing the value or making a selection. This could be inappropriate in some situations. To prevent this the window can be locked by tapping the lock symbol:

When the lock is tapped, the symbol gets highlighted and the detail window can be used to change the value many times.

The locked detail window can be closed by tapping the lock symbol again or tapping the display elsewhere outside the detail window.

Detail window knob assignment

The detail window keeps the assignment of the original knob. It means you can use the same knob to change the value of the Control when it is displayed in the detail window.

Resetting to the default value

A double-tap on the Control resets its value to the default defined in the preset.

Changing active value of envelopes

The multi-value controls, such as envelopes, have knobs assigned to one of their values when the preset is loaded. This default assignment is set by the user in the Preset editor. There are situations, however, when a different value needs to be adjusted and opening the envelope detail window is not appropriate. In such situation, a quick active value change gesture can

be used. There are two ways to achieve changing the value that is assigned to the knob:

- hold the knob of the envelope (it means the name of the control is underlined) and tap the envelope control on the display. The value assigned to the knob will be switched to the next available and will become highlighted.
- hold the knob of the envelope (it means the name of the control is underlined) and press either `[SECTION 2]` or `[SECTION 3]` button. The `[SECTION 2]` will switch to the next available value within the envelope. The `[SECTION 3]` will switch to the sprevious available value within the envelope.

ELECTRA ONE

MIDI message routing

Electra One is not only a conventional MIDI controller but it is also a programable MIDI router and merger. Electra's internal router can pass MIDI messages between all types of interfaces. MIDI IO, USB device, and USB host.

Each type of interface has two ports. You can see those ports as a communication channels, where all ports 1 are interconnected as well as there is a connection between all port 2.

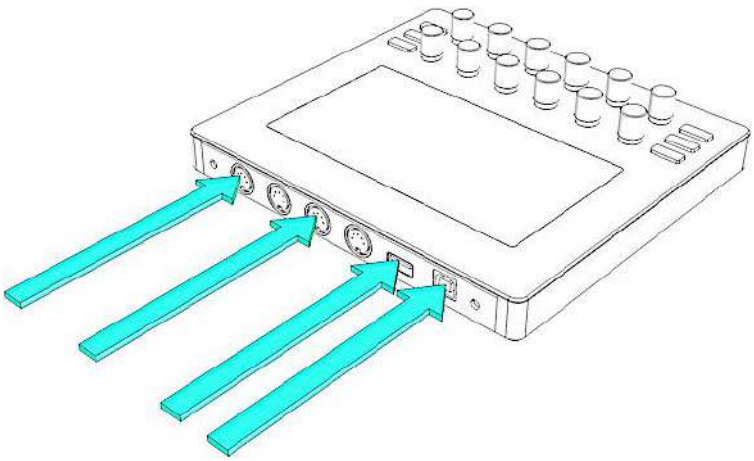
In the default configuration, all MIDI messages received on one type of interface are forwarded to other interfaces, keeping the same port and MIDI channel. The same rule applies to messages generated by twisting Electra's knobs or using LCD touch. These messages are sent to all interfaces and merged with other MIDI messages flowing through the controller. This configuration can be overridden by uploading a custom Electra One configuration.

Electra One forwards and merges all types of MIDI messages, including SysEx messages with size up to 1MB.

As this may not be still clear, let's take a look at a few examples.

Electra In

MIDI messages generated with external gear or software can be received on all Electra's interfaces, ie. `<MIDI IO>`, `<USB DEVICE>`, and `<USB HOST>`.

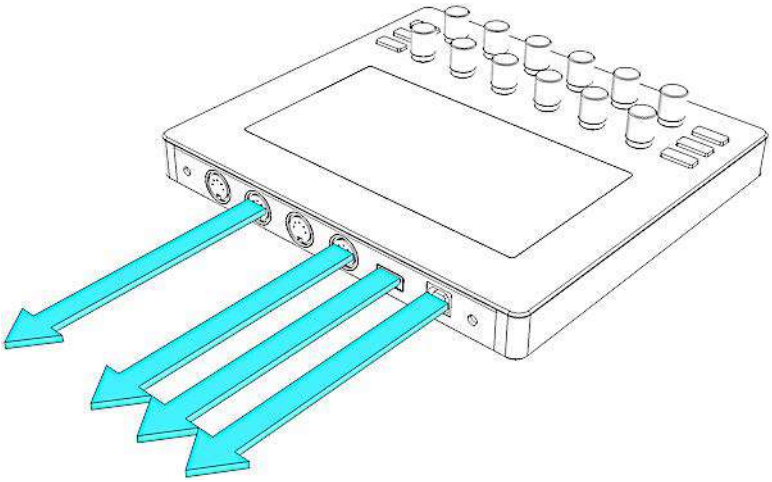


If the configuration of a Control matches the incoming MIDI message information about the port and channel (**defined by the device configuration**), the MIDI message will affect the value of the Control and you will get visual feedback on the display.

In other words, Electra monitors all incoming MIDI messages on all interfaces, if the preset consists of a Control listening to that given message, you will be able to see it.

Electra Out

MIDI message created by twisting Electra's knobs and using the display touch is sent to all Electra's interfaces, ie. `<MIDI IO>`, `<USB DEVICE>`, and `<USB HOST>`.

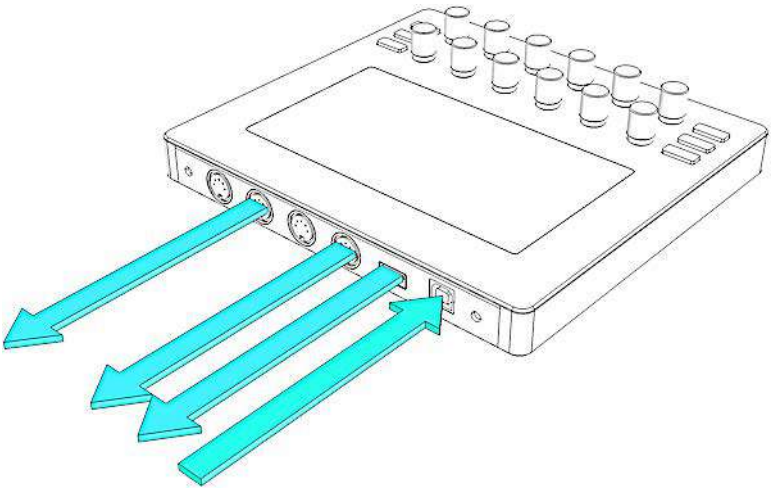


A control is always assigned to a particular device, which in turn represents a port and channel. When you twist a knob and it affects the value, a MIDI message is generated and send out to all interfaces.

In other words, twisting a knob will cause the same MIDI message to be sent out on the same port and channel of `<MIDI IO>`, `<USB DEVICE>`, and `<USB HOST>` interfaces.

USB Device input

A MIDI messages received from a computer over the USB cable, ie. received on the `<USB DEVICE>` interface.



Inbound `<USB DEVICE>` MIDI message received on port 1, will be automatically forwarded to port 1 of `<USB HOST>` interface and `<MIDI 1 OUT>` port. Of course, if the MIDI message matches a configuration of any Control within a preset, the value of that Control will be updated and shown.

The `<USB DEVICE>` interface makes the following ports (MIDI cables) available on the computer, where Electra One is connected to:

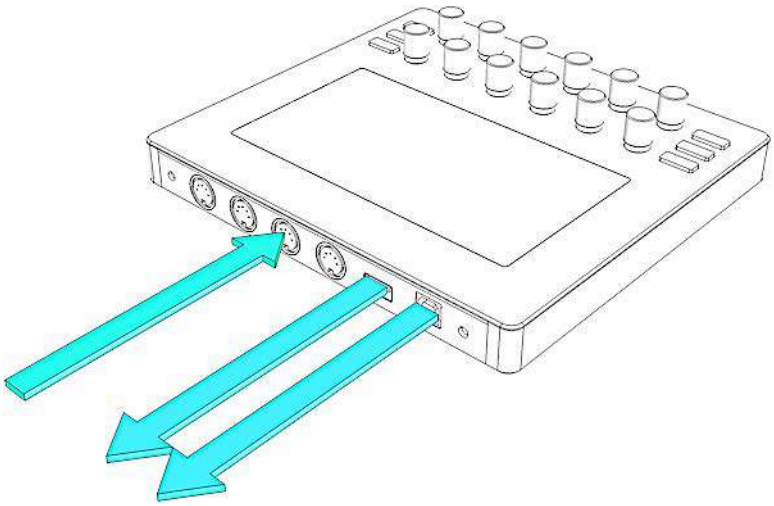
- `Electra Controller Port 1`
- `Electra Controller Port 2`
- `Electra Controller CTRL`

> WARNING
>
> On some systems the port names are not detected correctly, please review information in Troubleshooting connection issues article for more details.

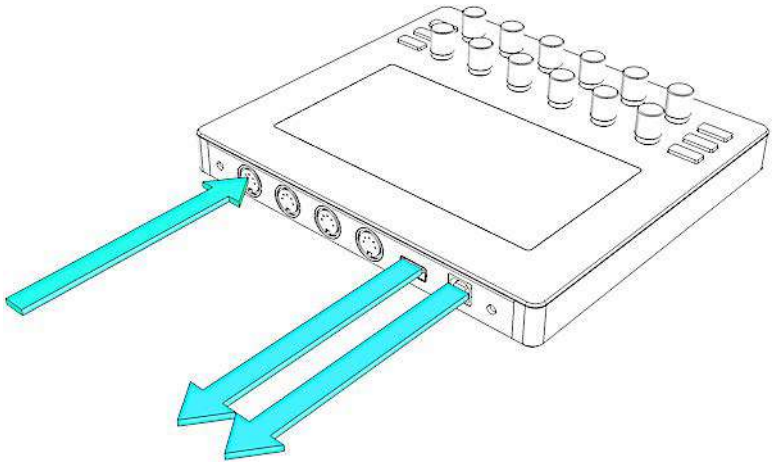
`Electra Controller CTRL` is Electra's management port. It is used for data transfers between Electra and the computer.

MIDI IO input

A MIDI message received from external MIDI gear connected to `<MIDI 1 IN>` or `<MIDI 2 IN>` will be forwarded to a corresponding port of `<USB DEVICE>` and `<USB HOST>` interface.



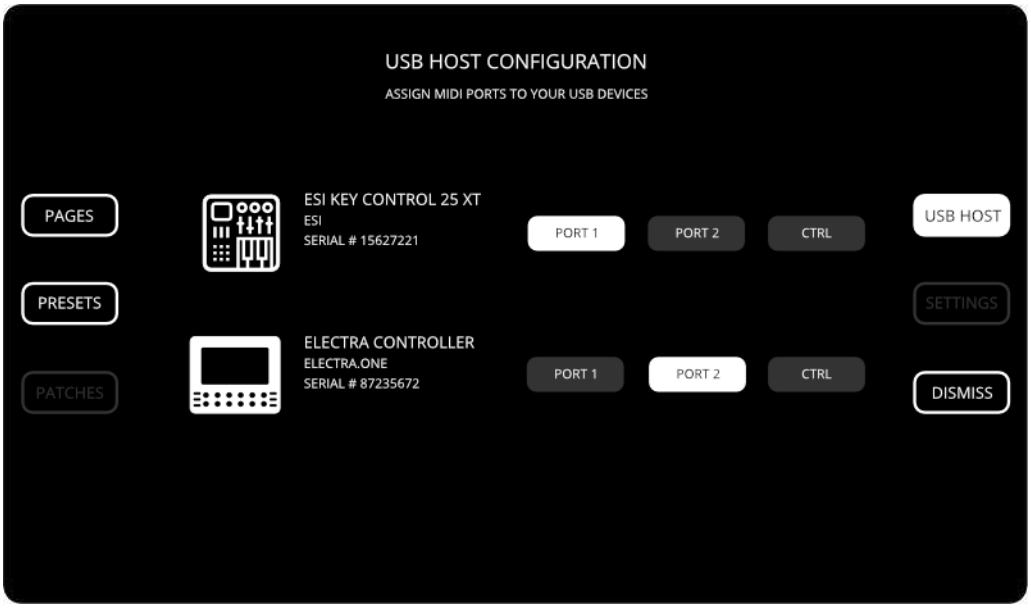
All messages received on `<MIDI 1 IN>` port will be forwarded to `<USB HOST>` port 1 and `<USB DEVICE>` port 1.



All messages received on `<MIDI 2 IN>` port will be forwarded to `<USB HOST>` port 2 and `<USB DEVICE>` port 2.

USB Host port assignments

The USB host interface is a bit different from the USB device and MIDI IO interfaces. USB host does not have hard-wired port assignments. The default Electra configuration assigns devices to ports in the order as they are connected. The device you connect first will be assigned to port 1, the device connected as second will be assigned to port 2.

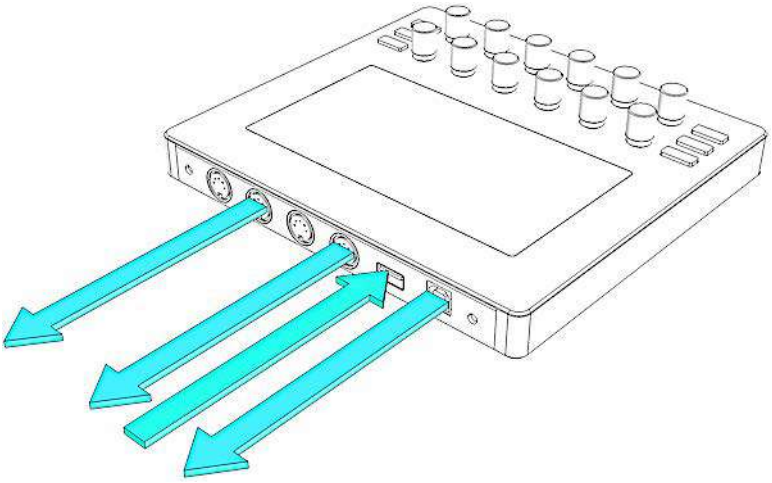


This might not be appropriate in all situations. The USB Host configuration page gives you an option to change the port assignments. These assignments will be, however, lost, when you turn Electra off.

Should you require a permanent assignment of devices, a custom Electra configuration must be uploaded to your Electra.

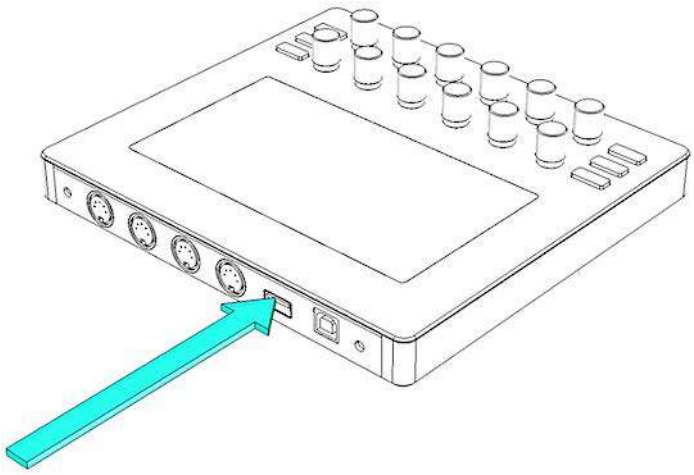
USB Host input

A MIDI Message received from a device connected to the `<USB HOST>` port will be forwarded to `<USB DEVICE>` interface and `<MIDI IO>` interface, keeping the information about the port.



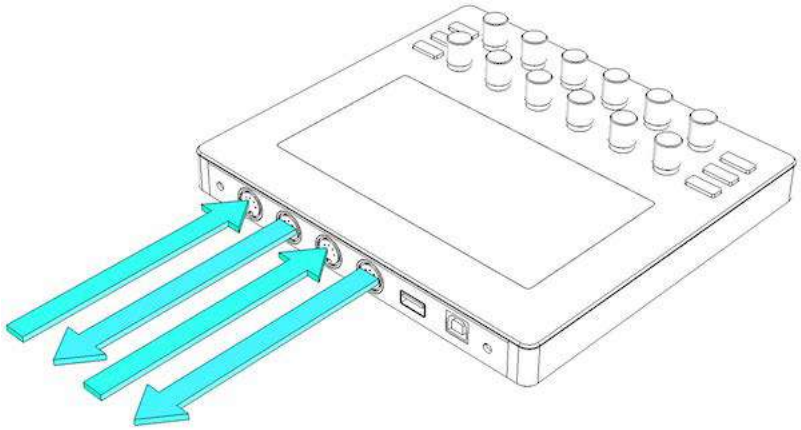
USB Host CTRL port

Messages sent to the `<USB HOST>` CTRL port are not forwarded to any other interface. They are interpreted as commands for Electra. The CTRL port is used for an External MIDI control.



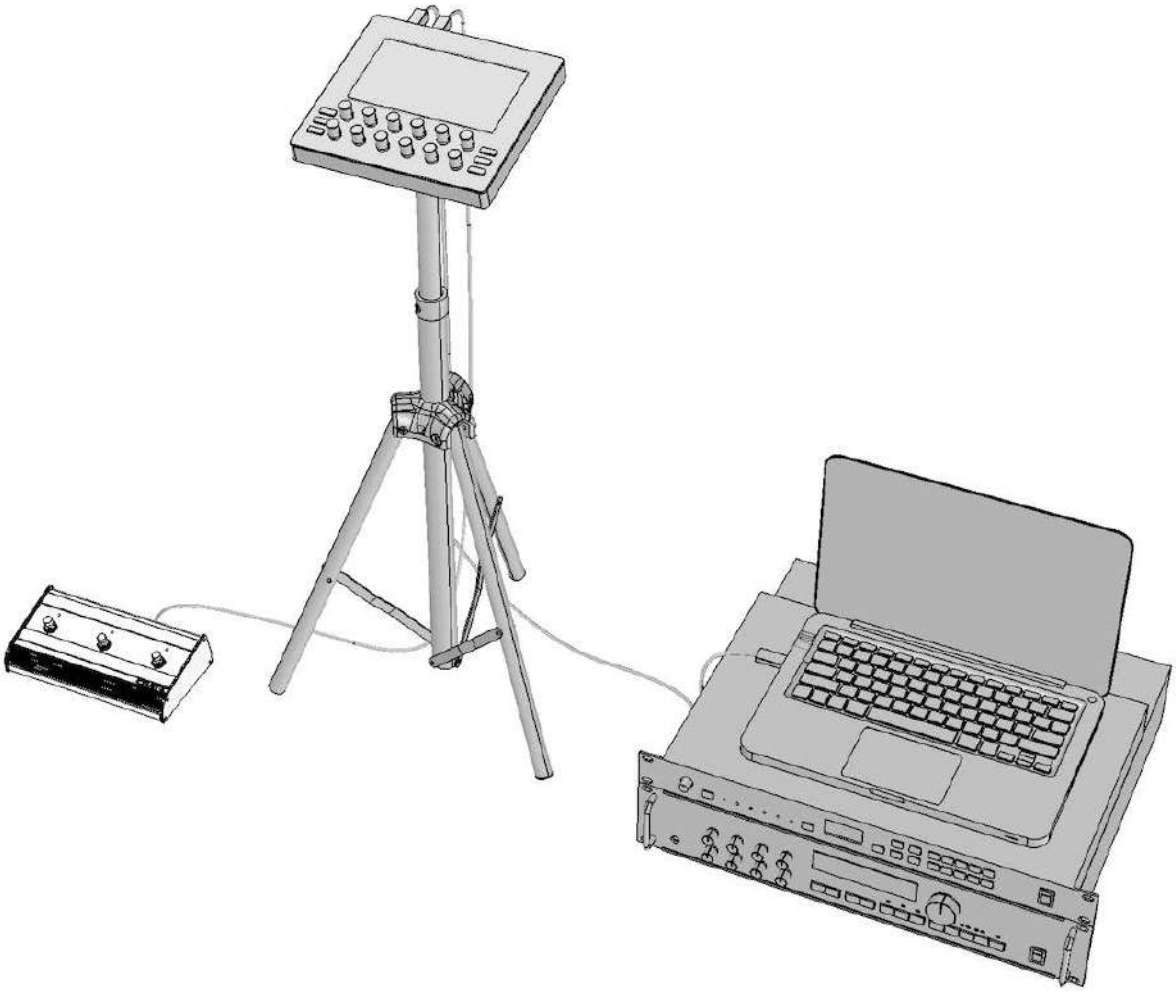
MIDI IO thru

When MIDI IO thru is enabled for port 1, all messages received on `<MIDI 1 IN>` port are forwarded and sent out on `<MIDI 1 OUT>` port. The same applies for port 2.



External MIDI control

There are often situations during your gig when you need both hands to perform and switching presets of pages could be cumbersome. The external MIDI control provides a neat solution.

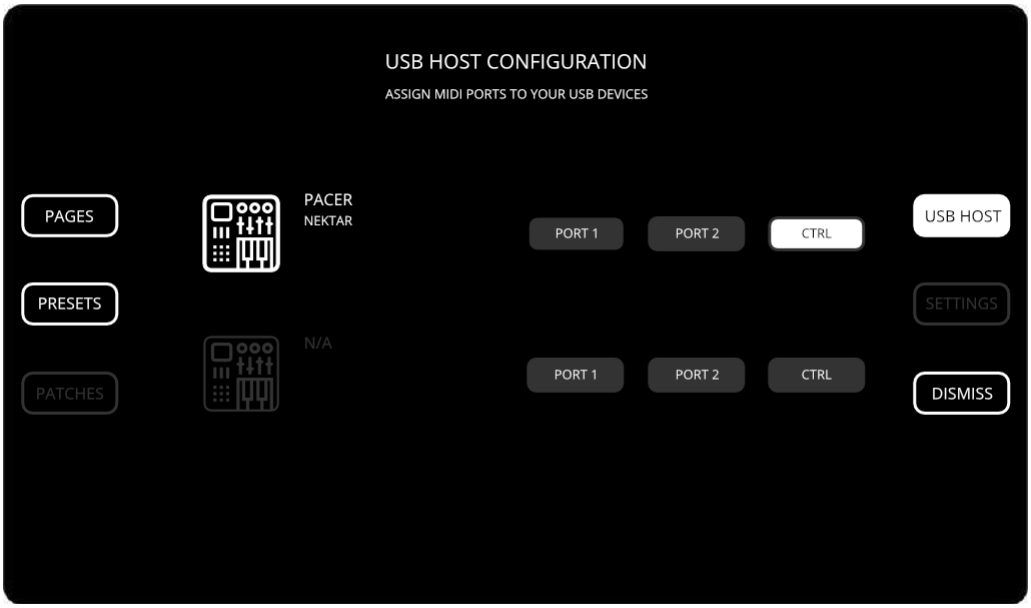


The external MIDI control allows you to map a number of Electra actions to MIDI messages sent over to Electra's `<USB HOST>` port. The currently supported actions are:

- switch to a specific page, identified by the page number
- switch to the previous page
- switch to the next page
- load a specific preset, identified by the preset slot number
- load previous preset
- load next preset

Configuring a USB Host device

In order to use a MIDI Controller to send commands to Electra, it must be connected to Electra's `<USB HOST>` and assigned to CTRL port. This assignment can be done manually at the USB Host configuration in the Menu.



A permanent assignment can be done by uploading a custom configuration file to your Electra One.

MIDI message assignments

Electra uses default mapping of MIDI messages as shown below. The default settings can be overridden by uploading a custom configuration file to your Electra One. When overriding default assignments, the following MIDI message types can be used:

- control change (CC)
- note on
- program change

We recommend Nektar Pacer and LaunchPads made by Novation for external MIDI control.

ELECTRA ONE

Electra Editor

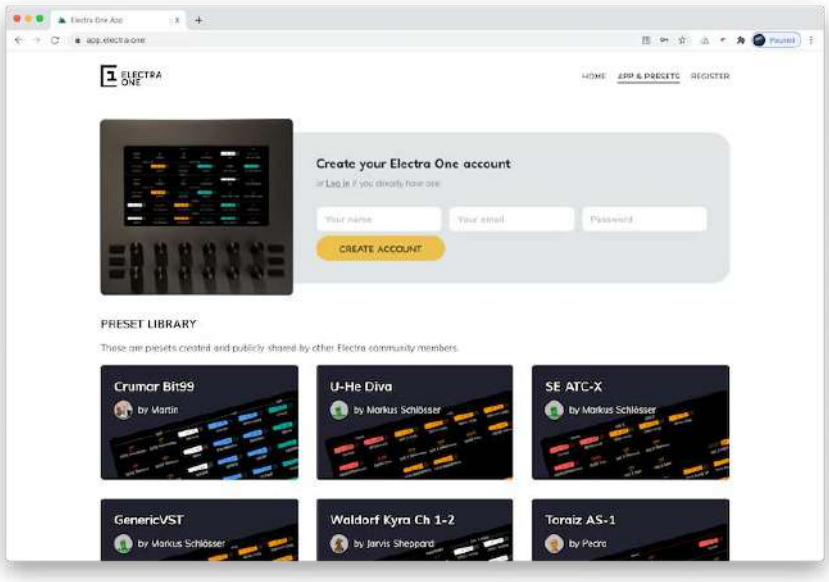
Electra App account

Electra App is preset management and editor

> WARNING
>
> Please note, browser with WebMIDI support is required. WebMIDI is currently supported with Chrome and Edge browsers.

Login to the application

Signing in to your Electra App account is required for accessing the library of presets and the Preset editor.

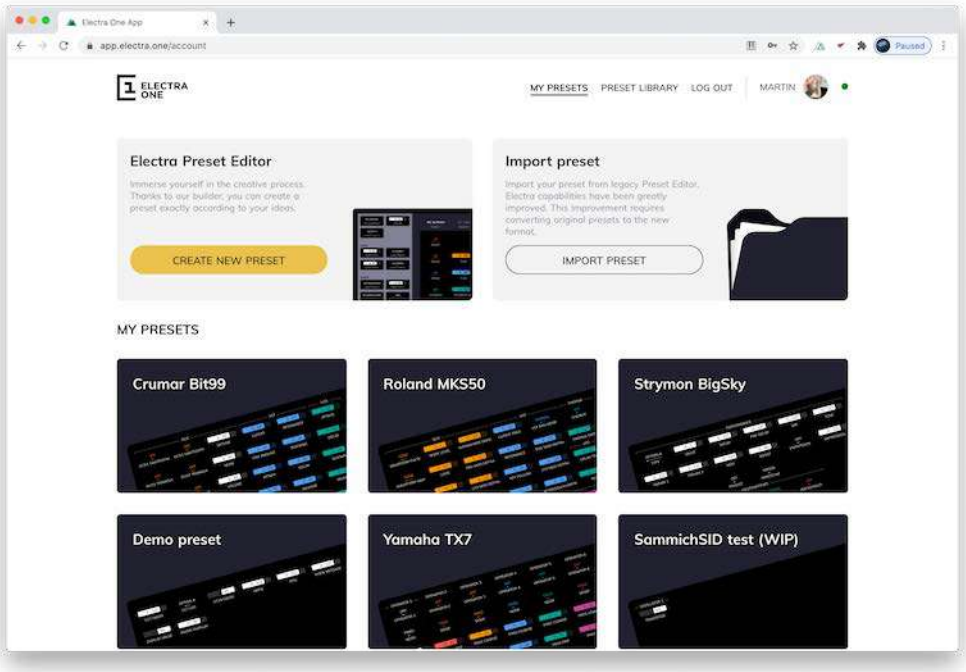


Once you log in you get access to two libraries of presets. One is owned by you while the other is a library of presets shared by other Electra App users.

My presets

My presets, as the name suggests, is a library of your own presets. There are three ways to add presets to My presets library:

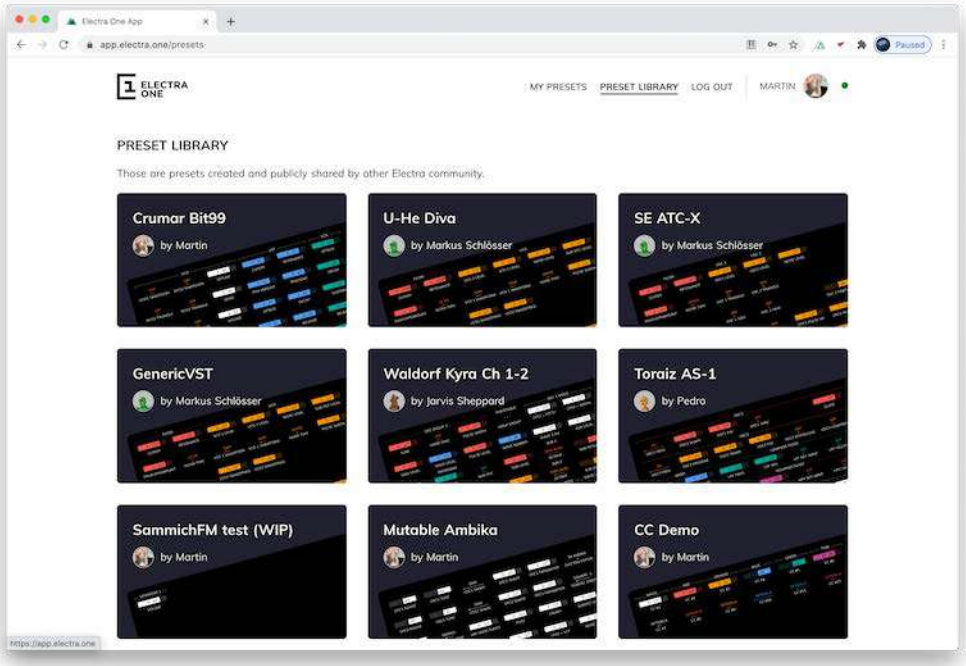
- Create them from scratch
- Import .epr files
- Get a copy of a preset shared in the public Preset library



The presets of your preset library are shown as tiles, each tile representing one preset. A click on the tile takes you to the Preset detail page. The tile also gives you an option to send the preset to Electra One controller directly from the My presets listing.

Preset library

The Preset library is a collection of all presets maintained and shared by other Electra App account users. A click on the preset tile takes you to the Preset detail page.

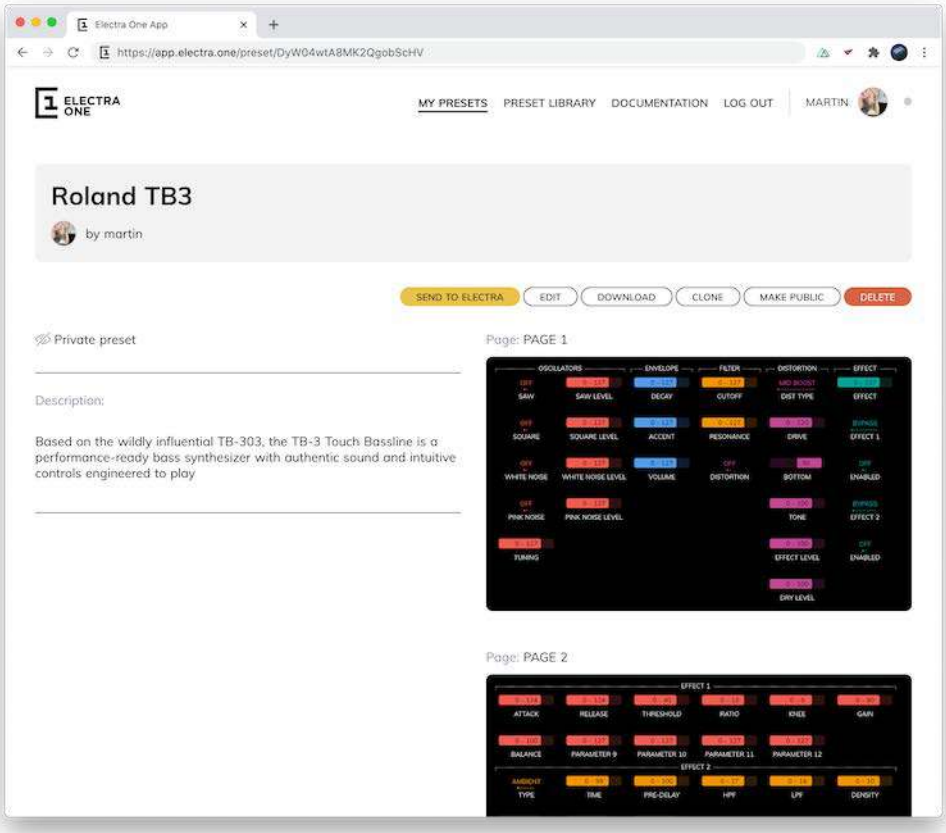


Preset detail

The Preset detail page gives you complete information about the and number of actions to take.

The top banner reads brief information about the name and the author of the preset. The actions are listed on the right side of the banner.

Below the banner, the left panel describes the preset while the right side shows the renderings of all preset pages. That means you can review the preset even without opening an Electra Editor or loading the preset to Electra One controller.



The preset detail has different actions for presets you own and for public presets.

With your own presets you can:

- Send the preset to your Electra One MIDI controller
- Open the preset in the Electra Editor to make changes to it
- Download the preset as a file
- Make a copy of the preset
- Share the preset with other Electra App account users, ie. made it public
- Delete it

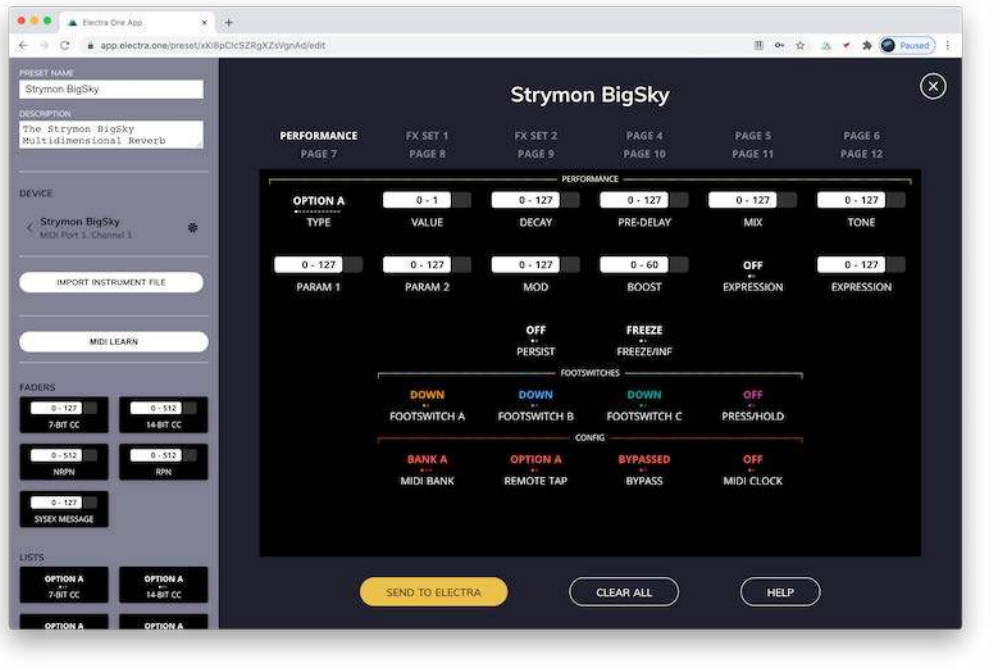
> TIP
>
> Deleting a preset removes the preset from your preset library. It does not remove it from Electra One controller internal storage.

With public presets you can:

- Send the preset to your Electra One MIDI controller
- Get a copy of the public preset to your library, ie. clone the preset.

Preset Editor

The editor screen is divided into two sections, the sidebar, and the canvas.



The sidebar provides context information about the application and the selected element. It consists of:

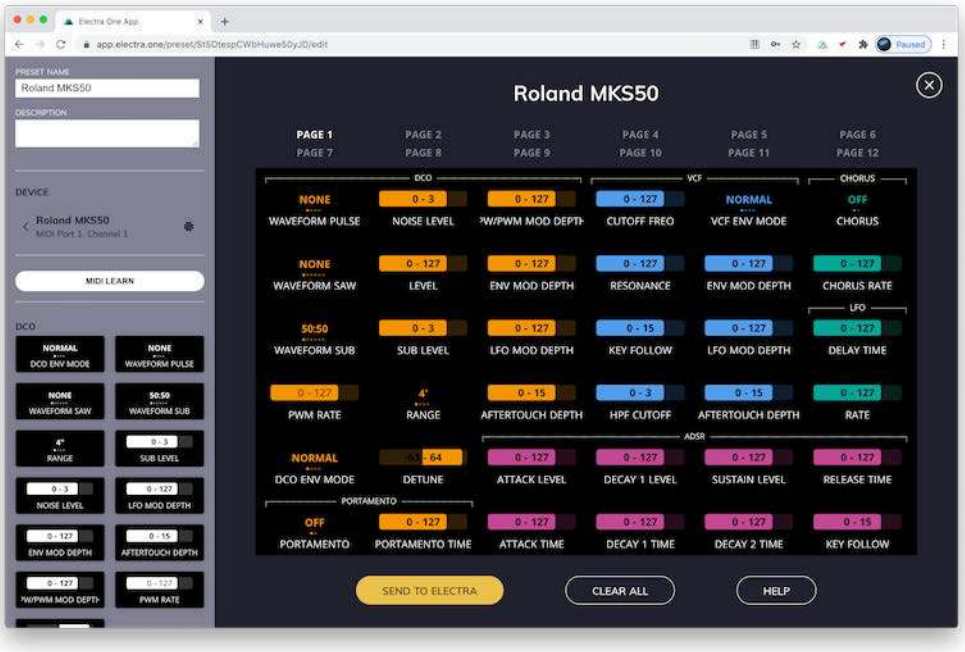
- General information about the preset
- Predefined controls
- Available devices
- Device details
- Control details

The main window consists of:

- Preset name
- Page selection
- Working grid
- Action buttons

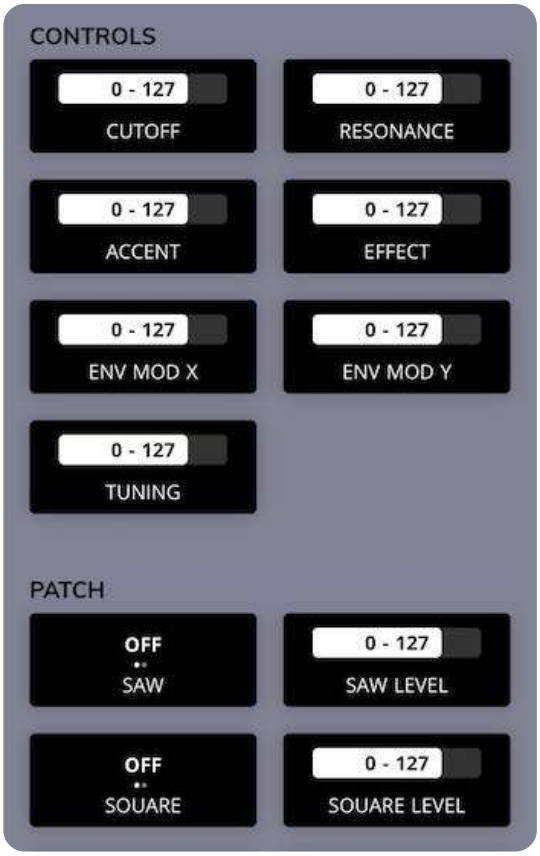
Working with controls

When you start the editor, the default sidebar with tools related to the Devices is shown. The tools provide slightly differ depending on whether a Generic MIDI device or an Integrated MIDI device (defined with the instrument file)



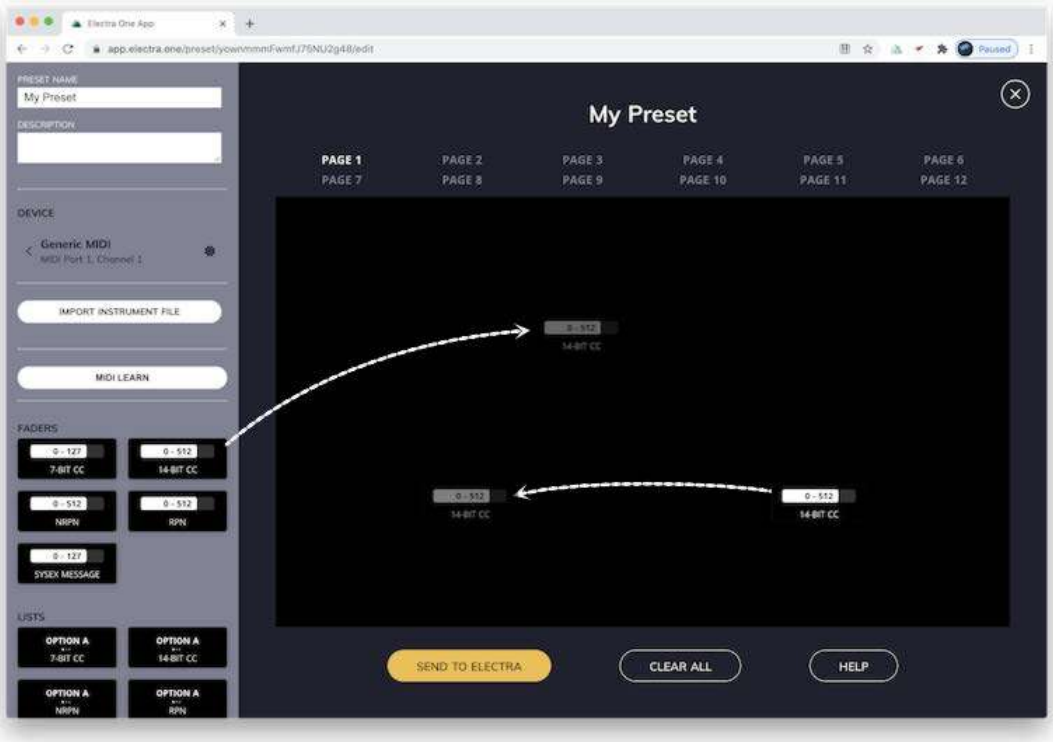
For Generic MIDI devices, a palette of controls representing common MIDI messages such as generic CC, NRPN, RPN, and SysEx messages is provided:

For integrated MIDI devices a palette of controls that reflect patch parameters of a given device is provided:



Grid

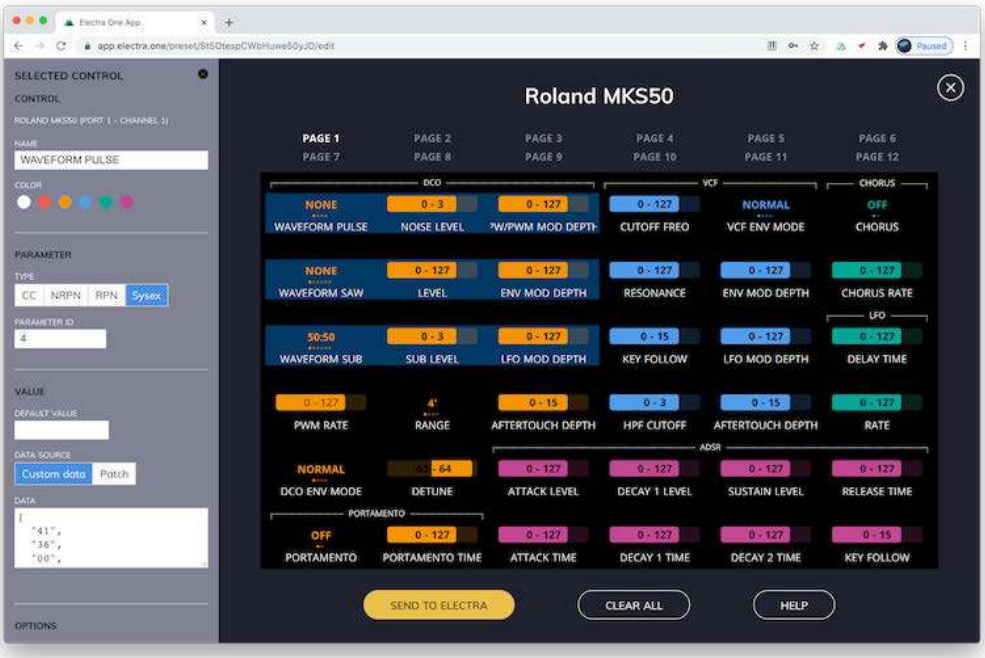
The grid mirrors Electra's display. Users place the Controls from the sidebar palette to the grid by dragging them with a mouse. Drag and drop is also used to move Controls around the grid.



The grid can be also navigated with keyboard arrows and a number of keyboard shortcut commands is supported:

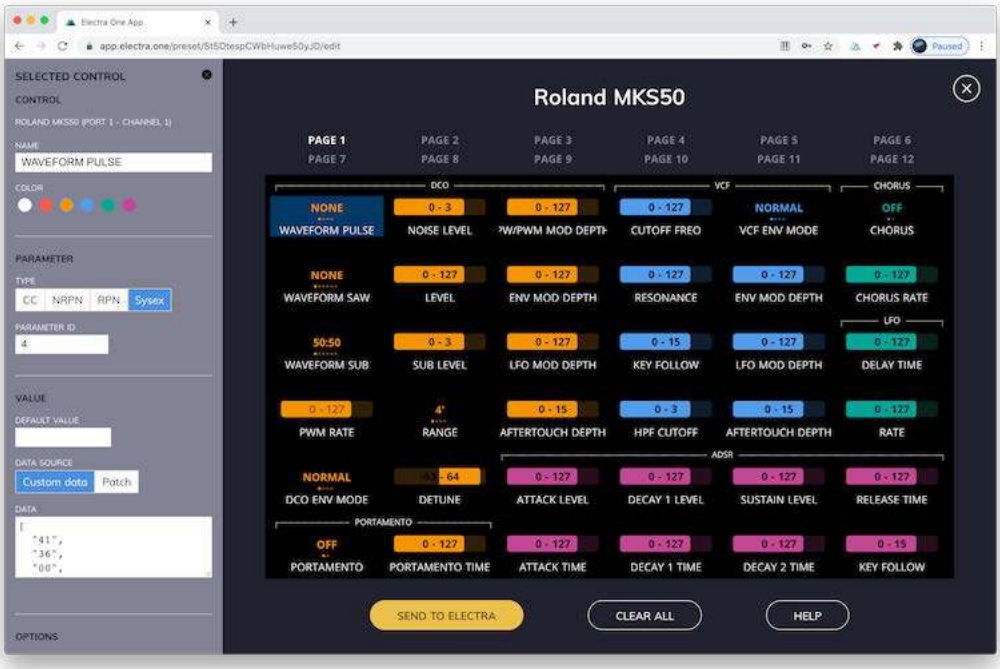
Keyboard shortcut	Action
SHIFT + mouse click	select a continuous range of Controls
CMD / CTRL + C	copy to the clipboard
CMD / CTRL + X	cut to the clipboard
CMD / CTRL + V	paste from the clipboard
BACKSPACE	remove
ARROWS	navigating within the grid
SPACE	will display information about Category and Parameter assignment for all Controls on the page

The keyboard shortcuts are extremely helpful when multiple controls are selected. This is handy when a group of controls needs to be moved around the pages.



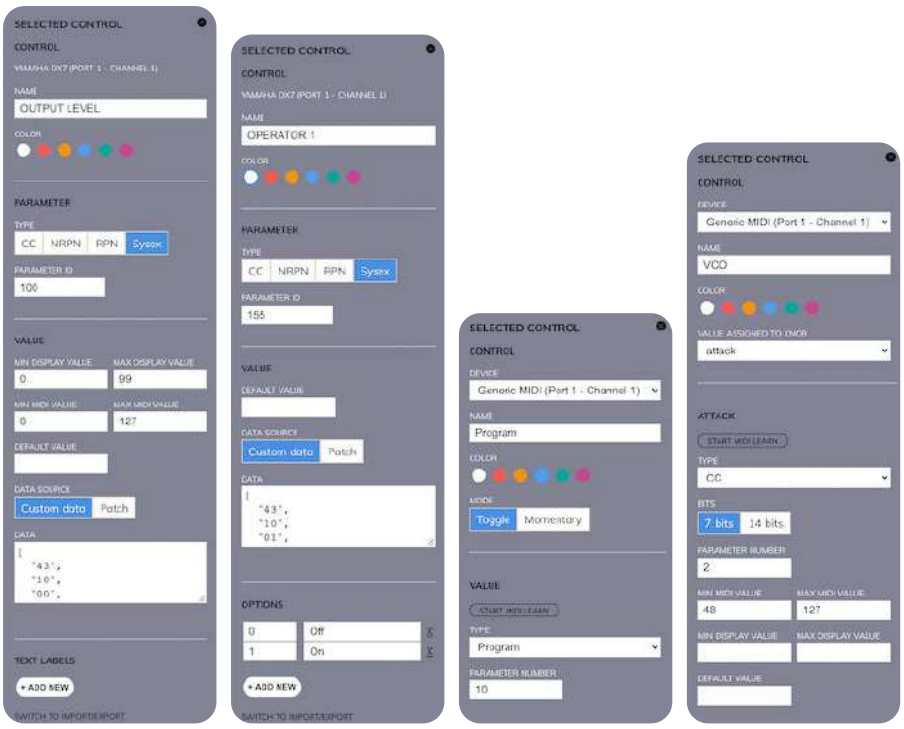
Control attributes

Each Control has several attributes to configure. Some of the attributes are common for all types of Controls, while others are available only for certain types of Controls. Control attributes are configured in Electra Editor.



The Control details sidebar is shown for selected Control. The control is selected by clicking the Control on the grid. The selected Control is highlighted with a dark-blue background.

The attributes shown in the Control details are different types of controls.



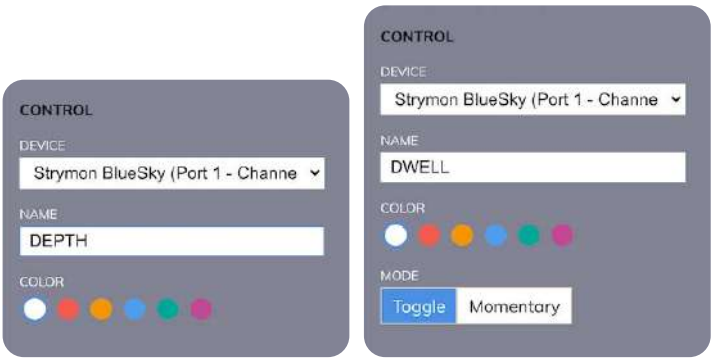
The Control attributes are divided into several groups, each covering a specific part of the Control's functionality:

- General
- Parameter
- Value
- Text values, also known as an Overlay

Multi-value controls, such as envelopes, provide these sections for all values present in the controls, ie. for ADSR envelope, there will be Attack, Decay, Sustain, and Release section.

General attributes

As the name suggests the general group of attributes covers attributes common to all types of Controls.



Device

An identification of the synthesizer, sampler, VST plugin, or any other MIDI device where the MIDI messages generated by the Control MIDI messages will be sent to and the MIDI messages the Control listens to. An example of a device is "Yamaha DX7, Rack 1". The device represents a particular synthesizer connected to a MIDI port and channel.

Name

The name of the Control that will be shown on the display. The name is shown below the

value. For example a “Filter cutoff”

Color

Color of the Control. It is up to the user how the colors are used. They are meant to improve the readability of the presets and to help to organize controls to logical groups and clusters of parameters. For example, users might want to have all Controls of one device to share the same color or to have one color for all parameters related to the VCF parameters.

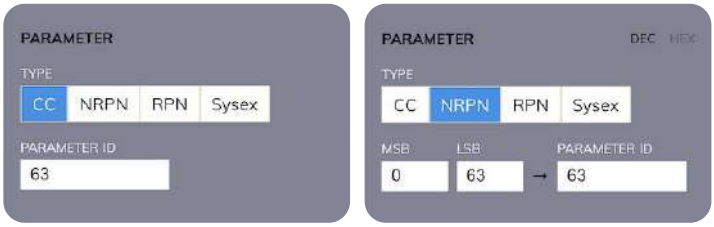
Mode

The mode is applicable only to pad controls. It instructs the control whether it returns back to the **Off** state after it is released or if it stays in the new state.

- Momentary pads always return to the **Off** position after they are released
- Toggle pads act as switches between the **Off** and **On** values

Parameter

The parameter attributes group is used to describe the MIDI message operated by the Control.



Type

The parameter type defines the type of MIDI message assigned to the Control. Whenever the value of Control is changed by turning the knob or with touch, a given MIDI message will be sent to the connected synthesizer. On the receiving side, whenever there is a MIDI message coming from a connected synthesizer and it matches a parameter settings of the Control, the value of the Control will be updated accordingly.

Electra supports the following MIDI message types:

CC

7-bit or 14-bit Control Change MIDI message. The value of 7-bit Control change is restricted to a range of 127 values. There is an option to switch CC to 14-bit Control Change mode. 14-bit Control change follows the MIDI standard which says that the first 32 7-bit control change messages (CC #0 .. CC #31) can be used as 14-bit messages. The parameter that users specify is the MSB part of the control change, LSB part is automatically calculated by Electra. It is always MSB parameter + 32.

NRPN

NRPM MIDI message type is used to send a standard MIDI NRPN message. The parameter and the value are both 14-bit numbers.

RPN

RPM MIDI message type is used to send a standard MIDI RPN message. The parameter and the value are both 14-bit numbers.

SYSEX

SysEx MIDI message type is used to send templated MIDI SysEx messages. Users are allowed to specify an array of bytes that will be sent whenever the Control’s value is changed. The fact that the message is **templated** means that users are not restricted to sending constant bytes only, instead, they can insert Variable, Checksum, Parameter placeholders to the message. The placeholders will be transformed to values at the time of sending the templated SysEx MIDI message. More detailed information about SysEx templates can be found in Writing SysEx templates.

NOTE

NOTE is used to send note on and off MIDI messages. The note type is supported only by pads. The note on is triggered when pad is pressed and note off is send when the pad is released.

PROGRAM

PROGRAM type sends a standard MIDI Program change message. The program can be used only with pads.

START

START type sends a standard MIDI real-time Start message. The start can be used only with pads.

STOP

STOP type sends a standard MIDI real-time Stop message. The stop can be used only with pads.

TUNE

TUNE type sends a standard MIDI Tune request message. The tune can be used only with pads.

Parameter Id

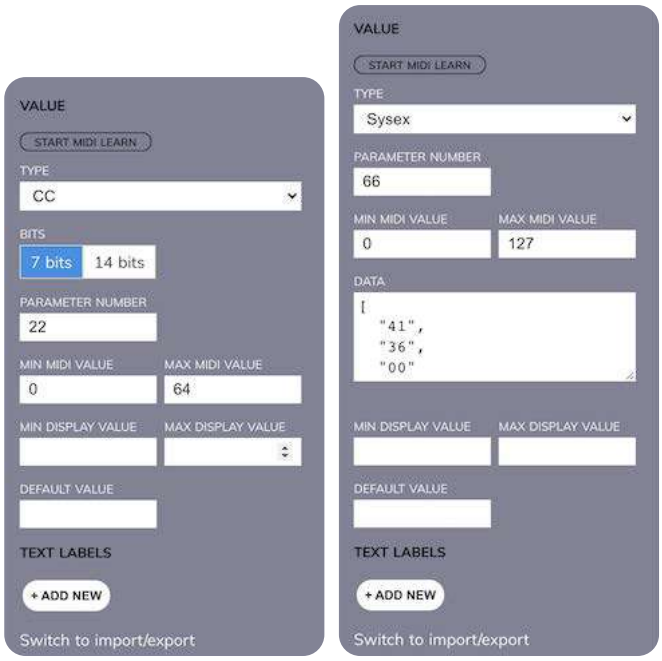
Identifier of the parameter to be assigned to the Control. When control is used to send CC MIDI message and parameter is set to 56, the value of the Control will be sent as CC #56 MIDI message. There are situations when there is no real parameter, for example, if the parameter is represented only by a few bits with a byte of a SysEx message. In such situations, users must invent their own Parameter Ids. More information on this can be found in Writing SysEx templates.

When working with NRPN and RPN Controls, the MSB and LSB fields are shown. These fields make it easier to enter the parameter Id if the synthesizer manual uses MSB and LSB notation. The MSB and LSB must be entered in decimal notation.

Value

The value attributes group tells Electra how the values should be handled and how to translate MIDI values to values shown on the display.

Fader values



List values

Envelope values

Pad values

Min display value

Defines the minimum value of the data range controlled with a fader. The minimum may be negative.

Max display value

Defines the maximum value of the data range controlled with a fader.

Min MIDI value

Defines the minimum midi value mapped to the Min display value. A typical example is mapping Min MIDI value of 0 to the Min display value of -64. In such a case, the Control will show negative figures while it will still send out positive data in MIDI messages according to this setting.

Max MIDI value

Defines the maximum MIDI value mapped to the Max display value.

Default value

A default value to be pre-filled when a preset is loaded. The default value is set to 0 when not filled in. Double-tap on the Control will reset the current value of the Control to the default value.

On Value

A MIDI value to be sent when the pad goes from the **Off** state to the **On** state. If the field is left empty, no MIDI message will be sent.

Off Value

A MIDI value to be sent when the pad goes from the **On** state to the **Off** state. If the field is left empty, no MIDI message will be sent.

Default state

Tells Electra if the pad is set to **On** or **Off** state, when the preset is loaded.

Bits

When the control is set up to send CC MIDI messages, you can specify whether it will be a simple 7-bit message or a two-byte 14-bit message.

Bits order

14-bit CC, NRPN, and RPN MIDI messages send the value of the parameter in the form of two 7-bit bytes. These two bytes represent MSB (most significant/coarse) and LSB (least significant/fine) part of the 14-bit value. Although the MIDI standard says what part is the MSB and what the LSB is, some synthesizers do not follow the standard. Bits order option gives you a chance to swap MSB and LSB part of the 14-bit value.

Two's complement

When the display value configuration allows going below zero to negative values, the Two's complement option tells Electra One controller if two's complement representation of the negative numbers should be used.

No reset

It has become quite a common practice that each NRPN or RPN message is followed by the Reset instructions (sending CC #100 and CC #101). If this is not appropriate for your instrument, set No reset to TRUE.

Data source

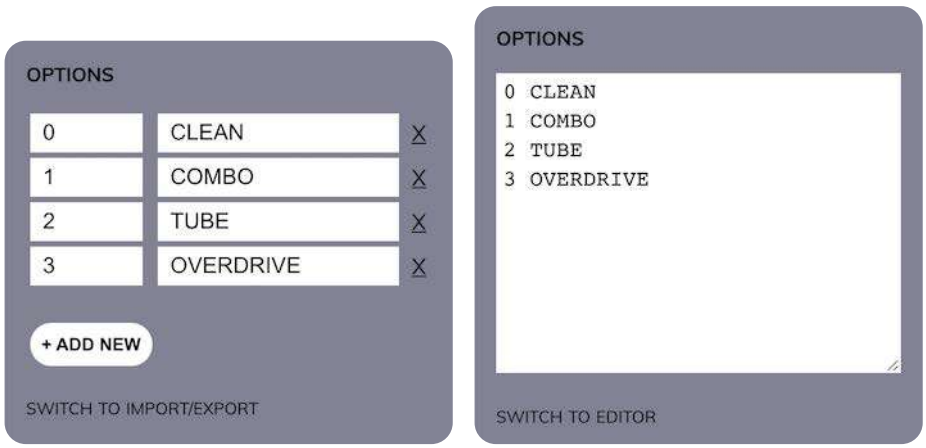
In the majority of cases, the Custom data does the job. The Custom data means that the SysEx template defined in the Data attribute is a single message sent whenever Control's value changes. The Patch, on contrary, is an option that instructs Electra to send the whole Patch definition to the Device when the Control's value changes. As Patch is a rather advanced option, it will be described in a separate document.

Data

The data field is available for SysEx Controls. The field allows you to enter SysEx bytes and variables in the form of a JSON document. Detailed description of the JSON format can be found at Tutorial on Writing SysEx templates.

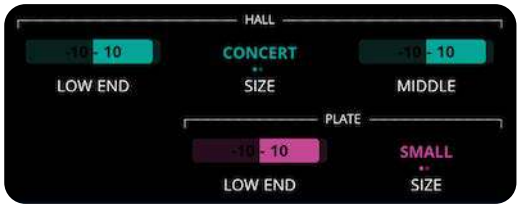
Text labels / Overlays

Overlays are text labels that can be assigned to specific values. Overlays are required to be used with List Controls. They define the list items. Overlays may, however, be also used with faders. In that case, the text value will be displayed on the fader instead of a numeric value. This is often used for some special cases, where for example a maximum value of a fader is **infinite**, or when the parameter value is a mixture of both continuous value and a few discrete values.

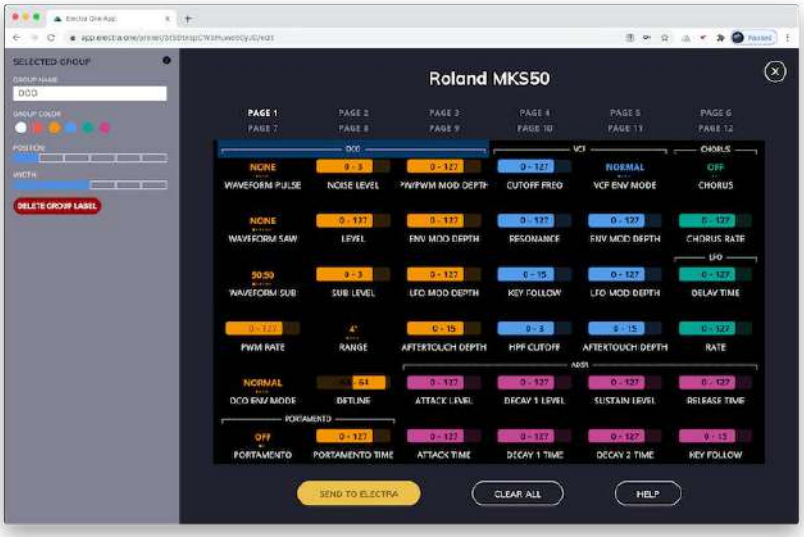


Groups

Groups are meant to improve the layout of the preset and give it more structure. They can be used to make collections of Controls that are related to a specific type of parameters, for example, parameters of VCF section. Groups, however, do not impose any functionality. It is fully up to users who the groups will be used.



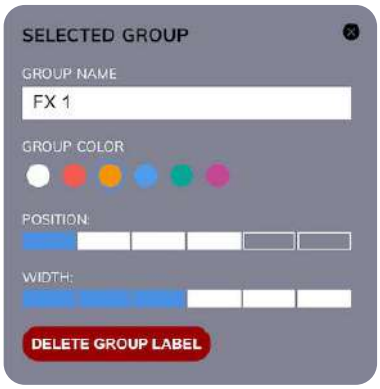
To add a group, click on the group placeholder on the grid, the group details sidebar will be shown.



Each group can be customized to suit your needs. The following can be set:

- Name of the group
- Color
- Horizontal position within the grid
- Width of the group

You can also use the Group details to remove an existing group.

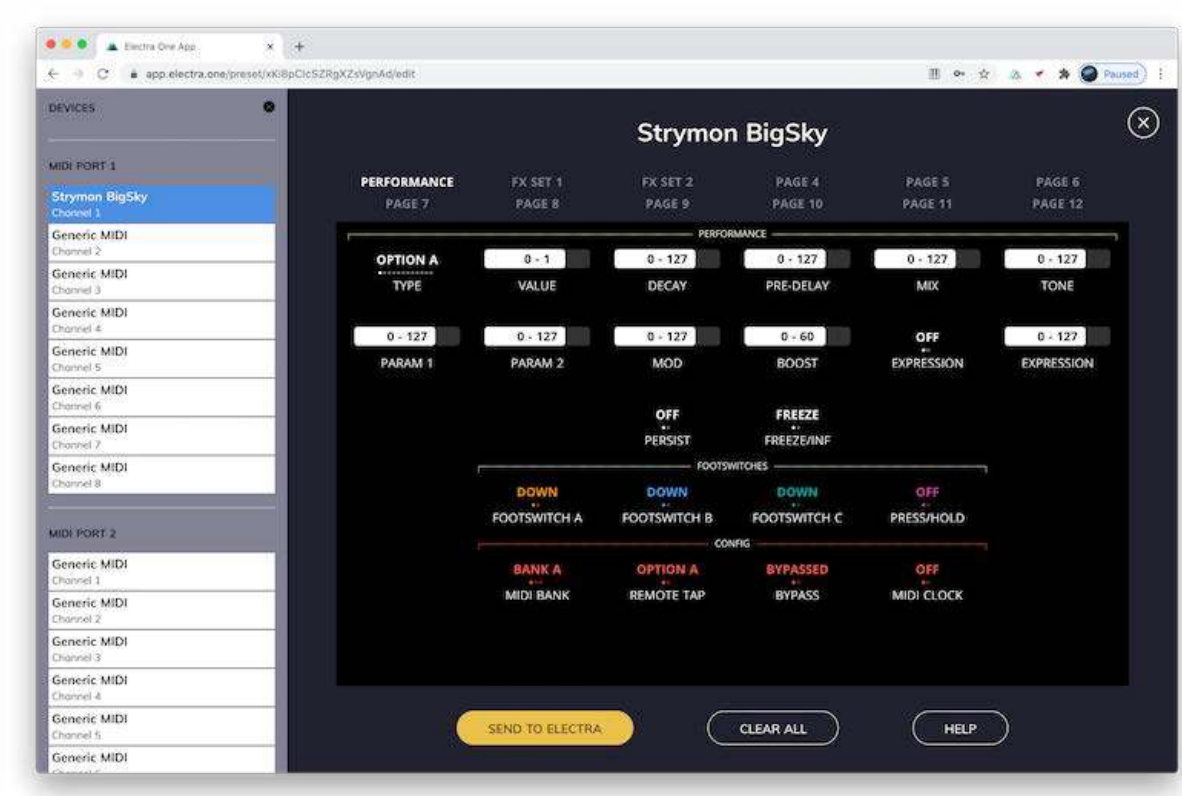


Devices

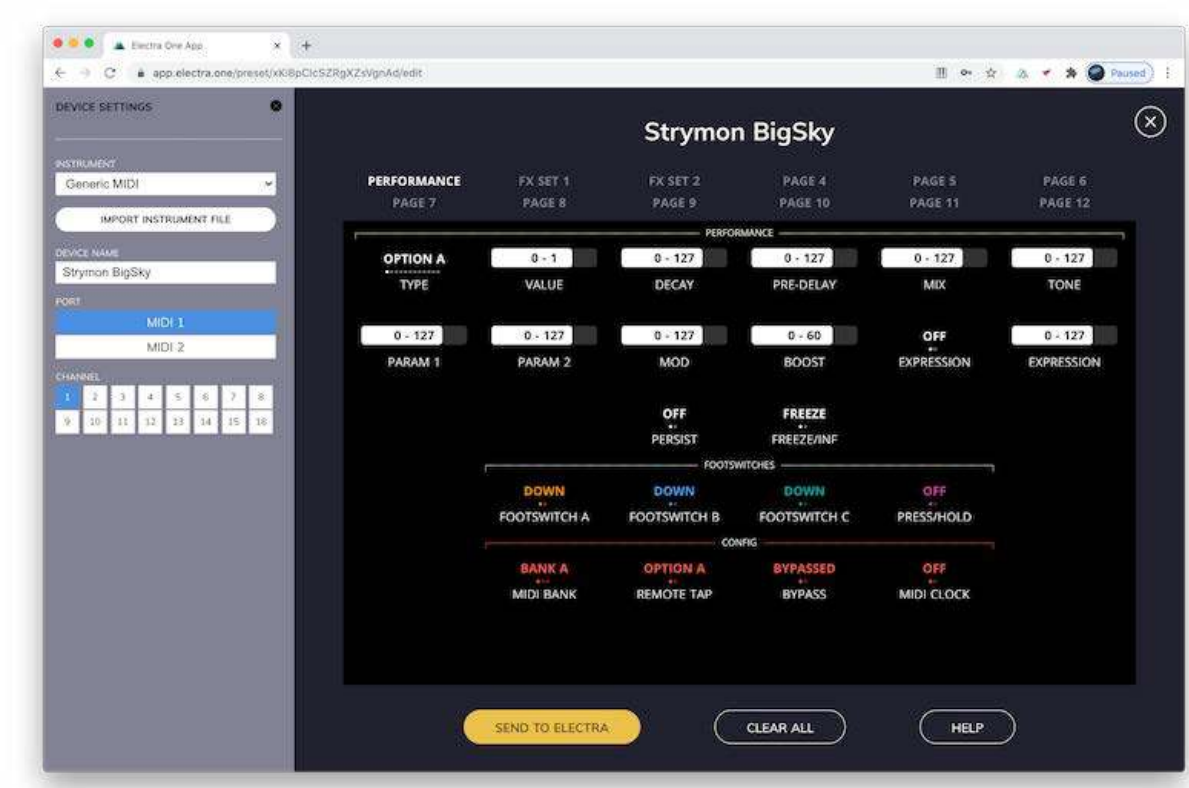
As it is important to understand the concept of devices, their meaning was described in a separate chapter of this User Guide. The following paragraphs just describe how to manage devices.

The new controls are always picked in the context of a Device. The currently chosen device can be seen in the main sidebar. Any Control you pick and place on the grid will be always associated with that device.

If your preset uses more than one device, you can switch between them, by clicking the Device in the sidebar. A list of all available devices will be shown. You can choose the Device that you want to work with. Once you make your choice the sidebar will show a palette of controls for the selected Device. Any Control that you drag on the grid now will be associated with that Device.



If you need to adjust the settings of a device, for example when the MIDI channel of the device was changed, click the Device settings icon (cog symbol). The Device details will be shown in the sidebar.



You can set here:

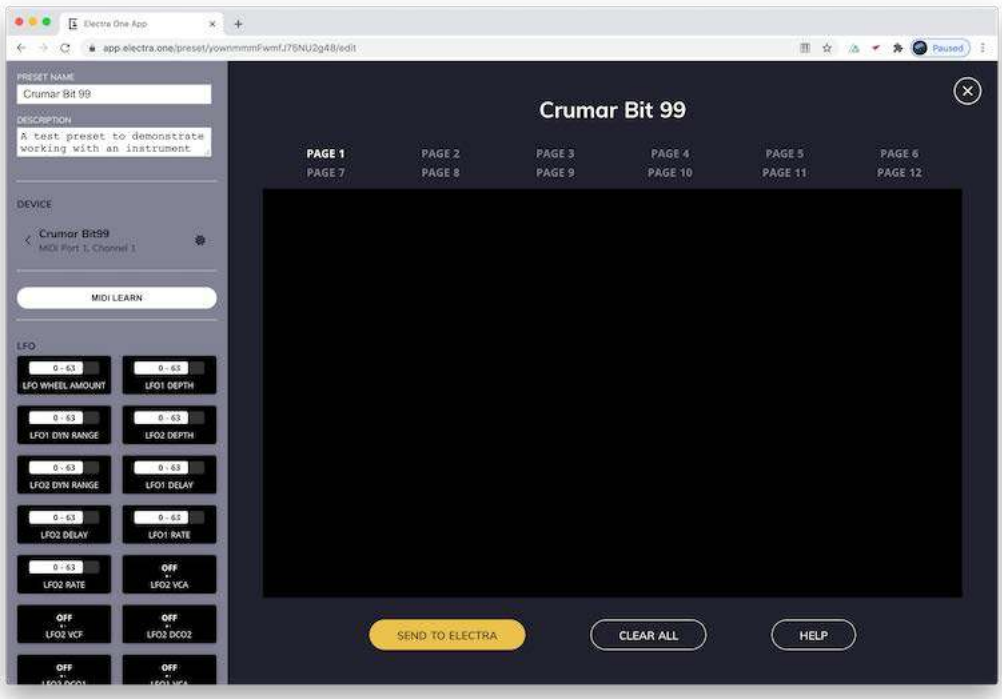
- Name of the Device
- MIDI port
- MIDI channel
- Type of device from the list of predefined instruments
- Load a custom Instrument file

Import an Instrument file

Although the Electra One Instrument Files are normally imported in the Device details sidebar, this button provides a quick way to upload an Instrument file.

An Instrument file is a file containing a pre-defined set of Controls for a given model of a synthesizer or any other MIDI device.

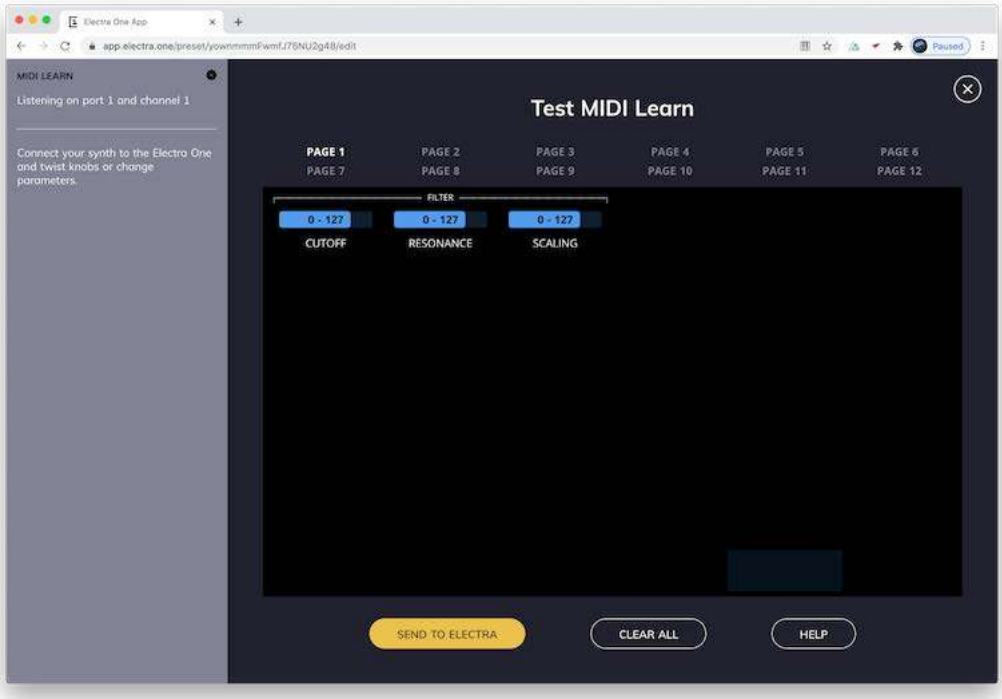
Loading an instrument file will populate the sidebar with a palette of Controls related to the given instrument. Work on the presets is greatly simplified when the instrument file is available. Simply because the tedious work of defining MIDI messages, value ranges, text labels, and patch definitions is already done for you. The picture below shows Preset editor with Crumar Bit 99 Instrument file loaded:



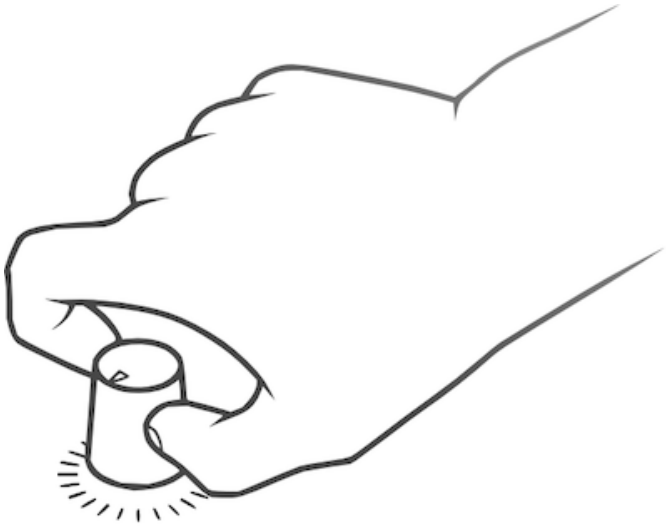
MIDI learn

The MIDI learn function is another tool to make preset creation less tedious. When the MIDI learn is activated.

Electra One MIDI Control is switched to a special mode when it reports all incoming messages to the editor. If the MIDI message matches currently selected devices (MIDI port and channel), a new Control reflecting the incoming MIDI message is created and shown in the sidebar.



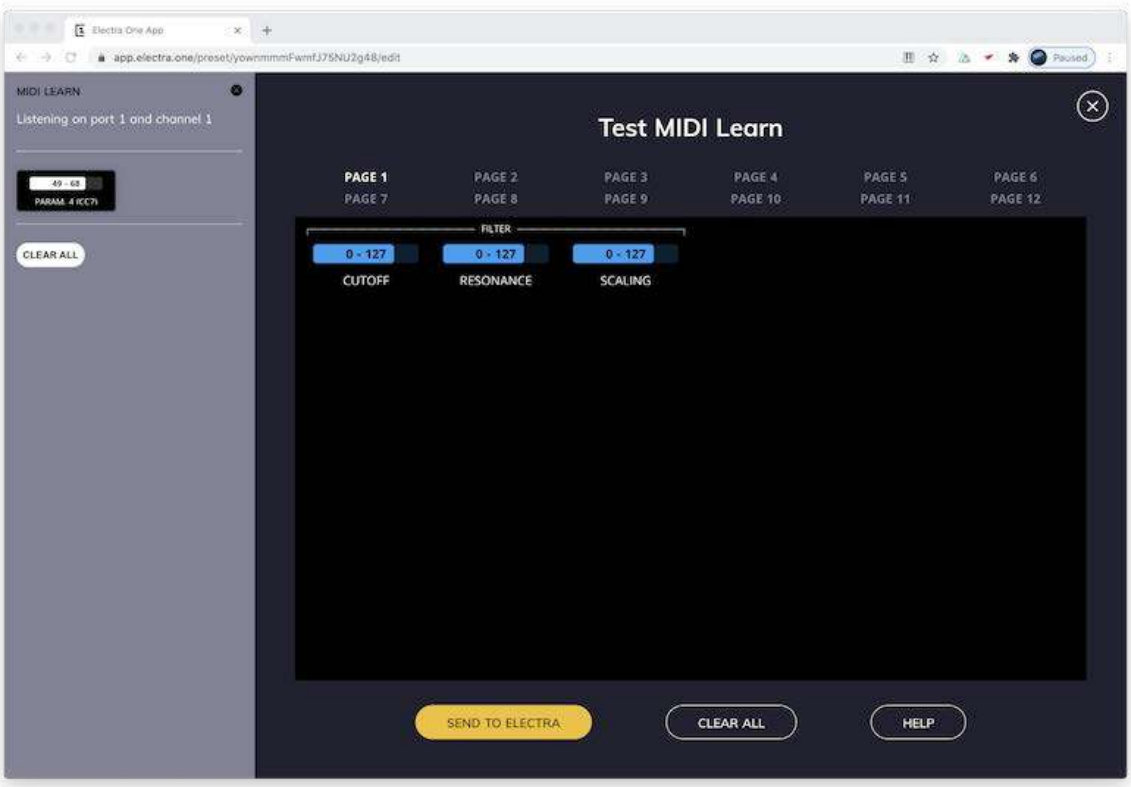
The best approach to let Electra and the Editor to understand the MIDI implementation of your instrument, is to twist the knobs of the instrument for various parameters.



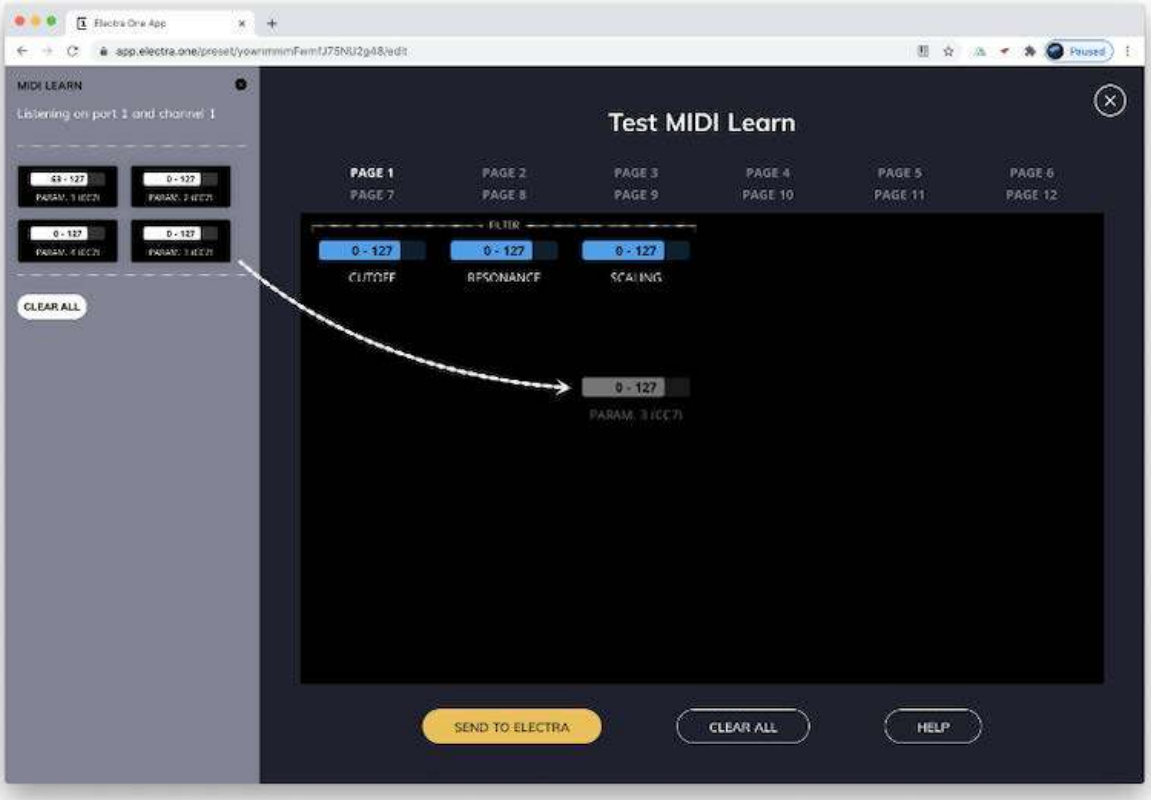
This way a Control is created for each parameter and shown in the sidebar. The MIDI learn does not detect MIDI message type and parameter number only, it can also detect the minimum and the maximum MIDI value.

TIP

Always try to twist the knob all the way to the left and then to the right. This way Electra will be able to detect the full range of the values of the given parameter.



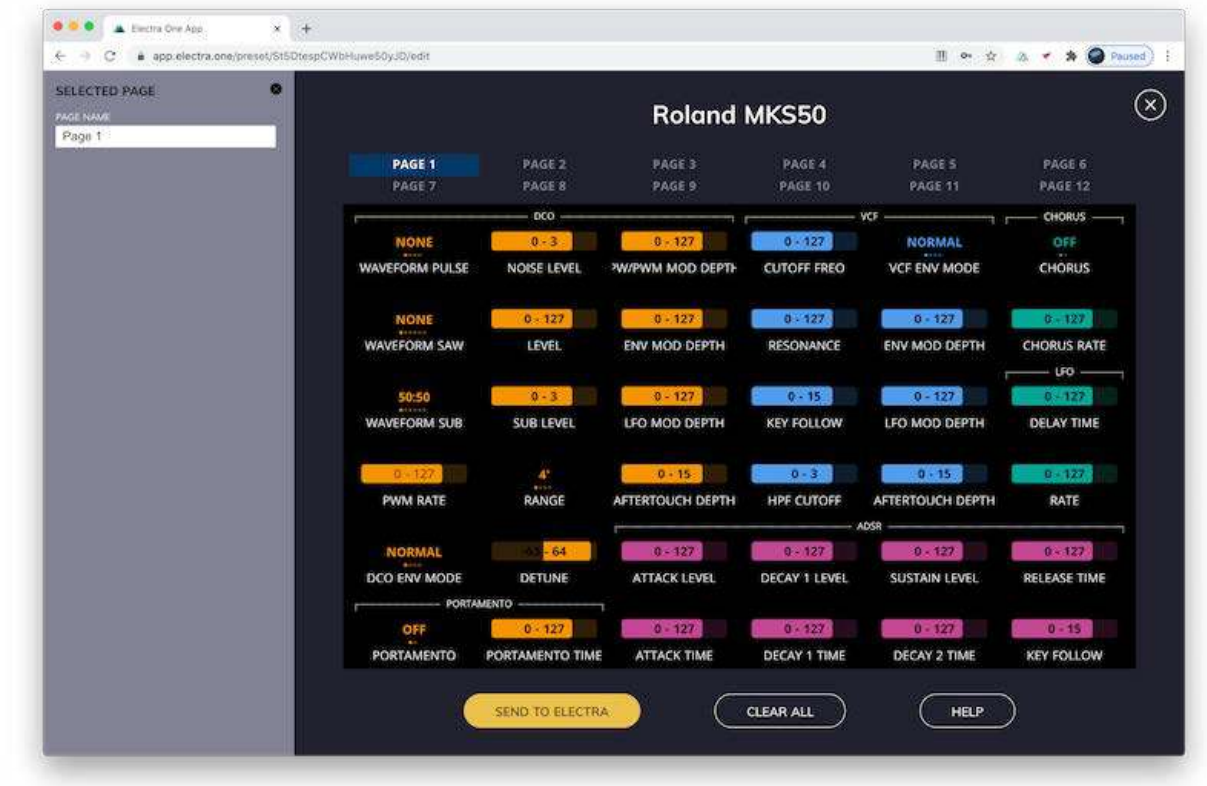
The Controls residing in the MIDI learn sidebar can be dragged and dropped to the grid as any other Control. The `CLEAR` button removes all learned Controls from the sidebar so that you can start the process of learning all over again.



Page selection

Pages are an important element of Electra. They multiply the number of available controls. It is up to the user how the pages will be used. They do not provide any other function, they are merely a tool to organize the controls within the preset. Usually, users use pages to hold sets of related Controls. Other uses are possible too, for example, a page may represent a set of Controls for each song or a scene of your performance.

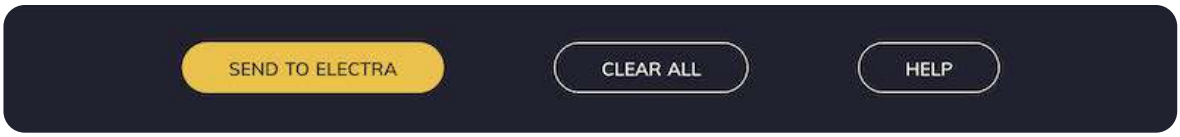
The Page selection at the top of the editor allows users to switch between the pages. Clicking on the page name will switch the active page and the Page detail will be shown in the sidebar. The Page detail allows you to rename the page.



Preset actions

There are two actions you can do with a preset. You can:

- Send it to Electra One MIDI Controller
- You can clear all its contents



Preset slots

The `SEND TO ELECTRA` button initiates the transfer of the preset to the currently selected preset slot in the controller. It means you should select the preset slot on the hardware controller first. If you fail to do so, you may accidentally overwrite a previously stored preset.

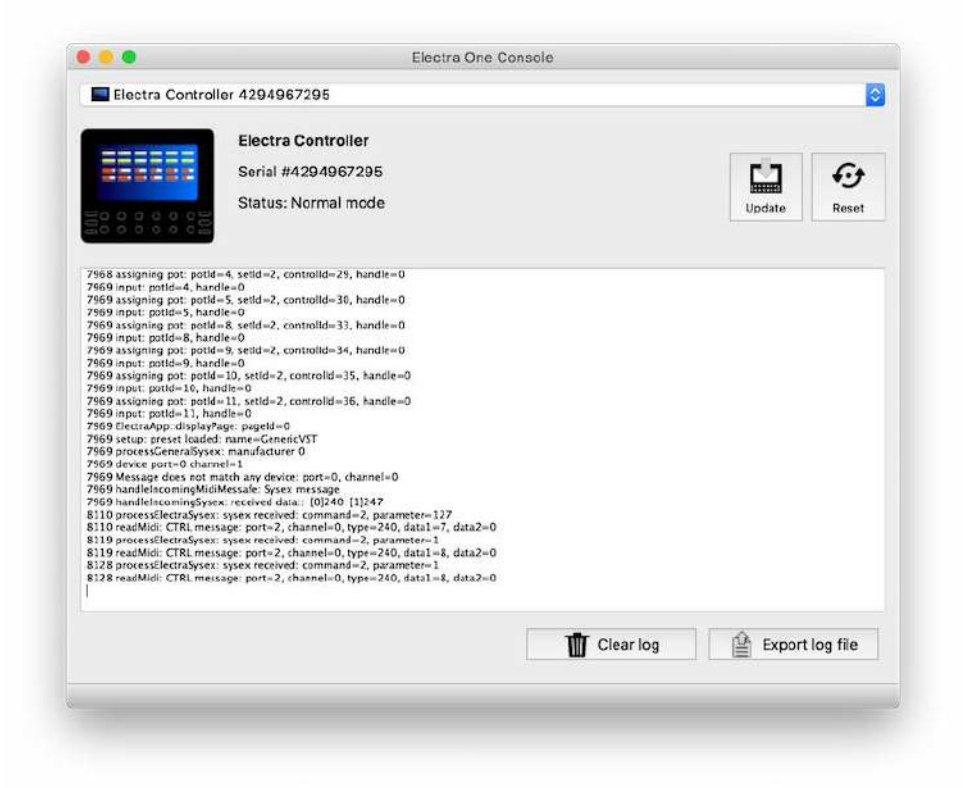
←External MIDI control ElectraOne Console ←

ElectraOne Console

ElectraOneConsole is a management tool to update Electra's firmware and review internal debugging logs.

TIP

We encourage all Electra owners to install the ElectraOneConsole and use it to update firmware frequently. The firmware updates are safe, fast, and easy.



There is a number of actions supported:

Update

The `Update` button lets you choose and upload a new firmware file to Electra One MIDI controller. Under normal circumstances the firmware update is fully facilitated by the application, there is no need to switch Electra One to the update mode.

Reset

The `Reset` button initiates a soft reset of the MIDI controller. You can use it to restart Electra if you feel it does not work properly.

Clear

The `Clear` button removes all debugging messages from the Log window.

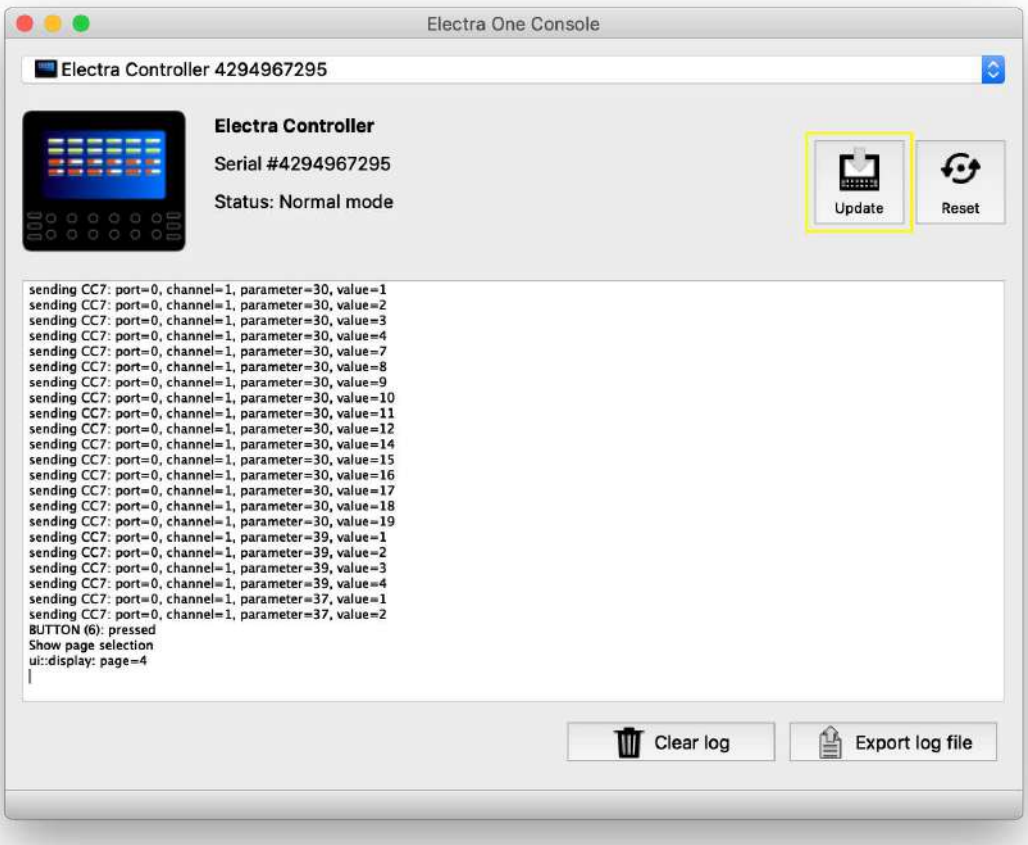
Export log file

The `Export log file` exports the contents of the Log window to a file. Exported log files are a great help for us (Electra developers) to understand possible software defects in your firmware.

Firmware update

In order to update the firmware of your Electra take the following steps:

- Download ElectraOneConsole application on your computer from the App downloads page
- Download the latest firmware from the Firmware downloads page
- Unzip the file. You need to have a file with .frm extension
- Connect Electra to your computer
- Run the Electra One Console application
- Click the "Update" button in the application, choose your firmware file, and confirm.



- Electra firmware update will be performed. Wait until Electra boots up and the Electra One Console indicates **Normal Mode** again. You can verify the firmware has been updated on Electra's start-up screen.

Troubleshooting

ELECTRA ONE

Firmware recovery

Problem description

- Your Electra is not reacting and it seems to be frozen
- There is a blank / black screen when powered on, nothing else happens.
- It is not possible to upload new firmware.

Even though we are working hard to make the firmware stable, there are occasional system freezes. This article is meant to help you to resolve the problems that it might bring. The most common situation when you need to perform the hard restart is a bricked Electra after a failed firmware update.

Recovering from a system freeze

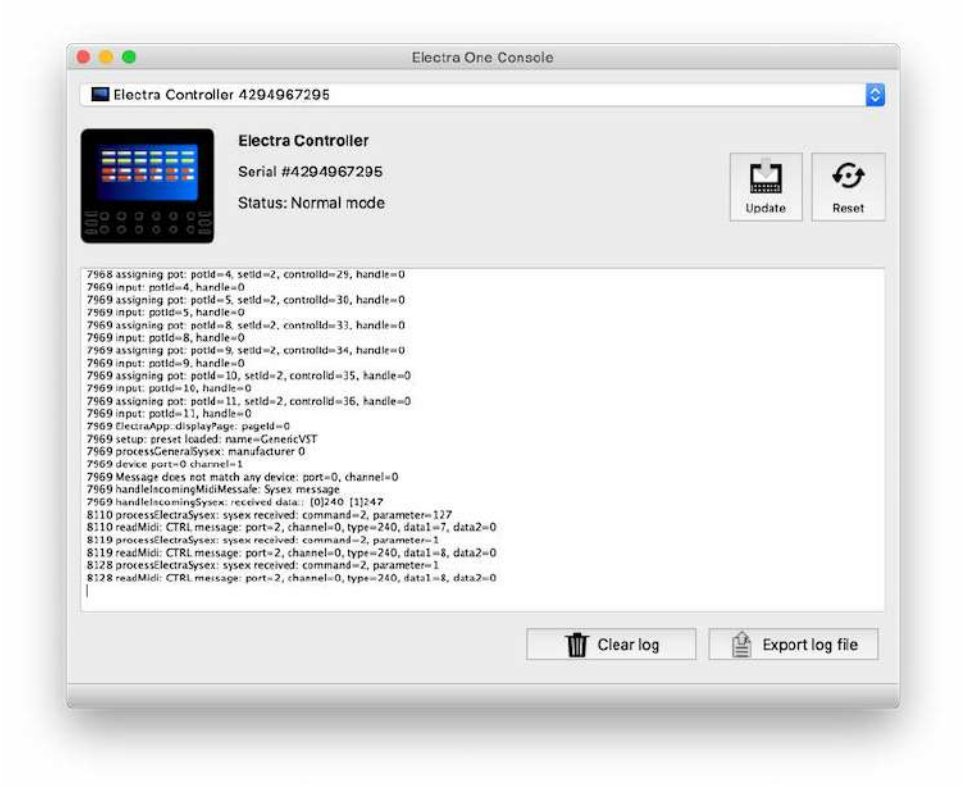
Simple restart / reset

Usually, it is enough to disconnect Electra One and connect it again. Another option is pressing the Reset button on the rear side of the controller. To do so, you will need a pen or some think stick. The reset button is located on the back panel next to the USB device port. If you have ElectraOneConsole application open, you can restart Electra from there by clicking the `Restart` button.

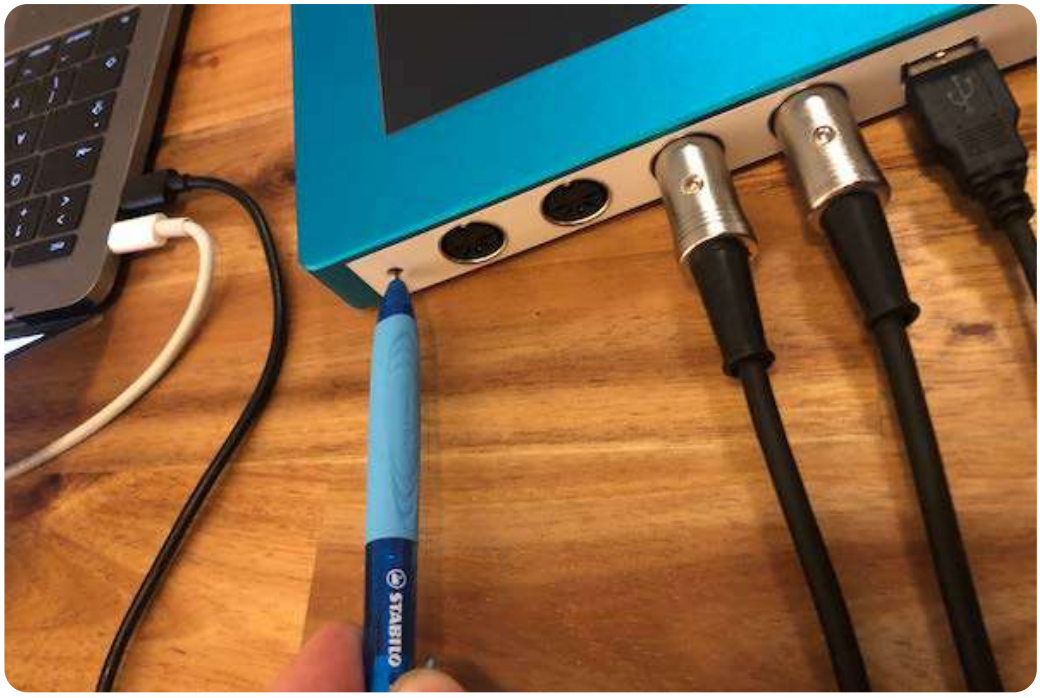
Hard reset procedure

If your Electra is not booting up and you are unable to load a new firmware using the standard update procedure, take the following steps to get your Electra working again:

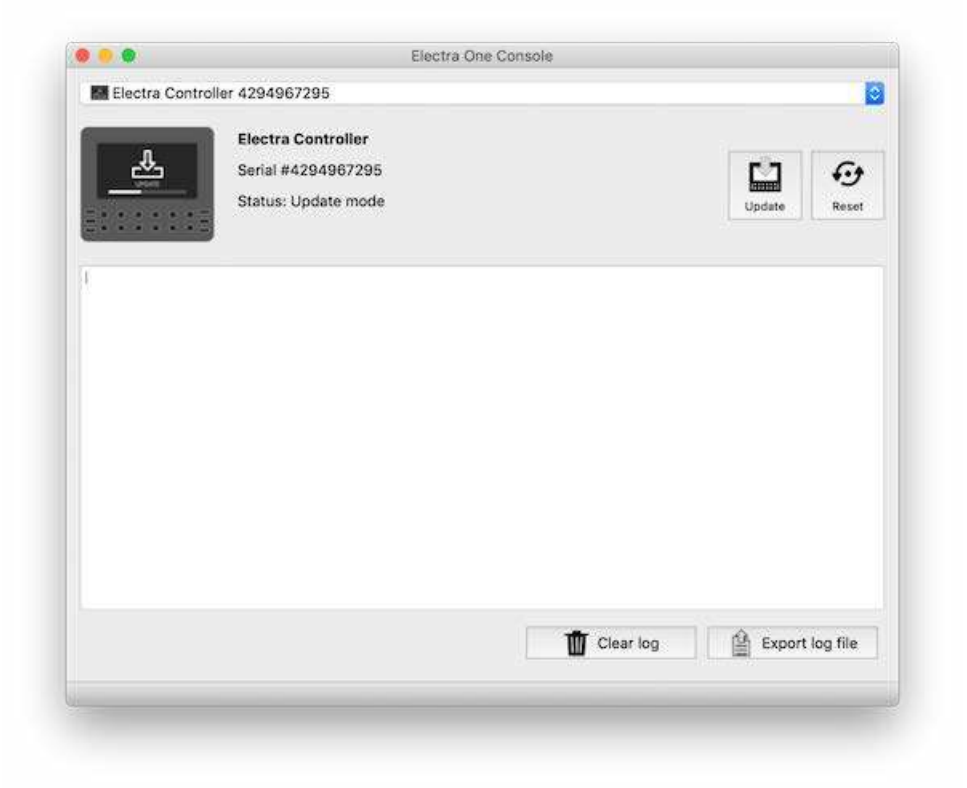
1. Disconnect Electra One MIDI controller from the computer.
2. Start Electra One Console application



3. Press the Update button on the back panel and hold it pressed



4. While you keep the button pressed, connect Electra to the USB port of a computer where ElectraOneConsole application is running.
5. Keep holding the button pressed for at least 10 seconds.
6. Release the button.
7. The ElectraOneConsole should announce that Electra is ready for flashing the firmware



8. Use the Upload button in the application to upload firmware to the Electra One MIDI controller.
9. After the firmware is uploaded, Electra will re-restart and should act normally.

ELECTRA ONE

USB connection issues

Problem description

- The connection indicator in the Electra App is not green
- Electra App and/or the Electra Editor do not communicate with Electra One
- You can send presets to Electra One, but you cannot read them

This article describes steps to take to resolve problems with communication with the computer and the Electra One MIDI controller.

Electra USB MIDI ports

Electra One MIDI controller has three distinct USB MIDI ports. They are named:

- `Electra Port 1`
- `Electra Port 2`
- `Electra CTRL`

These ports are implemented as virtual MIDI ports (sometimes called cables) of one USB MIDI device `Electra Controller`. Whenever you connect the Electra One MIDI controller to a computer, the `Electra Controller` USB MIDI device should be registered by the operating system.

Even though the port names are set and configured properly within the USB device descriptor, some versions of operating systems do not recognize it correctly, please see the details below.

If your Electra does not seem to communicate properly with Electra Editor, you need to check two important things:

- Is your system / your browser using the correct MIDI port for the communication?
- Are the message MIDI messages exchanged between Electra controller and Electra Editor?

Browser compatibility

A common problem with USB MIDI communication comes out the fact that users do not use a browser with WebMIDI support.

Chrome or Edge browsers required to run Electra App account and Electra Editor applications. Other browsers do not support WebMIDI standard yet and therefore fail to communicate with the Electra One MIDI controller.

Verifying USB MIDI Ports

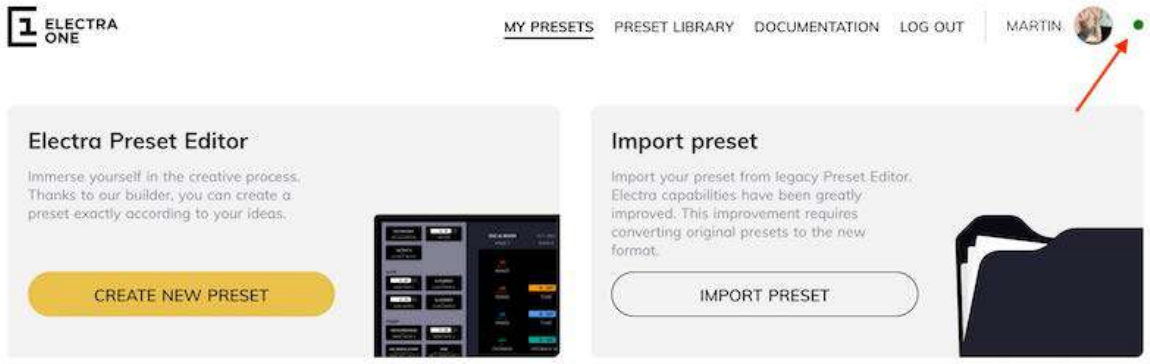
As described above, the Electra One MIDI controller has three different USB MIDI ports.

Ports `Electra Port 1` and `Electra Port 2` are dedicated to the communication with your gear, DAW, and software plugins.

Port `Electra CTRL` is solely used for communication between Electra One MIDI controller and the Electra App and Electra Editor web applications. A common reason for problems with USB communication is caused by Electra App not being configured to use `Electra CTRL` port.

Identify your Electra port names

The Electra editor has a connection status indicator. It is next to the Electra One editor title in the sidebar.

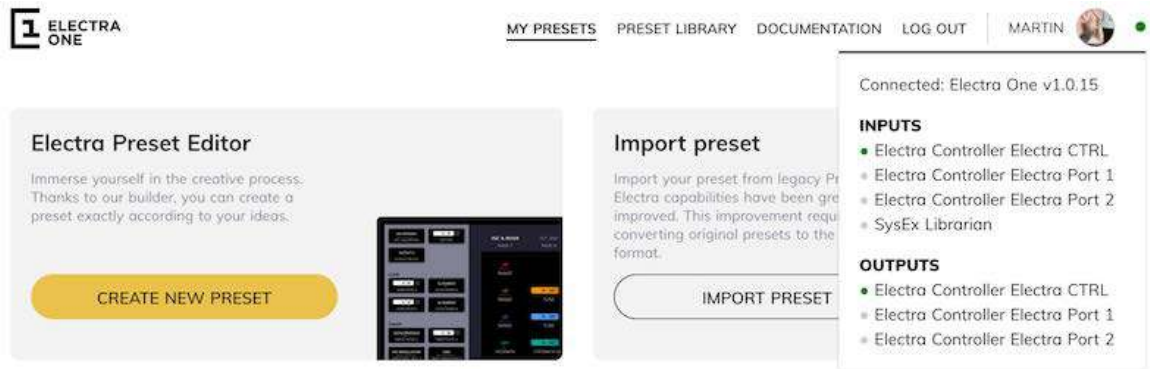


When the indicator is grey, the Electra App account cannot see any of Electra standard MIDI ports and it is unable to establish the communication over USB.

This can be caused by two reasons:

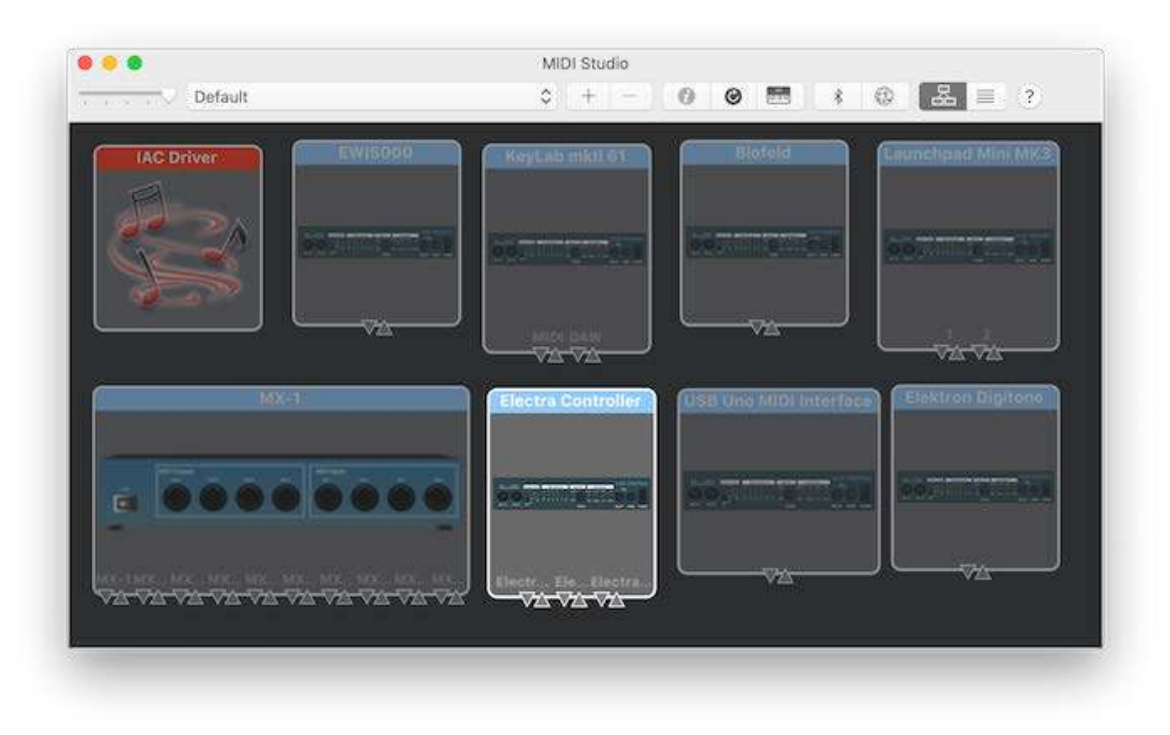
- Electra is not connected or detected correctly by your operating system
- The names of the ports are not picked up correctly or differ from expected port names

When the **indicator is clicked** the list of available MIDI ports is shown. Normally you should see the following ports:



Alternatively, you can take a look at the list of MIDI ports in your DAW or other application:

or in the MIDI Studio on MacOS X:



Changing MIDI port for communication with the Editor

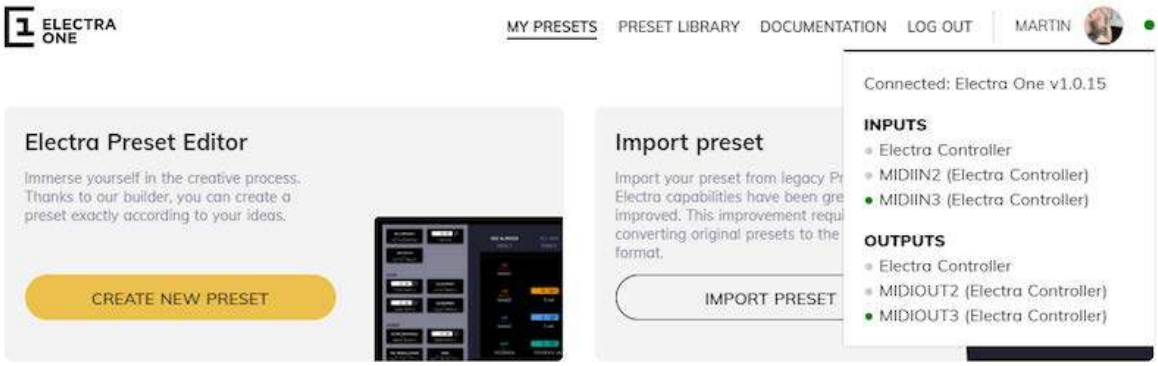
The list of MIDI ports in the Editor sidebar can be used to switch the port that will be used for communication with the Electra One MIDI controller.

To do so, simply click the any of available inputs or outputs. When the **Electra CTRL** input and output is not present, it is usually listed there as:

- **Port 3**
- **MIDIIN3** / **MIDIOUT3**.

If not, please contact us and send us your list of ports, we will review it for you and will come back with the correct settings for your setup.

The screenshot below shows how to switch the MIDI ports when they are not picked up automatically. Also, it is a valid configuration for Windows 10, when virtual port names are not recognized correctly:



Tutorials

AU/VST control in Ableton

Goal of the tutorial

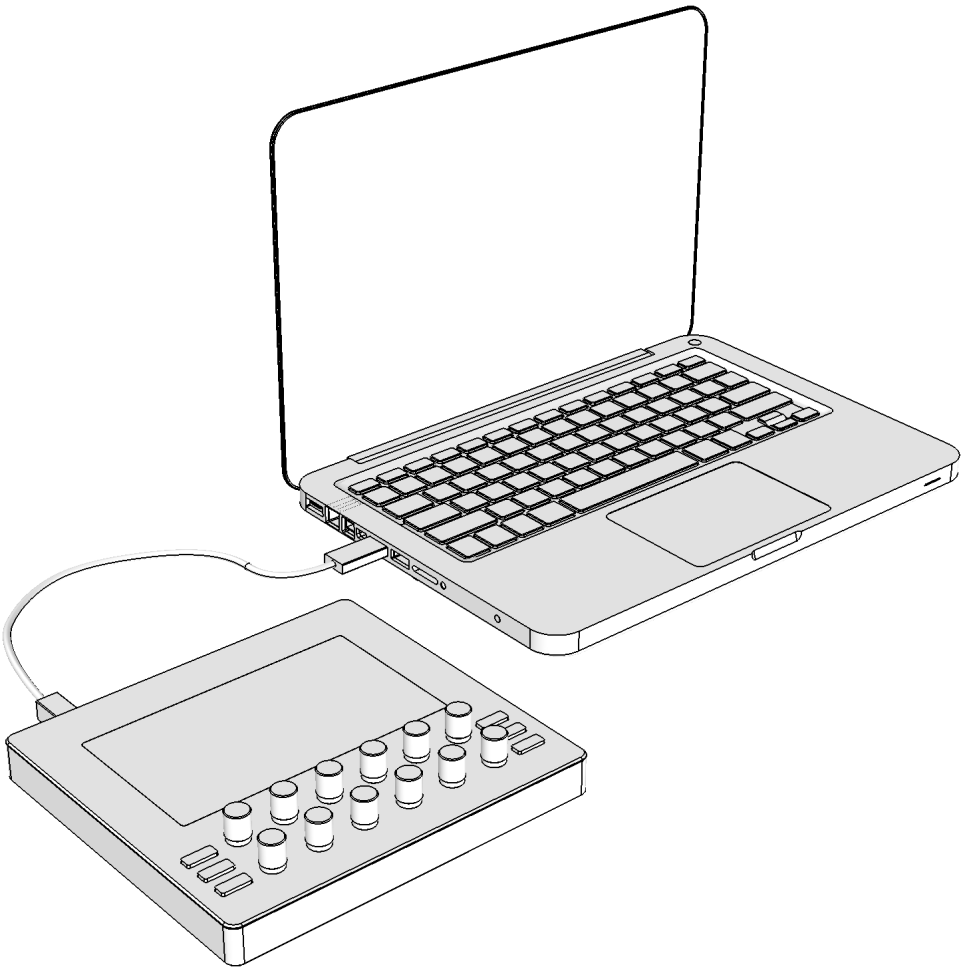
In this article, we will look at Ableton Live 10 and you will learn how to:

- control track parameters
- control VST plugin parameters
- keep Electra's Controls synced with AU/VST plugin parameters

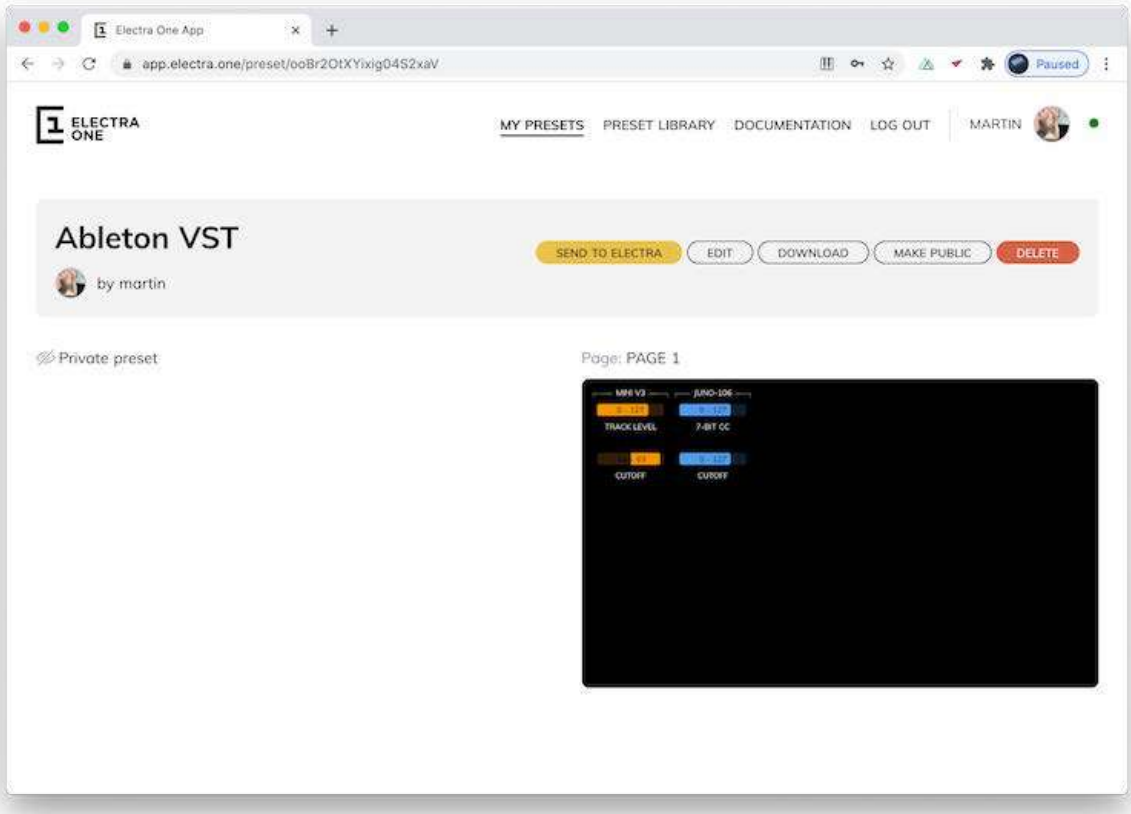
Electra One is fully compatible with Ableton's remote parameter control. The MIDI controller allows you to control parameters of almost everything in the Ableton as well as parameters of AU/VST plugins.

1. Load the demo preset

Connect Electra One to your computer's USB port. Open Chrome or Edge browser and go to https://app.electra.one/ and sign in to the Electra App.



Go to the `Preset library` and choose the Ableton VST demo preset. Click `SEND TO ELECTRA` to upload the preset to Electra One controller.



The preset has four Controls to control two parameters of Ableton and two parameters of AU/VST plugins.

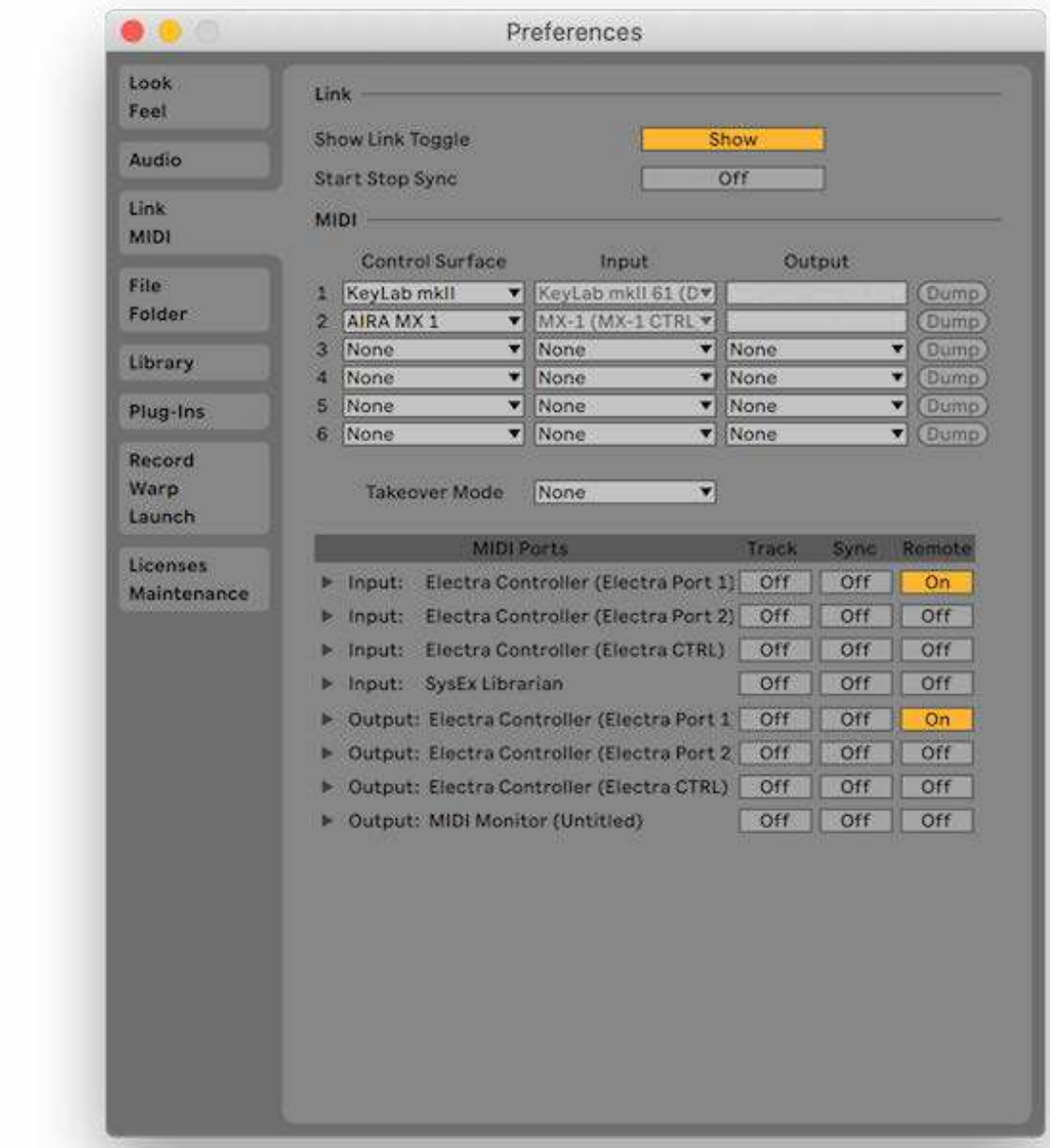
MIDI message	Parameter number	Description
CC	10	Level of track #1
CC	11	Filter Cutoff of Arturia Mini V3
CC	20	Level of track #2
CC	21	Filter Cutoff of Roland June-106

2. Configure MIDI in Ableton

Launch Ableton Live 10. When running configure MIDI settings so that Electra One is recognized by the Ableton as a control surface.

Go to `Preferences`, switch to `Link MIDI` tab, enable Remote option for:

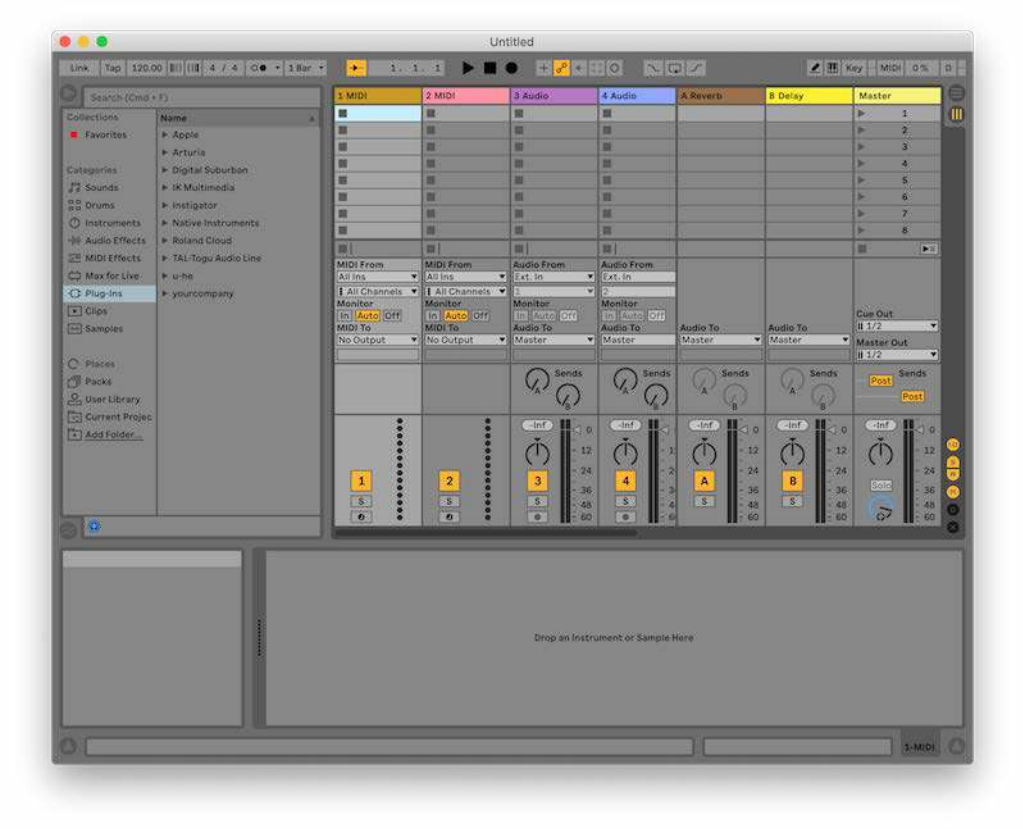
- Input: Electra Controller (Electra Port 1)
- Output: Electra Controller (Electra Port 1)



3. Assign plugins to the tracks

Open the browser sidebar by selecting the `Show browser` in the `View` menu. Click on `Plugins`

collection in the sidebar:



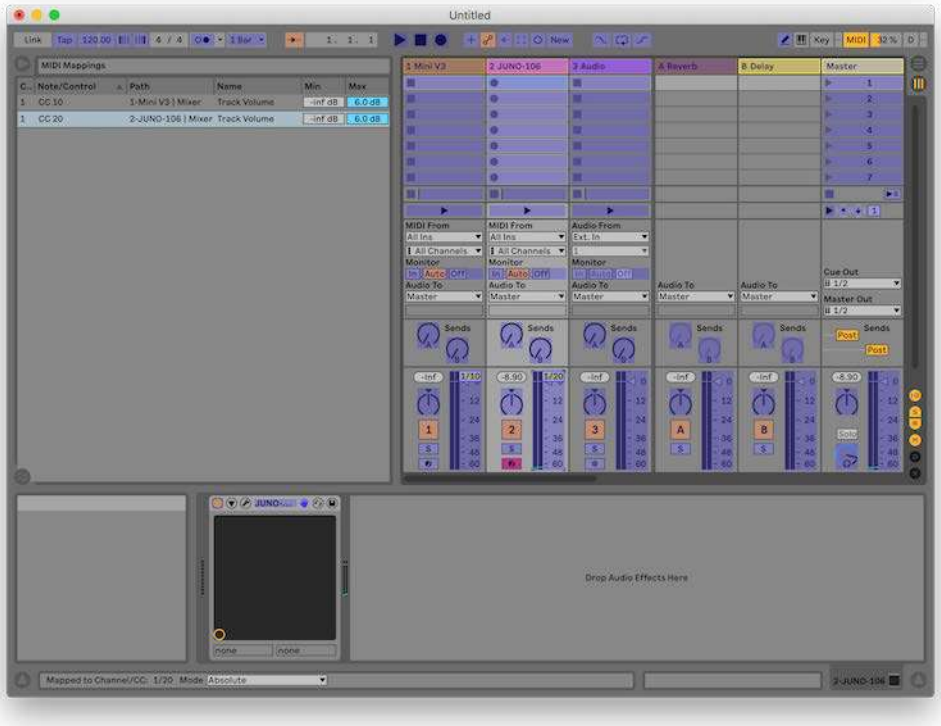
Pick Arturia Mini V3 plugin and assign it to track MIDI 1, then pick Roland Cloud JUNO-106 and assign it to track MIDI 2. Of course, you might not have these plugins, use any other plugins.

4. Map track levels to Electra Controls

Switch to MIDI mapping mode (`CMD + M` / `CTRL + M`) or click the MIDI toggle in the top-right corner of the main window. All of assignable Ableton's controls will become highlighted.

Click on the Track Volume of Mini V3 track and twist the TRACK LEVEL knob in the MIDI V3 group on the controller. The Track Volume is now linked to TRACK LEVEL knob.

Now click on the Track Volume of JUNO-106 track and twist the TRACK LEVEL in the JUNO-106 group on the controller.



If you have done everything correctly, you will see in the MIDI mappings sidebar that:

- CC 10 is assigned to Mini V3 Track value
- CC 20 is assigned to JUNO-106 Track volume

Now disable the MIDI mapping mode (eg. press `Esc`).

5. Verify that MIDI mapping works

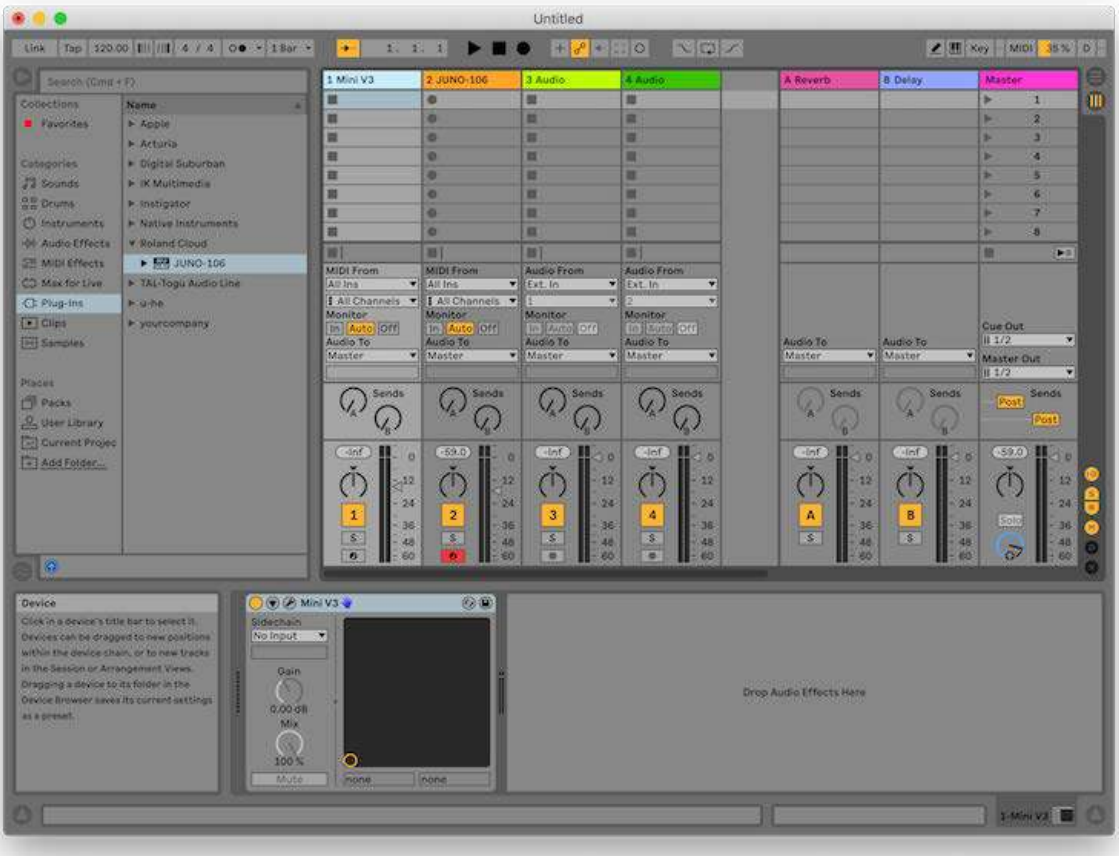
Twist both TRACK LEVEL Controls. You should see that the Track Volume of Mini V3 track and the Track Volume of JUNO-106 tracks are moving.

Now, change the value of the Track Volume using the mouse on the computer. The TRACK LEVEL Controls on the Electra One controller should reflect the changes. If not, review and repeat the above steps.

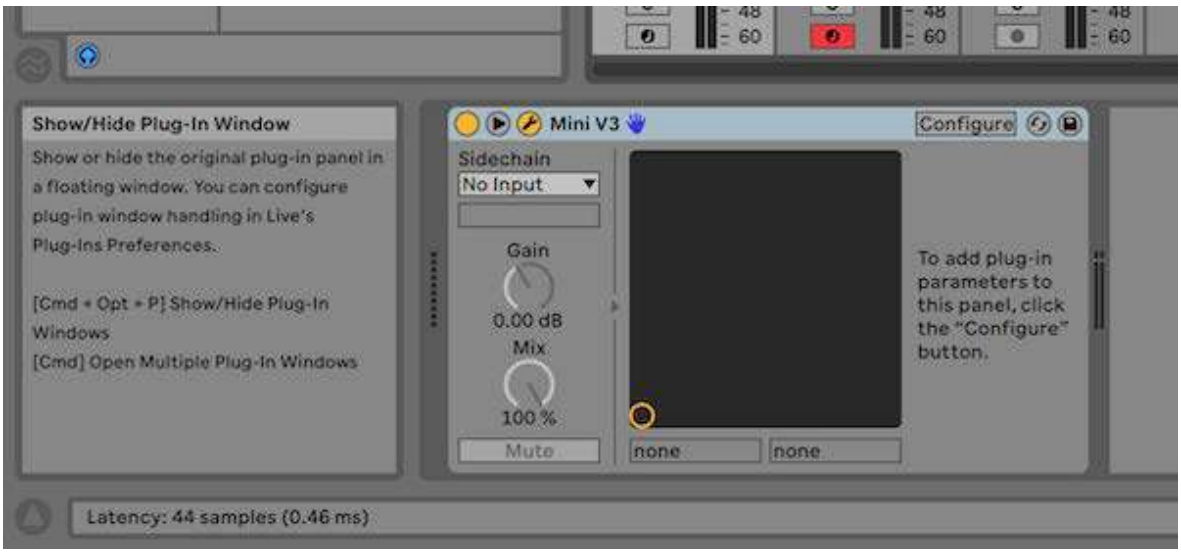
6. Extract plugin parameters

Select Mini V3 track by clicking on it. the Detail view underneath the Tracks will display

Details about Mini V3 plugin.

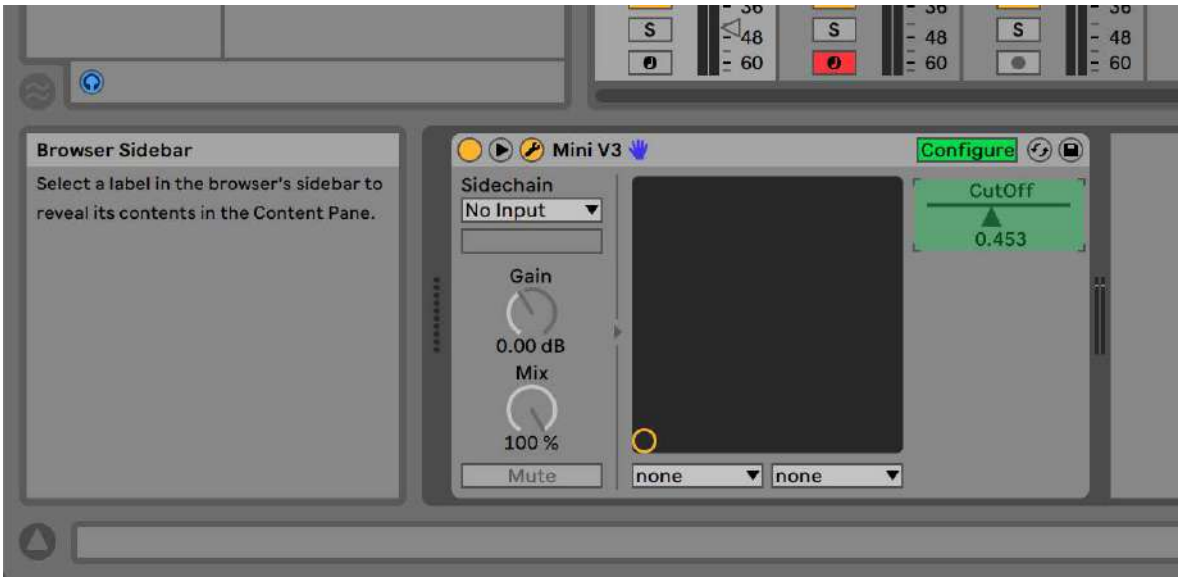


Click both `Unfold Device Parameters` and `Show/Hide Plugin Window`. The Device detail will unfold and will show the `Configure` button.



The plugin window will show up as well.

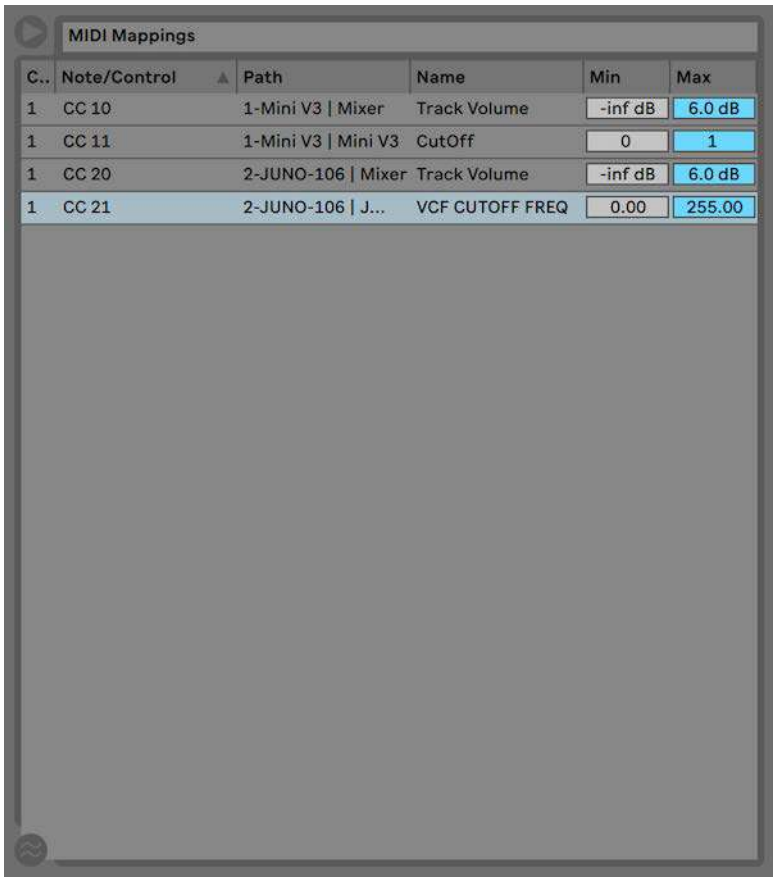
Now, click the `Configure` button, it will become lit green. Then click on the `Cutoff frequency` knob in the plugin window. The Device detail should now show that `Cutoff` has been added to the device.



7. Map the extracted parameter to Electra Control

The mapping of an extracted parameter is done in the same way as we mapped the Track Volume earlier in this tutorial. Enable MIDI mapping (`CMD + M` / `CTRL + M`). Click on the extracted `CUTOFF` parameter and twist the CUTOFF knob in the MINI V3 group on the controller. Press `Esc` to leave the MIDI mapping mode.

Repeat the steps for mapping the cutoff frequency of Roland JUNO-106 plugin. If done correctly, your MIDI mapping should read:



8. Well done

If you did everything listed above:

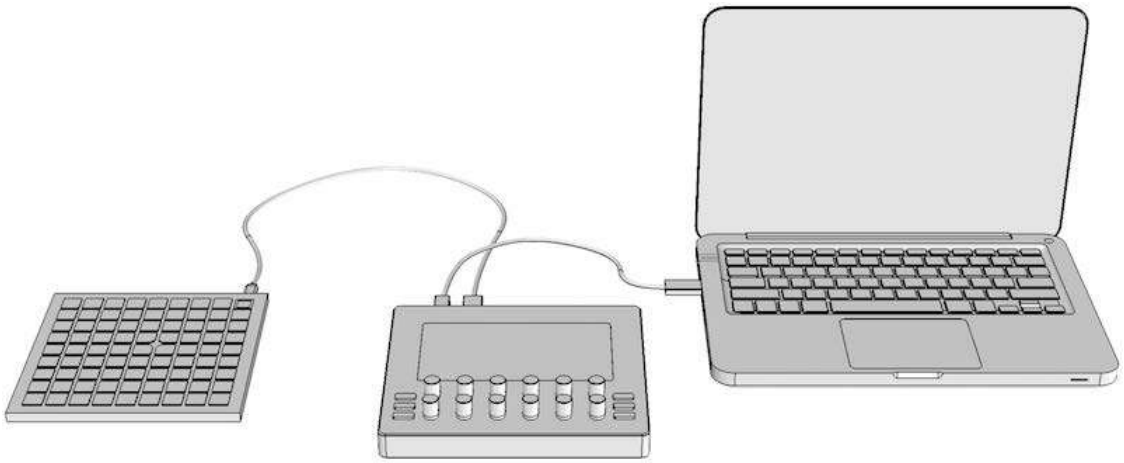
- you will be able to change values of cutoff frequencies of both synths from Electra One MIDI Controller
- Electra One MIDI Controller will reflect all changes of cutoff frequencies done on the computer, including switching of the sounds/patches in the plugin.

External control with LaunchPad

Goal of the tutorial

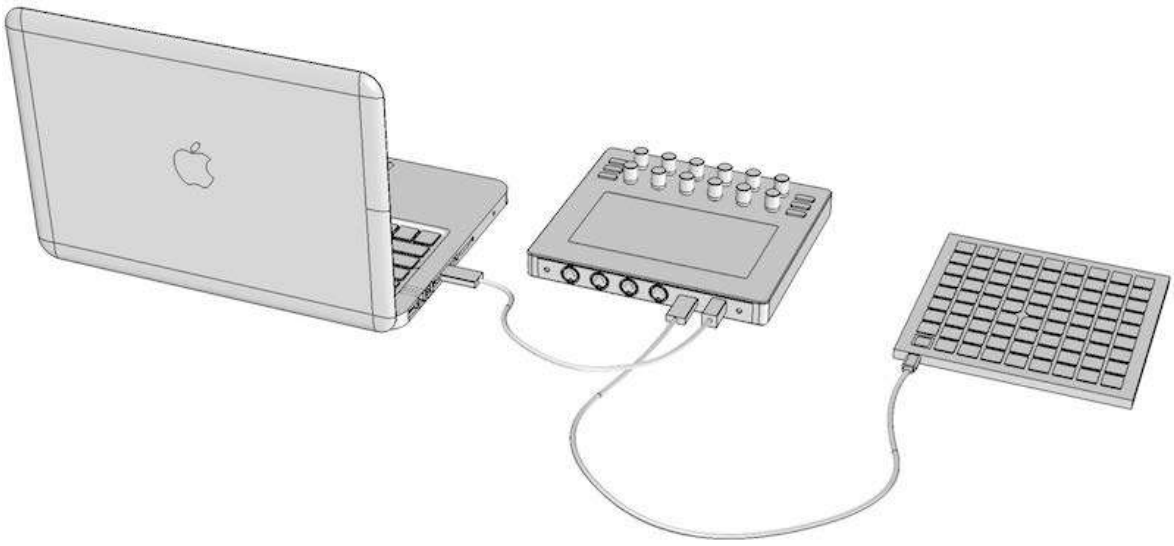
In this article you will learn how to use Novation LauchPad Mini to switch between the first six presets stored in Electra One.

Novation LaunchPad is a great companion of Electra One MIDI controller. Amongst other things it can speed up switching between presets.



1. Connect LaunchPad

Connect LaunchPad with USB cable to Electra One MIDI controller's `<USB HOST>` port on the rear panel.

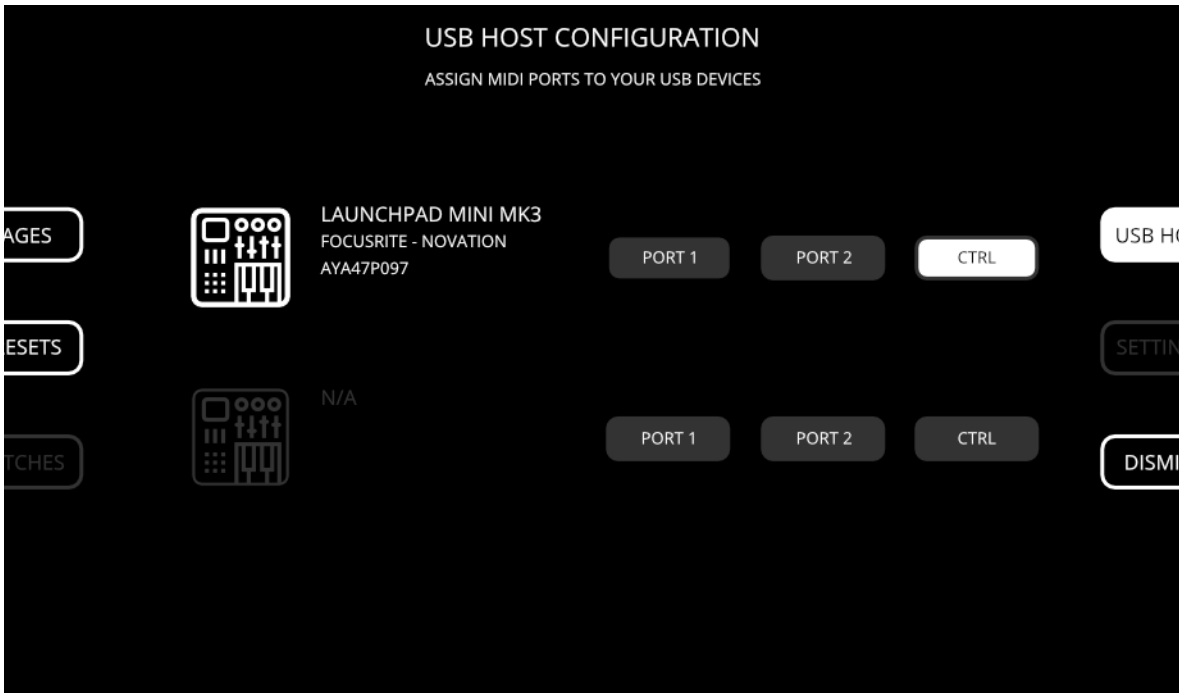


2. Assign LaunchPad to CTRL port

In order to use LaunchPad to send commands to Electra, it must be assigned to `<USB HOST>` CTRL port:

- Press the `[ MENU ]` button to open the Menu selection window.
- Select USB Host configuration, by touching the `[ USB HOST ]` on-screen button.
- Finally assign the CTRL port to LaunchPad device, by touching the `[ CTRL ]` on-screen button.

Your screen should read now:



LaunchPad now sends MIDI messages to Electra's CTRL port. Close the USB Host configuration by hitting the `[ DISMISS ]`.

3. Configure MIDI messages

LaunchPad now sends MIDI messages to Electra, but the messages do not match Electra's default assignments, see External MIDI control for details.

Let's reconfigure Electra so that it switch presets with LaunchPad's Note On MIDI messages.

- Make sure LaunchPad is switched to "Keys" mode.
- Prepare Configuration file with customized assignments.
- Upload the Configuration file to Electra.

The customized configuration file:

```
{
   "version":1,
   "midiControl":[
      {
         "event":"switchPreset",
         "eventParameter":1,
         "midiMessage":"note",
         "parameterNumber":36
      },
      {
         "event":"switchPreset",
         "eventParameter":2,
         "midiMessage":"note",
         "parameterNumber":38
      },
      {
         "event":"switchPreset",
         "eventParameter":3,
         "midiMessage":"note",
         "parameterNumber":40
      },
      {
         "event":"switchPreset",
         "eventParameter":4,
         "midiMessage":"note",
         "parameterNumber":41
      },
      {
         "event":"switchPreset",
         "eventParameter":5,
         "midiMessage":"note",
         "parameterNumber":43
      },
      {
        "event":"switchPreset",
        "eventParameter":6,
        "midiMessage":"note",
        "parameterNumber":45
      }
   ]
}
```

Now if you press any of C, D, E, F, G, A pads of the first row on the LaunchPad you will switch between the presets.

4. Auto-assign LauchPad to CTRL port

Now you have MIDI messages configured. This configuration is permanent, it means that it will stay in Electra after you power it off. The LaunchPad assignment to CTRL port, however,

will be forgotten on powering Electra off.

If you wanted to have this assignment permanent, adjust the Configuration as follows:

```
{
   "version":1,
   "usbHostAssigments":[
      {
         "pattern":"launchpad",
         "port":3
      }
   ],
   "midiControl":[
      {
         "event":"switchPreset",
         "eventParameter":1,
         "midiMessage":"note",
         "parameterNumber":36
      },
      {
         "event":"switchPreset",
         "eventParameter":2,
         "midiMessage":"note",
         "parameterNumber":38
      },
      {
         "event":"switchPreset",
         "eventParameter":3,
         "midiMessage":"note",
         "parameterNumber":40
      },
      {
         "event":"switchPreset",
         "eventParameter":4,
         "midiMessage":"note",
         "parameterNumber":41
      },
      {
         "event":"switchPreset",
         "eventParameter":5,
         "midiMessage":"note",
         "parameterNumber":43
      },
      {
        "event":"switchPreset",
        "eventParameter":6,
        "midiMessage":"note",
        "parameterNumber":45
      }
   ]
}
```

With this new configuration, LaunchPad will be automatically assigned to `<USB HOST>` port CTRL.

Writing SysEx templates

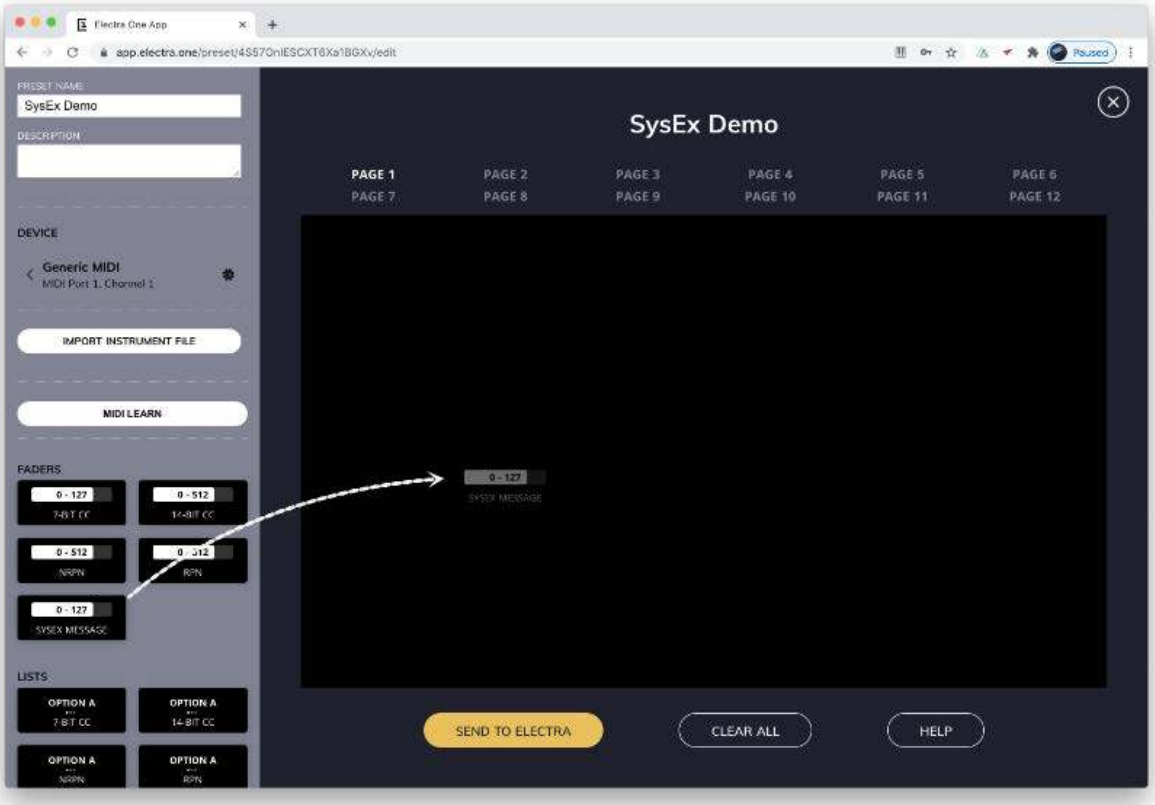
Goal of the tutorial

In this article, you will learn how to compose SysEx templates. SysEx templates will allow you to control parameters of your synths that are accessible only via SysEx messaging.

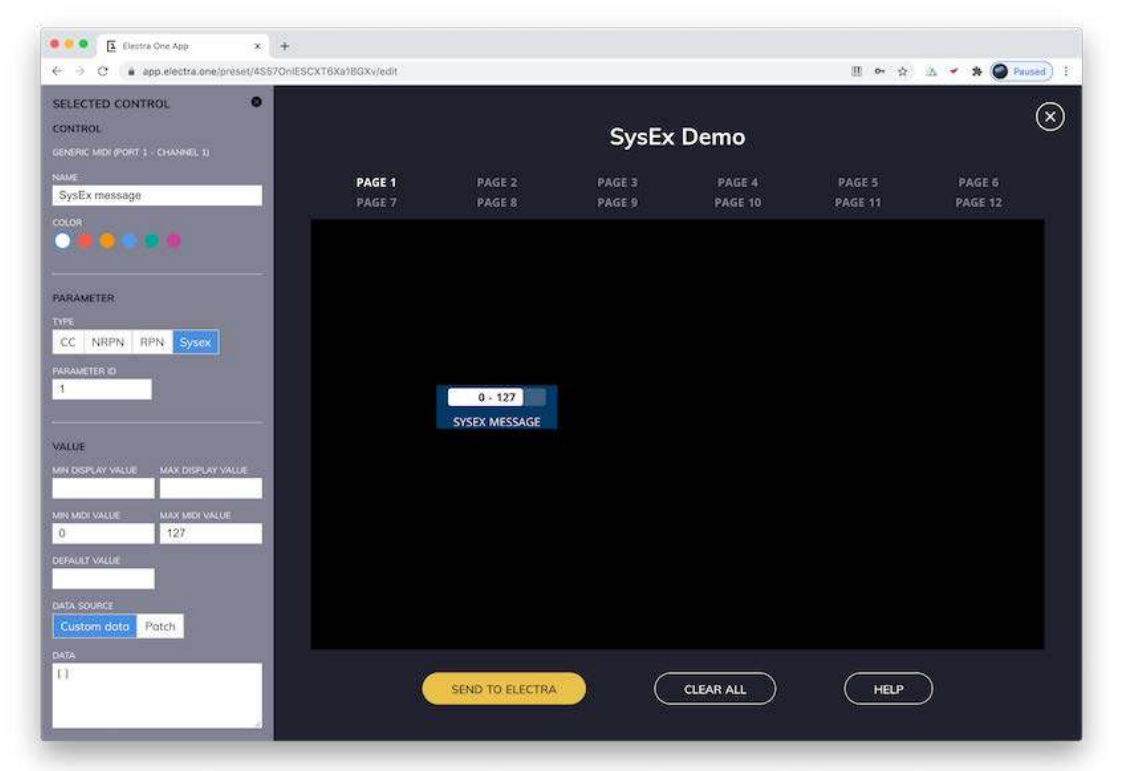
Electra One supports a number of MIDI message types. Some of them are very easy to use, for example, CC messages. Others are more complex and require a bit of technical knowledge of MIDI protocol. This article describes how to write **SysEx templates** - a way to compose SysEx messages sent to your gear when you twist the knobs.

SysEx message type

To control a SysEx parameter of your synth, you will need to pick a SysEx message Control from the palette and drag it on the canvas.



When the SysEx message Control is on the canvas, you can edit its properties in the sidebar.



There are two attributes specific to the SysEx message Control type:

- Data source
- Data

Data source

Data source tells Electra One what is the source of the SysEx data. it can be either **Custom data** or **Patch**. The **Custom data** allows you to enter a sequence of SysEx bytes and various variables for each Control individually. The **Patch**, on the other hand, will use a one patch definition of the device as the template for all your SysEx messages sent to one device.

We will focus on the **Custom data** only in this tutorial. The topic of using Patch is too complex and will be described in a separate tutorial.

Data

The **Data** text field is available only when Data source is set to **Custom data**. The field accepts JSON encoded data that describes the SysEx message. At present time entering the JSON formatted data is the only way of creating a SysEx template.

Creating an example SysEx messages

Constant bytes

This is quite a rare example, but let's start with that. It is the most simple SysEx message

possible. Imagine a situation when you need to send a constant message of the following SysEx bytes:

`F0h 3Eh 0Eh 00h 20h 00h 00h 3Eh 40h 7F F7h`

This is a SysEx message for Waldorf Microwave XT. It will set filter cutoff to 50% of its range. In order to create such a message, the following JSON needs to be inserted to the **Data** text field:

```
[ "3E", "0E", "00", "20", "00", "00", "3E", "40", "7F" ]
```

The leading F0 and F7 bytes are omitted. They are SysEx start and stop bytes, Electra inserts those to the message automatically. The rest is just an array of the bytes to be sent. The bytes may be entered in the hexadecimal format - quotes are required for hexadecimal notation. Another option is to use decimal notation. Above SysEx message can be expressed also as:

```
[ 62, 14, 0, 32, 0, 0, 62, 64, 127 ]
```

Now, when you turn the knob assigned to this Control, the above SysEx message will be sent to corresponding MIDI ports. Good as the first example, but not very useful in the real world. The message will be the same disregarding the value of the Control.

Simple value

The above example sets the filter cutoff frequency of Waldorf Microwave XT to 50%. The 50% is represented by the eighth byte "40". This is the byte holding the value of the cutoff frequency. "40" is a hexadecimal equivalent of number 64. If you will be changing this number, the cutoff frequency will change accordingly.

```
[ "3E", "0E", "00", "20", "00", "00", "3E", "00", "7F" ]
```

will close the filter, while

```
[ "3E", "0E", "00", "20", "00", "00", "3E", "7F", "7F" ]
```

will open the filter and will let all harmonics to pass. "00" till "7F" express a full range between 0 and 127 in decimal numbers.

A next logical step is to replace this eight byte with a variable that will be changing as you twist the knobs. The **Value** placeholder is what you use for that:

```
[ "3E", "0E", "00", "20", "00", "00", "3E", {"type":"value"}, "7F" ]
```

Above SysEx template will send the SysEx bytes whenever the value of the Control changes. It will, however, replace the `{"type":"value"}` placeholder with the current value of the Control.

Particular bits

The simple value placeholder is sufficient in the majority of situations. Unfortunately, there are moments when you need to have more control over inserting the value to the SysEx message. Electra One offers a few handy tools to handle such obstacles.

The value placeholder may have a few optional attributes. These can be used to specify:

- what parameter the value comes from

- what bits of the parameter value should be used
- where the value should be placed within the SysEx byte
- how many bits it will use.

This is an example of such a message (Yamaha DX7):

`F0h 43h 10h 00h 60h 2_bits_of_parameter_96 F7`

In Electra JSON you will express the same in the following way:

```
[
  "43",
  "10",
  "00",
  "60",
  {
    "type": "value",
    "rules": [
      {
        "parameterNumber": 96,
        "byteBitPosition": 0,
        "bitWidth": 2
      }
    ]
  }
]
```

The nested rules array provides additional info on how to compose the SysEx bytes. In this particular example, it says: place 2 bits (LSB) of the value of parameter 96 to the bit position 0 (LSB) in the SysEx byte.

attribute function parameterNumber id of the parameter as specified in the sidebar (always decimal) byteBitPosition bit position in the SysEx byte, ie. where the value will be placed bitWidth number of bits to be used

Two nibbles

As MIDI effectively uses 7-bits only, synthesizer manufacturers were quite creative about how to get values with higher resolution to their SysEx messages. That is where you meet nibbles, MSB bits, 7-bit encoded high-resolution numbers, and many more. Let's take a look at how Electra can divide one value into two separate SysEx bytes. Imagine following SysEx message (Roland TB3 without checksum):

`F0h 41h 10h 00h 00h 7Bh 12h 00h 20h 30h 0Ah value_upper_nibble value_lower_nibble F7h`

In this message, we have to place the upper 4 bits of the value to the **value_upper_nibble** SysEx byte and the lower 4 bits of the value to the _ value_lower_nibble_ SysEx byte. This is the way to do it, using the Electra's JSON:

```
[
  "41",
  "10",
  "00",
  "00",
  "7B",
  "12",
  "00",
  "20",
```

```
    "30",
    "0A",
    {
      "type": "value",
      "rules": [
        {
          "parameterNumber": 39,
          "parameterBitPosition": 4,
          "byteBitPosition": 0,
          "bitWidth": 4
        }
      ]
    },
    {
      "type": "value",
      "rules": [
        {
          "parameterNumber": 39,
          "parameterBitPosition": 0,
          "byteBitPosition": 0,
          "bitWidth": 4
        }
      ]
    }
]
```

For the first byte,s the message says: take 4 bits of the parameter 39 starting at bit position 4 and place them to this SysEx byte at position 0. For the second SysEx byte, it say: take 4 bits of parameter 39 but this time starting at position 0 and place them to the SysEx byte at position 0.

Now, when you twist the knob, Electra will split the value according to rules. You have just learned here about a new Rules attribute:

attribute	function
parameterBitPosition	bit position within the parameter value

Complex composed value

In early MIDI days, the memory of the machines was very limited. Manufacturers often used one SysEx byte to hold several parameter values. A very good example of this is the Operator switches byte of Yamaha DX7. Within this SysEx byte, 6 individual bits express if each of the six voice operators is switched on or off. From what you read above, you know you can address each of those bits by using **byteBitPosition** and **bitWidth** rule attributes. The tricky part here is, however, the fact that you always need to provide all six bits. Even if you work just with one operator.

***7 OPERATOR ON OFF**

BIT	b_5	b_4	b_3	b_2	b_1	b_0
OP	OP1	OP2	OP3	OP4	OP5	OP6

Bit Map
"0"...OFF "1"...ON

If you wondered why rules are a JSON array, you might know where we are heading to. This is how you would resolve this pickle in Electra JSON:

```
[
  "43",
  "10",
  "01",
  "1B",
  {
    "type": "value",
    "rules": [
      {
        "parameterNumber": 155,
        "byteBitPosition": 5,
        "bitWidth": 1
      },
      {
        "parameterNumber": 156,
        "byteBitPosition": 4,
        "bitWidth": 1
      },
      {
        "parameterNumber": 157,
        "byteBitPosition": 3,
        "bitWidth": 1
      },
      {
        "parameterNumber": 158,
        "byteBitPosition": 2,
        "bitWidth": 1
      },
      {
        "parameterNumber": 159,
        "byteBitPosition": 1,
        "bitWidth": 1
      },
      {
        "parameterNumber": 160,
        "byteBitPosition": 0,
        "bitWidth": 1
      }
    ]
  }
]
```

Please note, that in this JSON you do not have 6 different value placeholders. Instead, you have one value placeholder with six rules in it. These rules do so-called SysEx byte composition - they take more parameter values to compose just one SysEx byte.

In order to do this, you create 6 Controls in the Electra Editor and assign them any unique parameter id. In this example, we will use parameter ids 155, 156, 157, 158, 159, and 160. Make each of the Controls a list with O and 1 value. All of these controls have the same Data JSON defined. When you turn the knob, for instance, of Operator 3, Electra will not only use the value of parameter 157, it will compose the SysEx byte using the values of all six parameters.

Checksums

Checksums are special SysEx bytes that are meant to provide information on whether or not the data in the message is correct. They are mathematical formulae applied to part of the bytes of the SysEx message. The result of this calculation is placed at the end of the SysEx message. This allows the receiving MIDI device to know that the data was not malformed during the transmission. SysEx messages with wrong checksum byte are usually ignored by the receiving device.

By definition, the checksum will be a constant number for messages that are made of constant bytes only. As soon as you include a value placeholder to the JSON, the constant checksum will stop working. The checksum placeholder comes in play in such a situation.

Above we have shown an example SysEx message for Roland TB3. This message is not actually complete and would not work. Roland often uses checksums in their messages. We simplified that example on purpose. Now, let's take a look at how the correct message would look like:

```
[
  "41",
  "10",
  "00",
  "00",
  "7B",
  "12",
  "00",
  "20",
  "30",
  "0A",
  {
    "type":"value",
    "rules":[
      {
        "parameterNumber":39,
        "parameterBitPosition":4,
        "byteBitPosition":0,
        "bitWidth":4
      }
    ]
  },
  {
    "type": "value",
    "rules": [
      {
        "parameterNumber": 39,
        "parameterBitPosition": 0,
        "byteBitPosition": 0,
        "bitWidth": 4
      }
    ]
  },
  {
    "type": "checksum",
    "algorithm": "roland",
    "start": 6,
    "length": 6
  }
]
```

The last SysEx byte is a **Checksum** placeholder. It has three attributes that define what data should be part of checksum calculation and what type of the checksum calculation should be used.

The **Checksum** placeholder will be replaced with the result of the calculation in the outgoing SysEx message.

attribute	function
algorithm	type of checksum calculation formulae. Often based on manufacturer
start	position of the first byte to be included in calculation
length	number of bytes to be included in the calculation

attribute function algorithm type of checksum calculation formulae. Often based on manufacturer start position of the first byte to be included in calculation length number of bytes to be included in the calculation

Currently, we support the following algorithms:

```
roland
waldorf
fractal
```

If you wish to have a new one, please create a github issue or contact us.

Final words

We have designed the Data JSON so that it can be easily extended with new types of placeholders or new types of attributes can be added to existing ones. Please feel free to talk to us if you have suggestions or ideas.

If you had trouble with getting your SysEx messages working, contact us using the integrated Chat system in the Electra Editor. Always mention what synth you are working on. We will try to help you to get it working. Either immediately on the chat or later on when we have time to work on it.

Last but not least, JSON is used because it is handy to describe data objects. This is easy to understand for tech-savvy people. For those who do not have the technical skills or who do not want to spend time on it, we will provide soon a visual editor.

Developers

ELECTRA ONE

SysEx implementation

Electra One MIDI controller can be configured, programmed, and controlled with two different communication protocols:

- SysEx MIDI messages
- low-level USB Raw HID

This document describes the essential data exchange and control commands using the SysEx message format over USB MIDI.

The management port

Electra One can be controlled by exchanging SysEx messages over `Electra Controller CTRL` port. Messages sent to other Electra's ports will be ignored.

Manufacturer SysEx Id

Electra One uses the MIDI association Manufacturer SysEx Id of Electra One s.r.o.

```
0x00 0x21 0x45
```

Get Electra info

Electra One MIDI controller can provide info about the hardware and currently loaded firmware on a request. This call comes in handy if you need to find out if connected Electra is working correctly and get information about the firmware it runs.

For example, the Electra App account and Electra Editor use this call to verify that Electra One controller is connected correctly and display the connection indicator.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x7F 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x02 Query data
- 0x7F Electra information
- 0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x7F info-json-data 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x01 Data dump
- 0x7F Electra information
- **info-json-data** JSON document with info about Electra (see below)
- 0xF7 SysEx closing byte

An example of info-json-data

```
{
   "versionText":"v0.9.11",
   "versionSeq":91100,
   "serial":"E01-123456"
}
```

Upload a preset

The preset upload call is meant to upload a new preset to the Electra One MIDI controller. The preset is always loaded to a currently selected preset slot (out of 12 preset slots supported). Once the preset is uploaded, it is activated immediately and the user may use it.

Request

```
0xF0 0x00 0x21 0x45 0x01 0x00 preset-json-data 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x01 Upload data
- 0x00 Preset file
- **preset-json-data** bytes representing ascii bytes of the preset file
- 0xF7 SysEx closing byte

Detailed information about **preset-json-data** is provided at Preset format description

Response

- No response

Download a preset

Request

```
0xF0 0x00 0x21 0x45 0x02 0x00 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x02 Query data
- 0x00 Preset file
- 0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x00 preset-json-data 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x01 Data dump / Upload data
- 0x00 Preset file
- **preset-json-data** JSON document with info about Electra (see below)
- 0xF7 SysEx closing byte

An example of preset-json-data

```
{
  "version":2,
  "name":"ADSR Test",
  "projectId":"d8WjdwYrP3lRyyx8nEMF",
  "pages":[
  ],
  "devices":[
  ],
  "overlays":[
  ],
  "groups":[
  ],
  "controls":[
  ]
}
```

Electra One MIDI controller responds with the SysEx message that has exactly the same format as the Preset upload message (see above). Thus a SysEx message downloaded with a preset download call can be used to upload the preset to Electra later on.

This call always downloads a preset that is currently selected and active in the controller.

Detailed information about **preset-json-data** is provided at Preset format description

Upload a configuration

The configuration upload call is meant to upload and apply a new Electra One configuration to the controller.

Request

```
0xF0 0x00 0x21 0x45 0x01 0x02 configuration-json-data 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x01 Upload data
- 0x02 Configuration file
- **configuration-json-data** bytes representing ascii bytes of the configuration file
- 0xF7 SysEx closing byte

Detailed information about **configuration-json-data** is provided at Configuration format description

Response

- No response

Download a configuration

Request

```
0xF0 0x00 0x21 0x45 0x02 0x02 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x02 Query data
- 0x02 Configuration file
- 0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x02 configuration-json-data 0xF7
```

- 0xF0 SysEx header byte
- 0x00 0x21 0x45 Electra One MIDI manufacturer Id
- 0x01 Data dump / Upload data

- 0x02 Configuration file
- **configuration-json-data** JSON document with info about Electra (see below)
- 0xF7 SysEx closing byte

An example of configuration-json-data


```
{
   "version":1,
   "router":{
      "usbDevToMidiIo":true,
      "usbDevToUsbHost":true,
      "midiIoToUsbDev":true,
      "midiIoToUsbHost":true,
      "usbHostToMidiIo":true,
      "usbHostToUsbDev":true,
      "electraToMidiIo":true,
      "electraToUsbHost":true,
      "electraToUsbDev":true
   },
   "presetBanks":[
     {
        "id":1,
        "name":"FX UNITS",
        "color":"FFFFFF"
     },
     {
        "id":2,
        "name":"SAMPLERS",
        "color":"529DEC"
     }
   ],
   "usbHostAssigments":[
     {
        "pattern":"launchpad",
        "port":3
     }
   ],
   "midiControl":[
     {
        "event":"switchPreset",
        "eventParameter":1,
        "midiMessage":"program",
        "parameterNumber":1
     }
   ]
}
```

ELECTRA ONE

Preset format description

This document describes the format of the Electra One preset file. The preset file holds a complete definition of Electra preset, no other data needs to be transferred to Electra to run a preset.

> WARNING
>
> Do not load presets in this format to Electra with firmware 0.9.x. This preset format is supported with version 1.0.0 and above.

Why a new version?

Electra One has been used by real users for about over 9 months now. We have listened to the feedback from our users as well as we have come up with a number of new features we wanted to implement. It has become obvious that the original format of the Electra preset file was rather limiting.

The new format is designed to address the following feature requests:

- support pads (touch-operated on-screen controls)
- support controls that represent more than one value (envelopes, etc)
- support macro functionality (one knob changing of multiple controls at the same time)
- support lists with images (eg. waveshapes) instead of or next to text labels
- make it possible to assign "functions" to rubber buttons
- place controls freely on the canvas (not strictly aligned in the 6x6 grid)
- allow setting a custom size to individual controls

Preset JSON format

JSON schema

This is a WIP (work-in-progress) version of JSON schema of Electra preset file format version 2

Minification

We strongly advise minifying the JSON data prior to transfer to Electra One over the USB MIDI protocol. Minification greatly affects the amount of data that must be transferred, therefore it speeds up the time of uploading and downloading presets.

Top level objects

The preset has a number of top-level objects. These are either simple elements providing info about the preset itself or complex objects that define the structure and data of the preset.

```
{
  "version": 2,
  "name": "ADSR Test",
  "projectId": "d8WjdwYrP3lRyyx8nEMF",
  "pages": [
  ],
  "devices": [
  ],
  "overlays": [
  ],
  "groups": [
  ],
  "controls": [
  ]
}
```

version

Provides information about the version of the preset file. Electra controller uses version information to distinguish between various preset file formats.

- mandatory
- numeric

name

A name of the preset. The name will be shown to the user on the screen in the status bar.

- mandatory
- string
- minLength = 0
- maxLength = 20

projectId

An external identifier. This id can be used to refer any external data within your Electra applications. For example, the projectId is used to associate a preset within the meta data, such as the preset description, author, etc., in the Electra Preset Editor application.

- optional
- string
- minLength = 0
- maxLength = 20

pages

An array of pages. A page is a collection of controls and groups displayed to the user at once.

- mandatory
- array

example:

```
"pages": [
   {
      "id": 1,
      "name": "OSCILLATORS"
   },
   {
      "id": 2,
      "name": "FILTER"
   }
]
```

devices

An array of devices. A device is a hardware MIDI device or software device (application, VST/AU plugin) connected to the Electra One.

- mandatory
- array

example:

```
"devices": [
   {
      "id": 1,
      "name": "My MKS-50",
      "instrumentId": "roland-mks50",
      "port": 1,
      "channel": 1
   },
   {
      "id": 2,
      "name": "BeatFX plugin",
      "instrumentId": "generic-MIDI",
      "port": 2,
      "channel": 1
   }
]
```

overlays

An array of overlays. An overlay is a list of text labels or graphical symbols that can be assigned to a control.

- optional
- array

examples:

```
"overlays": [
   {
      "id": 1,
      "items": [
         {
            "value": 0,
            "label": "SAW"
         },
         {
            "value": 1,
```

```
            "label": "SQUARE"
         },
         {
            "value": 2,
            "label": "WHITE NOISE"
         },
         {
            "value": 4,
            "label": "PINK NOISE"
         }
      ]
   },
   {
      "id": 2,
      "items": [
         {
            "value": 0,
            "label": "Morph (P6)"
         },
         {
            "value": 16,
            "label": "Sinus"
         },
         {
            "value": 32,
            "label": "Triangle"
         }
      ]
   },
   {
      "id": 3,
      "items": [
         {
            "value": 0,
            "label": "Square",
            "bitmap": "AAAAAAAA/P//AQAA/P//AQAwDACAAQAwDACAAQAwDACAA
QAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAw
DACA//8/DACA//8/AAAAAAAAAAAAAAAA"
         },
         {
            "value": 1,
            "label": "Triangle",
            "bitmap": "AAAAAAAAAADgAAAAADwAQAAAAAYAwAAAAAMBgAAAAAGD
AAAAAADGAAAAIABMAAgAMAAYAAwAGAAwAAYBDAAgAEMDBgAAAMGGAwAAAYDMAYAAIwB4
AMAAPgAwAEAAHAAAAAAAAAAAAAAAAAA"
         },
         {
            "value": 2,
            "label": "Saw",
            "bitmap": "AAAAAAAAAAAAwAEAAAAA8AEAAAAAnAEAAAAAhwEAAADAg
QEAAABwgAEAAAAcgAEAAAAHgAEAAMABgAE4AHAAgAE0ABwAgIEDAAcAgOEAwAEAgDkAc
AAAgA8AHAAAgAMAAAAAAAAAAAAAAAAA"
         }
      ]
   }
]
```


groups

An array of groups. A group is a graphical separator to improve a layout of presets.

- optional
- array

example:

```
"groups": [
   {
      "pageId": 1,
      "name": "ATTRIBUTES",
      "bounds": [
         170,
         16,
         485,
         16
      ],
      "color": "FFFFFF"
   }
]
```

controls

An array of controls. A control is a representation of one or more MIDI parameters or messages that can be controlled by the user.

- mandatory
- array

examples:

A simple control with one value assigned.

```
"controls": [
   {
      "id": 1,
      "type": "fader",
      "name": "WHITE",
      "color": "FFFFFF",
      "bounds": [
         0,
         40,
         146,
         56
      ],
      "pageId": 1,
      "controlSetId": 1,
      "inputs": [
         {
            "potId": 1,
            "valueId": "value"
         }
      ],
      "values":[
         {
            "id": "value",
            "message": {
               "deviceId": 1,
               "type": "cc7",
               "parameterNumber": 1,
```

```
                "min": 0,
                "max": 127
            },
            "min": 0,
            "max": 127
        }
    ]
},
{
    "id": 2,
    "type": "fader",
    "name": "RED",
    "color": "F45C51",
    "bounds": [
        170,
        40,
        146,
        56
    ],
    "pageId": 1,
    "controlSetId": 1,
    "inputs": [
        {
            "potId": 2,
            "valueId": "value"
        }
    ],
    "values": [
        {
            "message": {
                "deviceId": 1,
                "type": "cc7",
                "parameterNumber": 2,
                "min": 0,
                "max": 127
            },
            "min": 0,
            "max": 127
        }
    ]
}
]
```

Page

A page is a collection of controls and graphic objects shown on the screen at once. Each page has a name to make it easier to switch between pages. The page name is shown in the status bar.

example:

```
{
    "id": 1,
    "name": "OSCILLATORS"
}
```

id

A unique identifier of the page. The identifier is used by other objects to refer to a particular page.

- mandatory
- numeric
- min = 1
- max = 12

name

A name of the page. The name makes it easier for users to get oriented in Electra presets.

- mandatory
- string
- minLength = 0
- maxLength = 20

Device

A device is a MIDI hardware or software instrument connected to one of Electra's ports. It can be a hardware synth connected to a MIDI IO port, a hardware sequencer connected to Electra's USB host port, or a software plugin attached to Electra's USB device ports. Electra can handle up to 16 simultaneously connected devices. When working with Electra, you always need to define your connected devices, you never send or receive MIDI messages from port and channel directly.

example:

```
{
   "id": 1,
   "name": "Generic MIDI",
   "instrumentId": "generic-MIDI",
   "port": 1,
   "channel": 1
}
```

id

A unique identifier of the device. The identifier is used in other objects to refer to a particular device.

- mandatory
- numeric
- min = 1
- max = 16

name

A user-defined name of the device. The name makes it easier for users to remember and identify devices.

- mandatory
- string
- minLength = 0

- maxLength = 20

port

A port number that represents the MIDI bus inside the Electra. Port 1 interconnects MIDI IO port 1, USB Host port 1, USB device port 1. Port 2 interconnects MIDI IO 2, USB Host port 2, and USB device port 2.

- mandatory
- numeric
- min = 1
- max = 2

channel

A MIDI channel where the device transmits the MIDI messages.

- mandatory
- numeric
- min = 1
- max = 16

Overlay

Overlays are lists of text labels linked to particular MIDI values. Typically they are assigned to selection list controls or faders. Overlays are referred to by its identifier. Overlay items can be both text labels or graphical symbols represented with bitmap images.

examples:

an overlay with text labels only

```
{
   "id":1,
   "items":[
      {
         "value": 0,
         "label": "SAW"
      },
      {
         "value": 1,
         "label": "SQUARE"
      },
      {
         "value": 2,
         "label": "WHITE NOISE"
      },
      {
         "value": 3,
         "label": "PINK NOISE"
      }
   ]
}
```

an overlay with the bitmap data


```
{
   "id": 3,
   "items": [
      {
         "value": 0,
         "label": "Square",
         "bitmap": "AAAAAAAA/P//AQAA/P//AQAwDACAAQAwDACAAQAwDACAAQAw
DACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACAAQAwDACA
//8/DACA//8/AAAAAAAAAAAAAAAA"
      },
      {
         "value": 1,
         "label": "Triangle",
         "bitmap": "AAAAAAAAAADgAAAAADwAQAAAAAYAwAAAAAMBgAAAAAGDAAA
AAADGAAAAIABMAAgAMAAYAAwAGAAwAAYBDAAgAEMDBgAAAMGGAwAAAYDMAYAAIwB4AMA
APgAwAEAAHAAAAAAAAAAAAAAAAAA"
      },
      {
         "value": 2,
         "label": "Saw",
         "bitmap": "AAAAAAAAAAAAwAEAAAAA8AEAAAAAnAEAAAAAhwEAAADAgQEA
AABwgAEAAAAcgAEAAAAHgAEAAMABgAE4AHAAgAEOABwAgIEDAAcAgOEAwAEAgDkAcAAA
gA8AHAAAgAMAAAAAAAAAAAAAAAAA"
      }
   ]
}
```


id

A unique identifier of the overlay. List and fader controls use the identifier to refer to a particular overlay.

- mandatory
- numeric
- min = 1
- max = 51

items

An array of value - label pairs.

- mandatory
- array

value

A MIDI value assigned to the label.

- mandatory
- numeric
- min = 0
- max = 16383

label

A text label assigned to the MIDI value. The control displays the text labels to the users.

- mandatory
- string
- minlength = 0
- maxLength = 20

bitmap

A base64 encoded bitmap image. The bitmap image is in XBM format with 48 x 18 size.

- mandatory
- base64 encoded XBM image

Group

Graphical separators used to organize controls to groups by meaning. For example a group "Envelope 1" can be created for controls "Attack", "Decay", "Sustain", and "Release". Groups do not provide any other functionality than the visual grouping of controls.

example:

```
{
   "pageId": 1,
   "name": "ENVELOPE",
   "bounds": [
      0,
      16,
      486,
      16
   ],
   "color": "FFFFFF"
}
```

pageId

A reference to a page identifier. Each control must belong to exactly one page and the page must be defined within the pages array.

- mandatory
- numeric
- min = 1
- max = 12

name

A name of the group. The name is shown to the user inside the group graphics. The name is trimmed to fit the size of the group.

- mandatory
- string
- minLength = 0
- maxLength = 40

bounds

A bounding box of the control, ie. the definition of the control's position on the screen and its size. The bounding box is represented as an array of [x, y, width, height]

- mandatory
- array with fixed items

color

A 24-bit RGB code of the control's color. The colors are limited to six predefined colors.

- optional
- string
- default = FFFFFF
- enum
- FFFFFF (white)
- F45C51 (red)
- F49500 (orange)
- 529DEC (blue)
- 03A598 (green)
- C44795 (pink)

Control

A Control is a represetation of a MIDI parameter or a MIDI message. Controls visualize and change values of MIDI parameters. A control is for example a fader, knob, pad, or ADSR envelope. A control consists of information about values that are mapped to particular MIDI messages.

examples:

A simple control with one value assigned.


```
{
   "id": 1,
   "type": "fader",
   "name": "WHITE",
   "color": "FFFFFF",
   "bounds":[
      0,
      40,
      146,
      56
   ],
   "pageId": 1,
   "controlSetId": 1,
   "inputs": [
      {
         "potId": 1,
         "valueId": "value"
      }
   ],
   "values":[
      {
         "id": "value",
         "message": {
            "deviceId": 1,
            "type": "cc7",
            "parameterNumber": 1,
```

```
            "min": 0,
            "max": 127
         },
         "min": 0,
         "max": 127
      }
   ]
}
```

an ADSR control with multiple values assigned

```
{
   "id": 1,
   "pageId": 1,
   "bounds": [
      10,
      40,
      158,
      73
   ],
   "controlSetId": 1,
   "inputs": [
      {
         "potId": 1,
         "valueId": "attack"
      },
      {
         "potId": 2,
         "valueId": "decay"
      }
   ],
   "type": "adsr",
   "name": "ADSR",
   "color": "F49500",
   "values":[
      {
         "id": "attack",
         "min": 0,
         "max": 127,
         "message": {
            "deviceId": 1,
            "type": "cc7",
            "parameterNumber": 1,
            "min": 0,
            "max": 127
         }
      },
      {
         "id": "decay",
         "min": 0,
         "max": 127,
         "message": {
            "deviceId": 1,
            "type": "cc7",
            "parameterNumber": 2,
            "min": 0,
            "max": 127
         }
```

```
        },
        {
            "id": "sustain",
            "min": 0,
            "max": 127,
            "message": {
                "deviceId": 1,
                "type": "cc7",
                "parameterNumber": 3,
                "min": 0,
                "max": 127
            }
        },
        {
            "id": "release",
            "min": 0,
            "max": 127,
            "message": {
                "deviceId": 1,
                "type": "cc7",
                "parameterNumber": 4,
                "min": 0,
                "max": 127
            }
        }
    ]
}
```

id

A unique identifier of the control. Electra uses the id to uniquely identify each control.

- mandatory
- numeric
- min = 1
- max = 432

type

A type of functional and visual representation of the control.

- mandatory
- enum
- fader
- list
- pad
- vfader
- dial
- adsr
- adr
- dx7envelope

name

A name of the control. The name is usually shown underneath the control. When the control

receives touch-event via the physical knob, the name is highlighted. If a name is an empty string or the attribute is omitted, the name is not shown and touch indication is disabled.

- optional
- string
- minLength = 0
- maxLength = 14

color

A 24-bit RGB code of the control's color. The colors are limited to six predefined colors.

- optional
- string
- default = FFFFFF
- enum
- FFFFFF (white)
- F45C51 (red)
- F49500 (orange)
- 529DEC (blue)
- 03A598 (green)
- C44795 (pink)

bounds

A bounding box of the control, ie. the definition of the control's position on the screen and its size. The bounding box is represented as an array of [x, y, width, height]

- mandatory
- array with fixed items

pageId

A reference to a page identifier. Each control must belong to exactly one page and the page must be defined within the pages array.

- mandatory
- numeric
- min = 1
- max = 12

controlSetId

Controls placed on one page can be further divided into control sets. The control sets are used t assign controls to pots (knobs). Users may switch between controls sets by pressing the hardware buttons or by sending MIDI messages to Electra. Only one control set can be active at any time. The controls of the active control sets are highlighted.

- optional
- numeric
- default = 0
- min = 0
- max 12

values

An array of values associated with the control. A values represent an instance of the value of certain MIDI parameter. Actions made with the control (turning assigned pot, touch events) effectively change associated values and trigger transmission of MIDI messages.

- mandatory
- array

examples:

```
"values": [
   {
      "id": "value",
      "min": -64,
      "max": 63,
      "defaultValue": 0,
      "message":{
         "deviceId": 1,
         "type": "cc7",
         "parameterNumber": 1,
         "min": 0,
         "max": 127
      }
   }
]
```

Input

An Input provides information about an assignment of a physical control/gesture to a value. An example is assigning a knob to a value of the control.

- optional
- object

examples:

```
{
   "potId": 1,
   "valueId": "attack"
}
```

potId

An identifier of the physical pot (knob). There are 12 pots on Electra, identified as 1 (top-left) to 12 (bottom-right) pot. A control with an assigned pot can be controlled by turning the physical knob. Providing a given control set is active.

- optional
- numeric
- default = 0
- min = 0
- max = 12

valueId

An identifier of the value within the "values" array.

- optional
- string
- default = value
- minLength = 1
- maxLength = 20

Value

A value represents a parameter value of given control. A value is mapped to a value of a MIDI parameter or a MIDI message. The value object allows translation of MIDI values to user-friendly display values.

examples:

a continuous value

```
{
   "id": "value",
   "min": -64,
   "max": 63,
   "defaultValue": 0,
   "message": {
      "deviceId": 1,
      "type": "cc7",
      "parameterNumber": 1,
      "min": 0,
      "max": 127
   }
}
```

value with a list of discrete values (an overlay)

```
{
   "id": "value",
   "overlayId": 2,
   "message":{
      "deviceId": 1,
      "type": "cc7",
      "parameterNumber": 2
   }
}
```

id

An identifier of the value. This identifier is a text string. This is to make it easier for programmers to get oriented. The identifier expresses the meaning of the value, eg. attack, rate, or value.

- optional
- string
- default = value
- minLength = 1
- maxLength = 20

min

A minimum value that the control can display. Note this is not the MIDI value, it is the minimum value that can be displayed by the control.

- optional
- numeric
- default = 0
- min = -16383
- max = 16383

max

A maximum value that the control can display. Note this is not the MIDI value, it is the maximum value that can be displayed by the control.

- optional
- numeric
- default = 0
- min = -16383
- max = 16383

defaultValue

A value to be set when the preset is loaded. The default value is also recalled when user double-taps the control on the touch screen.

- optional
- numeric
- default = 0
- min = -16383
- max = 16383

overlayId

A reference to the overlay identifier defined in the array of overlays. The list control will use the overlay items as the list items. Fader control will show overlay labels for matching values.

- optional
- numeric
- min = 1
- max = 128

message

An object that describes a MIDI message assigned to the value.

- mandatory
- object

Message

An object that describes a MIDI message that will be sent when the value of the control is changed by the touch or turning the knobs. The message is also used to parse incoming MIDI messages. When incoming MIDI data matches the message object, the value of the control is adjusted accordingly.

- optional
- object

examples:

A simple CC7 message

```
"message": {
   "deviceId": 1,
   "type": "cc7",
   "parameterNumber": 1,
   "min": 0,
   "max": 127
}
```

A message with a SysEx template

```
"message": {
   "deviceId": 1,
   "type": "SysEx",
   "parameterNumber": 6,
   "min": 0,
   "max": 127,
   "data": [
      67,
      16,
      1,
      15,
      {
         "type": "value",
         "rules": [
            {
               "parameterNumber": 40,
               "bitPosition": 0,
               "bitWidth": 3
            }
         ]
      }
   ]
}
```

A simple CC7 message with event midi values assigned

```
"message": {
   "deviceId": 1,
   "type": "cc7",
   "parameterNumber": 1,
   "offValue": 0,
   "onValue": 127
}
```

A simple NRPN message handling a negative values

```
"message": {
   "deviceId": 1,
   "type": "cc7",
   "parameterNumber": 1,
```

```
    "lsbFirst": false,
    "twosComplement": true
}
```

deviceId

A reference to the device identifier defined in the array of devices. The message will be sent to the referenced device. Also, messages received from this device that match the message definition will modify the value accordingly.

type

A type of the MIDI message. The type is not limited to basic MIDI messages but supports their higher level implementation, such as NRPN, etc.

- mandatory
- enum
- cc7
- cc14
- nrpn
- rpn
- SysEx
- note
- program
- start
- stop
- tune

parameterNumber

A parameter number of the message. The parameterNumber is used to specify the parameter number, note number, program number. To fully support NRPN and SysEx, a parameterNumber is a 14-bit number.

- optional
- numeric
- min = 0
- max = 16383

min

A MIDI minimum value to be transferred. This minimum MIDI value is mapped to the display value minimum defined in the value object.

- optional
- numeric
- default = 0
- min = 0
- max = 16383

max

A MIDI maximum value to be transferred. This maximum MIDI value is mapped to the display

value maximum defined in the value object.

- optional
- numeric
- default = 0
- min = 0
- max = 16383

data

An array of bytes and placeholder variables to be sent and parsed for SysEx messages.

- optional
- array

onValue

A MIDI value to be transferred when the parent control goes to active state. On a receiving side, it is the value that switches the parent control to the active state. For example, a Pad is highlighted. When onValue is not defined, MIDI transmission is ignored.

- optional
- numeric
- default = undefined (ignore)
- min = 0
- max = 16383

offValue

A MIDI value to be transferred when the parent control goes to inactive state. When offValue is not defined, MIDI transmission is ignored.

- optional
- numeric
- default = undefined (ignore)
- min = 0
- max = 16383

lsbFirst

The lsbFirst is a flag that forces Electra to swap LSB and MSB value bytes of 14-bit MIDI parameters. It can be used in combination with cc14 and nrpn MIDI messages.

- optional
- boolean
- default = false

twosComplement

The twosComplement is a flag to instruct electra to use Two's complement represetation of negative numbers.

- optional
- boolean
- default = false

ELECTRA ONE

Control types

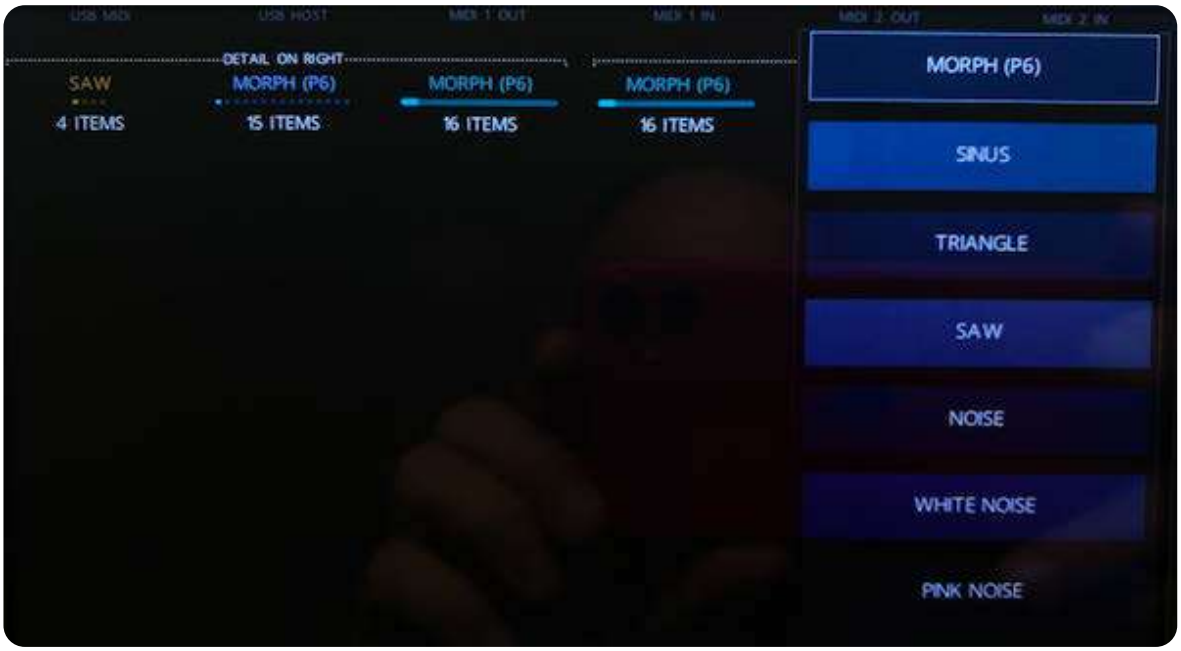
Faders

A demonstration of the preset that is visually compatible with version 1.

A simple demo of version 2 format

Scrollable lists

A long touch (longer than 0.5sec) on a control will show up a window with a large version of the control. Lists have been reworked so that they become swipeable containers of list items. We would like to get close to tablet-like behavior.



A demo of several list controls

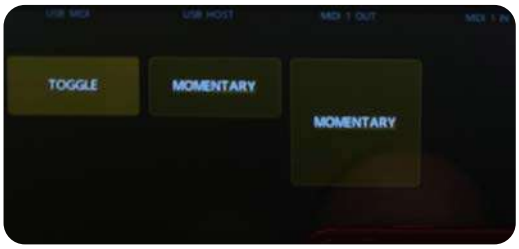
Lists with bitmap data

A picture is worth of thousand words. Often it is hard to describe a list item with words. For those situations, we added a way to provide graphical bitmap data. The idea is at its very beginning, but feel it is an important addition that needs to be developed further.

A demo of a lists with bitmap images

Pads

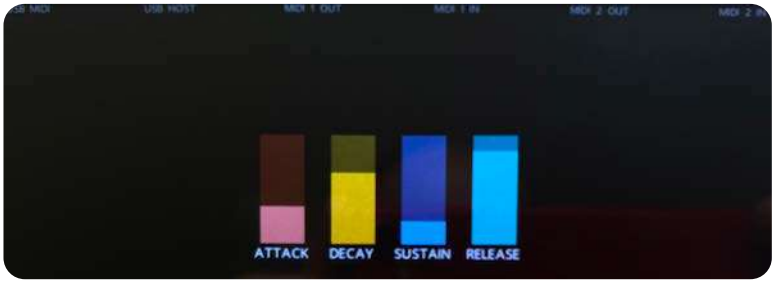
Pads are primarily meant to trigger notes, act as on/off switches, or selectors of predefined values. They can be configured either as momentary switches or toggles.



A demo of various types of pads

Vertical faders

Vertical faders are equivalents of horizontal faders. Although they can be assigned to pots, their true value might be fact that their values can be modified with on-screen touch.



A demo of vertical faders

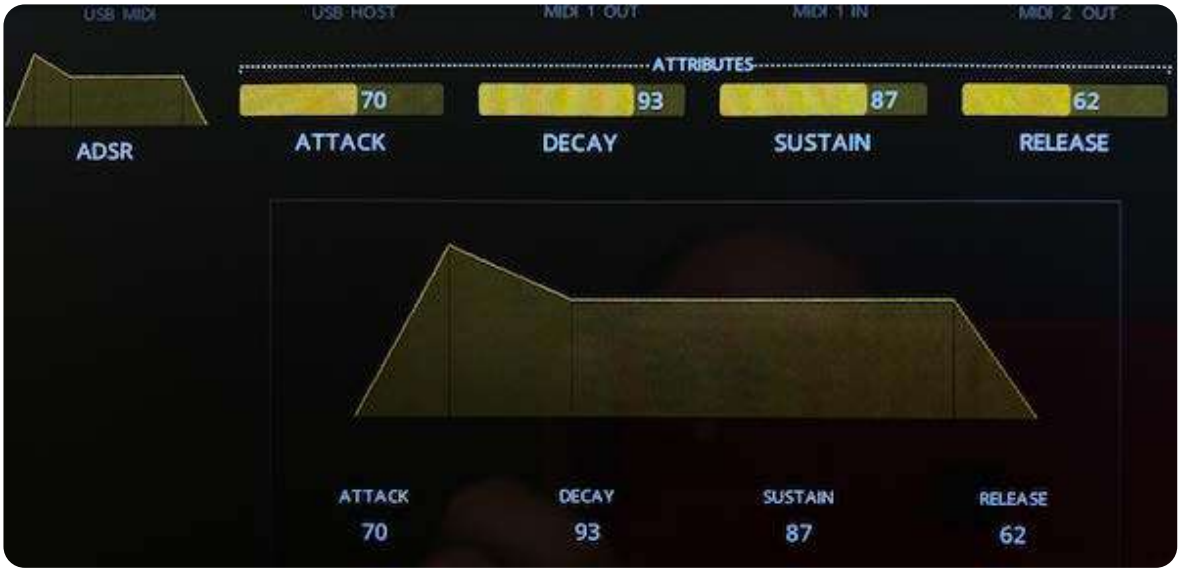
Dials

Many people asked about these. We are still not convinced if they are really needed.

A demo of dial knob controls

ADSR envelope

The envelope may have one of its parameters assigned to a pot. On long touch, a detail window is shown and provides access to all envelope parameters.



A demo of an ADSR envelope

ADR envelope

Provides the same functionality as ADSR, but for ADR envelope.

A demo of an ADR envelope

DX7 (multi-stage) envelope

Provides the same functionality as ADSR, but for DX7 multi-stage envelope.

A demo o a DX7 envelope

Macro

With version 2 it is possible to assign one pot to several controls. In such situation, turning the pot will affect the values of all assigned controls.

A demo of "macros"

ELECTRA ONE

Configuration format description

This document describes the format of the Electra One configuration file. The configuration file holds instructions on how Electra will operate.

Preset JSON format

JSON schema

This is a WIP (work-in-progress) version of JSON schema of Electra configuration file format

Minification

We strongly advise minifying the JSON data prior to transfer to Electra One over the USB MIDI protocol. Minification greatly affects the amount of data that must be transferred, therefore it speeds up the time of uploading and downloading configuration files.

Top level objects

A preset has a number of top-level objects. These are either simple elements providing info about the preset itself or complex objects that define the structure and data of the preset.

```
{
   "version":1,
   "router":{
   },
   "presetBanks":[
   ],
   "usbHostAssigments":[
   ],
   "midiControl":[
   ]
}
```

version

Provides information about the version of the configuration file. Electra controller uses version information to distinguish between various configuration file formats.

- mandatory
- numeric

router

An object that sets rules for forwarding of inbound MIDI messages to output ports of available MIDI interfaces.

- optional
- object

example:

```
"router":{
   "usbDevToMidiIo":   true,
   "usbDevToUsbHost":  true,
   "midiIoToUsbDev":   true,
   "midiIoToUsbHost":  true,
   "usbHostToMidiIo":  true,
   "usbHostToUsbDev":  true
}
```

Attributes:

attribute	description
usbDevToMidiIo	forwarding USB Device messages to MIDI IO ports
usbDevToUsbHost	forwarding USB Device messages to USB Host ports
midiIoToUsbDev	forwarding MIDI IO to USB Device ports
midiIoToUsbHost	forwarding MIDI IO to USB Host ports
usbHostToMidiIo	forwarding USB Host to MIDI IO ports
usbHostToUsbDev	forwarding USB Host to USB Device ports

Values:

value	description
false	forwarding disabled
true	forwarding enabled

presetBanks

An array of preset bank definitions. A preset bank is a named collection of 12 presets.

- optional
- array

example:

```
"presetBanks":[
   {
      "id":1,
      "name":"FX UNITS",
      "color":"FFFFFF"
   },
   {
      "id":2,
      "name":"SAMPLERS",
      "color":"529DEC"
   }
]
```

usbHostAssigments

An array of instructions on how to assign USB Host devices to Electra's ports.

- optional
- array

examples:

```
"usbHostAssigments":[
   {
      "pattern":"launchpad",
      "port":3
   },
   {
      "pattern":"keycontrol",
      "port":1
   }
]
```

midiControl

An array of assignments of MIDI messages to Electra internal commands, such as page and preset switching.

- optional
- array

example:

```
"midiControl":[
   {
      "event":"switchPage",
      "eventParameter":1,
      "midiMessage":"cc7",
      "parameterNumber":1
   },
   {
      "event":"switchPageNext",
      "midiMessage":"note",
      "parameterNumber":36
   },
   {
      "event":"switchPreset",
      "eventParameter":1,
      "midiMessage":"program",
      "parameterNumber":1
   },
   {
      "event":"switchPresetPrev",
      "midiMessage":"note",
      "parameterNumber":50
   }
]
```

presetBank

A preset bank is a named collection of 12 presets. Electra support 6 presets banks. Each bank has a name and color assigned.

example:

```
{
   "id":1,
   "name":"FX UNITS",
   "color":"FFFFFF"
}
```

id

A unique identifier of the preset bank.

- mandatory
- numeric
- min = 1
- max = 6

name

A user-defined name of the bank. The name makes it easier for users to remember and identify the banks.

- mandatory
- string
- minLength = 1
- maxLength = 20

color

A 24-bit RGB code of the control's color. The colors are limited to six predefined colors.

- optional
- string
- default = FFFFFF
- enum
- FFFFFF (white)
- F45C51 (red)
- F49500 (orange)
- 529DEC (blue)
- 03A598 (green)
- C44795 (pink)

usbHostAssignment

USB Host assignments allow automating the assignment of MIDI devices connected to the USB host interface to Electra's internal ports. A string pattern is used to match the product name of USB devices.

example:

```
{
   "pattern":"launchpad",
   "port":3
}
```

pattern

A string to be matched with USB Device product name. If the pattern is found within the product name, the device is assigned to the specified port. The matching algorithm is case-insensitive. The first match results in the port assignment, subsequent matches are ignored.

- mandatory
- string
- minLength = 1
- maxLength = 20

port

An identifier of Electra's internal port. The port is expressed as a number, where Port CTRL is represented with a port 3.

- numeric
- min = 1
- max = 3

midiControl

A rule that tells what internal event will be triggered after receipt of the given MIDI message. The MIDI message is defined by the MIDI message type and the parameter number.

example:

```
{
   "event":"switchPreset",
   "eventParameter":1,
   "midiMessage":"program",
   "parameterNumber":1
}
```

event

A command to be triggered.

- mandatory
- string
- enum
- switchPreset
- switchPresetPrev
- switchPresetNext
- switchPage
- switchPagePrev
- switchPageNext

eventParameter

An optional parameter of the event, eg. page number.

- optional
- numeric

ELECTRA ONE

Parsing SysEx messages

> WARNING
>
> Note, we are currently adjusting the SysEx parse. This article is kept here only for informative purposes so that users can understand the principle of SysEx message parsing. The document will be updated soon.

Electra One supports MIDI communication in both directions. Obviously, you can use Electra to send MIDI messages to your instruments, but you can also use it to receive MIDI messages and update the values of your Controls accordingly.

You do not have to do much extra work for channel messages such as CC7, CC14, and NRPN. Electra will automatically update the value of Controls that are linked to a given MIDI parameter/controller. With SysEx, however, Electra must be provided information how to parse incoming SysEx data. This tutorial provides an introduction to parsing SysEx messages.

A simple example

The SysEx parsing instructions can be included in both Electra presets and instrument files. We encourage users to develop instrument files. Instrument files allow other people to easily integrate supported instruments in their own presets.

The following example is a simplified Instrument file that allows bi-directional synchronization of two parameters. We use one **Voice** parameter and one **Performance** parameter on purpose, to show that multiple SysEx messages can be used to synchronize Electra and the connected synthesizer.

```
{
  "id": "yamaha-tx7",
  "name": "Yamaha TX7",
  "manufacturer": "Yamaha",
  "manufacturerId": "yamaha",
  "categories": [
    {
      "id": "op1",
      "label": "Operator 1"
    }
  ],
  "overlays": [
    {
      "id": 1,
      "name": "Op Modes",
      "items": [
        {
          "value": 0,
          "label": "Fixed"
        },
        {
          "value": 1,
          "label": "Ratio"
```

```
        }
      ]
    }
  ],
  "parameters": [
    {
      "id": 1,
      "type": "fader",
      "name": "Rate 2",
      "min": 0,
      "max": 99,
      "categoryId": "op6",
      "msg": "SysEx",
      "data": ["43", "10", "00", "01",
        { "type": "value",
          "rules": [
            { "id": 1, "bPos": 0, "size": 7 }
          ]
        }
      ]
    },
    {
      "id": 161,
      "name": "Keyboard mode",
      "categoryId": "setup",
      "type": "list",
      "overlayId": 3,
      "msg": "SysEx",
      "data": ["43", "10", "04", "02",
        { "type": "value",
          "rules": [
            { "id": 161, "bPos": 0, "size": 7 }
          ]
        }
      ]
    }
  ],
  "patch": [
    {
      "request": ["43", "20", "00"],
      "responses": [
        {
          "header": ["43", "00", "00", "01", "1B"],
          "rules": [
            {
              "id": 1,
              "pPos": 0,
              "byte": 1,
              "bPos": 0,
              "size": 7
            }
          ]
        }
      ]
    },
    {
      "request": ["43", "20", "01"],
      "responses": [
        {
```

```
          "header": ["43", "00", "01", "00", "5E"],
          "rules": [
            {
              "id": 161,
              "pPos": 0,
              "byte": 2,
              "bPos": 0,
              "size": 7,
              "msg": "SysEx"
            }
          ]
        }
      ]
    }
  ]
}
```

This is a complete Instrument file, the part that takes care of SysEx message processing is located in the `patch` element.

```
"patch": [
    {
      "request": ["43", "20", "00"],
      "responses": [
        {
          "header": ["43", "00", "00", "01", "1B"],
          "rules": [
            {
              "id": 1,
              "pPos": 0,
              "byte": 1,
              "bPos": 0,
              "size": 7,
              "msg": "SysEx"
            }
          ]
        }
      ]
    },
    {
      "request": ["43", "20", "01"],
      "responses": [
        {
          "header": ["43", "00", "01", "00", "5E"],
          "rules": [
            {
              "id": 161,
              "pPos": 0,
              "byte": 2,
              "bPos": 0,
              "size": 7,
              "msg": "SysEx"
            }
          ]
        }
      ]
    }
  ]
```

The Patch element

The patch element may consist of a number of objects that describe `request` and possible `responses`.

A request a sequence of bytes (expressed in hexadecimal notation) that is sent to your synthesizer when you want to read its settings. One occasion is pressing the [PATCH REQUEST] button.

The responses array then describes the format of expected SysEx messages that the instrument will send after receiving the request message. The response consists of a message `header` and the `rules`. The header is used to identify the incoming SysEx message and the rules provide instructions on how to parse the bytes of that SysEx message and how to assign parsed data to Electra's controls.

Let's look at an example...

When Yamaha TX7 receives SysEx message

```
F0h 43h 20h 00h F7h
```

It is supposed to respond with **One Voice data dump** SysEx message. It is a 163 bytes long SysEx message starting with a number of constant bytes followed by the sequence of bytes with voice parameter values. A fragment of such a response might look like this:

```
F0h 43h 00h 00h 01h 1Bh 01h 02h 03h .... F7h
```

These are the above messages translated to Electra's Instrument file:

```
  "patch": [
    {
      "request": ["43", "20", "00"],
      "responses": [
        {
          "header": ["43", "00", "00", "01", "1B"],
          "rules": [
            {
```

The parsing rules

The format of the parsing rules is essentially identical to the syntax that you use for the composition of SysEx templates, you might want to take a look at SysEx templates tutorial..

Each rule describes extraction of a byte or individual bits from given SysEx message byte and an application of this extracted value to Electra's internal parameter storage.

```
{
  "id": 1,
  "pPos": 0,
  "byte": 1,
  "bPos": 0,
  "size": 7,
  "msg": "SysEx"
}
```

attribute	function
id	id of parameter
pPos	bit position within the parameter
byte	position of the byte in the SysEx message, byte 0 is the first one after the header bytes
bPos	bit position within the SysEx byte
size	number of bits to be used
msg	type of parameter

The best is to describe on real world examples:

get 7-bit value of SysEx byte 1 and assign it to parameter 1 of the SysEx type:

```
{
  "id": 1,
  "pPos": 0,
  "byte": 1,
  "bPos": 0,
  "size": 7,
  "msg": "SysEx"
}
```

get value expressed with bits 2 and 3 in the SysEx byte 10 and assign it to parameter 5:

```
{
  "id": 1,
  "pPos": 0,
  "byte": 10,
  "bPos": 2,
  "size": 2,
  "msg": "SysEx"
}
```

compose value of parameter 65 out of SysEx bytes 10 (MSB) and 11 (LSB):

```
{
  "id": 65,
  "pPos": 0,
  "byte": 10,
  "bPos": 0,
  "size": 7,
  "msg": "SysEx"
},
{
  "id": 65,
  "pPos": 8,
  "byte": 11,
  "bPos": 0,
  "size": 7,
  "msg": "SysEx"
}
```

extract parameter 1 and parameter 5 out of SysEx byte 10:

```
{
  "id": 1,
  "pPos": 0,
  "byte": 10,
  "bPos": 0,
  "size": 3,
  "msg": "SysEx"
},
{
  "id": 5,
  "pPos": 0,
  "byte": 10,
  "bPos": 4,
  "size": 4,
  "msg": "SysEx"
}
```


The Instrument file inspector

As it may become quite hard to fully understand the structure of the instrument file, we have made a tool for you that visualizes the parameters of the instrument file and their assignment to the bytes of SysEx messages Instrument file inspector