

SysEx implementation

The Electra One MIDI controller can be configured, programmed, and fully controlled using MIDI SysEx (System Exclusive) messages. This document explains how SysEx messages are used to communicate with the controller — including how to send data, request information, and manage its behavior — all through the USB MIDI interface.

In fact, the full Electra One web-based editor running `app.electra.one` is built entirely on top of this very same SysEx API.

Whether you're building your own tools or integrating Electra One into a larger MIDI setup, this guide will help you understand the key SysEx commands and how to use them effectively.

Note

To utilize the SysEx Implementation described in this document, you must have Firmware version 4.0 or later installed.

Byte Notation

All byte values in this document are written in hexadecimal format, using the `0xNN` notation, where `NN` is a value between `00` and `FF`.

Unless otherwise noted, all numbers should be interpreted as hexadecimal. If decimal notation is used, it will be clearly stated.

Manufacturer SysEx Id

Every SysEx message must include a manufacturer Id to identify which device or brand the message is intended for. This helps prevent conflicts and ensures that messages are correctly interpreted by the right device.

Electra One uses the official Manufacturer SysEx Id assigned by the MIDI Association to Electra One s.r.o.:

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

This Id must appear at the beginning of every SysEx message sent to an Electra One controller.

The Management Port

Electra One SysEx messages can be sent through any of the controller's USB device MIDI ports. However, it is recommended to use the `Electra Controller CTRL` port whenever possible. Using this dedicated port helps separate Electra's management SysEx messages from regular MIDI traffic.

On some systems, this port may appear under a different name:

- **Windows:** `MIDIIN3`
- **Linux:** `PORT 3`

Responses to requests are always sent back through the same USB port the request came from.

Event notifications (triggered by user interaction on the controller) are sent by default to the **Electra Controller CTRL** port. This behavior can be changed — see [Set the MIDI port for UI events](#) for more details.

Request / Response Handshake

Electra One uses a simple request–response protocol for exchanging data over SysEx.

Each message sent to the controller expects a specific type of response. This handshake ensures reliable communication and lets you confirm whether the request was received and handled correctly.

Requests sent to Electra One fall into two main categories:

- **Data Queries** – Used to request information from the controller.
These requests do not modify any state or data on the device. They only fetch and return data.
- **Commands** – Used to perform actions or change data on the controller.
These requests do modify the controller’s internal state or configuration.

When a Data Query is sent, Electra One responds with a message containing the requested data in JSON format.

When a Command is sent, Electra One replies with either:

- **ACK** (Acknowledged) – The command was successfully received and executed.
- **NACK** (Not Acknowledged) – The command failed (e.g., due to incorrect structure or invalid data).

ACK and **NACK** responses let you know if the controller accepted your request, so your application can respond in the right way.

Transaction Id

There may be situations where multiple Commands are sent at the same time.

In these cases, it can be difficult to tell which **ACK** or **NACK** response belongs to which request. To solve this, Commands can optionally include a **Transaction Id**. This Id helps you track and match each response to its original request — especially useful when multiple requests are being processed asynchronously or out of order.

If used, the Transaction Id must be inserted immediately after the Manufacturer SysEx Id using the following format:

```
0x00 0xNN 0xMM
```

Where:

- **0xNN** is the least significant 7 bits (LSB) of the transaction Id
- **0xMM** is the most significant 7 bits (MSB) of the transaction Id

If a **Transaction Id** is included in the Command, the corresponding **ACK** or **NACK** response will also include the same two bytes, allowing you to match the response to the original command. See, [ACK / NACK](#) for more details.

Example

The transaction Id 4183 should be transferred as

```
0x00 0x77 0x20
```

Electra One firmware versions **earlier than 4.0.0 do not support Transaction Ids**. If you include a `Transaction Id` with a command on older firmware, it will not work as expected. For this reason, your software should always check the firmware version before using this feature.

Operation and Resource Bytes

After the Manufacturer SysEx Id (and optional `Transaction Id`, if used), the next two bytes in the message define:

1. **The Operation** – what kind of action should be performed
2. **The Resource** – what type of data the action should apply to

These two bytes are essential for telling the controller exactly what you're asking it to do and where the action should be applied.

Operations

The Operation byte tells Electra One whether the request is a Data query, or a Command that

The operation types include:

- `upload` – upload new data (e.g. a preset or Lua script)
- `request` - query data stored on the controller
- `create` – create a data resource (eg. snapshot)
- `update` – make a persistent change to a data resource
- `remove` – remove data permanently
- `switch` - change active resource
- `updateRuntime` - update run-time volatile data

There are additional special operations, which will be described later in this document.

Resource Byte

The second byte identifies the data resource the operation should target.

It tells the controller what kind of data is being queried or changed.

Some example resources include:

- `Preset` – the entire preset configuration
- `Control` – a single control within a preset
- `System` – system-level settings or configuration
- `File` – a file or file location
- `Device` – information about connected MIDI devices

There are many types of data resources available. You'll find their descriptions later in this document.

By combining the Operation and Resource bytes, your message tells Electra One exactly:

- What to do (operation)
- And what to do it with (resource)

Payload

Most operations require additional data to work, for example, a preset in JSON format or the number of a preset slot to activate. This extra data is called the Payload, and it comes immediately after the Operation and Resource bytes in the SysEx message.

Depending on the type of operation, the payload format can vary. Some operations require binary payloads, others require data formatted as JSON.

While handling different payload formats may add a bit of complexity for software developers using the SysEx API, this design greatly improves performance by avoiding unnecessary JSON parsing when it's not needed.

When transferring JSON payload, the individual bytes must be transferred using their ASCII codes and stay strickly in 7-bit range.

Message Structure

Now that we've covered all the components of a message, we can take a look at the overall structure of a SysEx API message.

Without Transaction Id

A SysEx API message without a Transaction Id:

```
0xF0 manufacturer-id operation resource payload 0xF7
```

for example a Command with binary data Payload:

```
0xF0 0x00 0x21 0x45 0x02 0x05 0x01 0x00 0x05 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x05** Operation Remove
- 0x01** Data resource Preset
- 0x00** **0x05** Payload (bankNumber and slot number)
- 0xF7** SysEx closing byte

or a Command with mixed binary and JSON data Paylaod:

```
0xF0 0x00 0x21 0x45 0x14 0x07 0x02 0x00 {"name":"Track2"} 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x14** Operation Update runtime

0x07 Data resource Control

0x00 **0x05** **{"name":"Track2"}** Payload (control Id LSB, control Id MSB, JSON data)

0xF7 SysEx closing byte

upon processing the command, the Electra One controller will respond with the **ACK** or **NACK** according to the result of the operation.

an example of the **ACK** response:

```
0xF0 0x00 0x21 0x45 0x7E 0x01 0x00 0x00 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x7E Status

0x01 Acknowledged

0x00 **0x00** No transaction Id available

0xF7 SysEx closing byte

With Transaction Id

A SysEx API message with a Transaction Id:

```
0xF0 manufacturer-id 0x00 transaction-id operation resource payload 0xF7
```

for example a Command with binary data Payload:

```
0xF0 0x00 0x21 0x45 0x00 0x77 0x20 0x02 0x05 0x01 0x00 0x05 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x00 Transaction Id flag

0x77 **0x20** Transaction Id 4183

0x05 Operation Remove

0x01 Data resource Preset

0x00 **0x05** Payload (bankNumber and slot number)

0xF7 SysEx closing byte

Upon processing the command, the Electra One controller will respond with either an `ACK` or `NACK`, depending on the result of the operation. If a Transaction Id was included in the request, it will be echoed back in the `ACK` / `NACK` response.

an example of the `NACK` response:

```
0xF0 0x00 0x21 0x45 0x7E 0x00 0x77 0x20 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x7E` Status
`0x00` Not-Acknowledged
`0x77` `0x20` Transaction Id 4183
`0xF7` SysEx closing byte

Controller events

A Controller Event is a special type of SysEx message that Electra One sends out when something important occurs. These events are typically triggered by user actions or as part of handling incoming SysEx messages or external MIDI control commands.

The controller may send an event message when:

- Switching a page
- Switching a preset
- Changing the Control Set
- Touching any knob
- Connecting a USB device
- Acknowledging a command
- Sending a log message at the user's request

Some event messages are always sent when the event occurs. Others require the user (or software) to explicitly subscribe in order to receive them. Details on which events require subscriptions — and how to subscribe — are provided in the sections below.

Querying data from the controller

This section covers the set of queries used to retrieve information from the Electra One controller. The data returned may include runtime information, static configuration, or files stored internally on the controller.

Get Electra info

The Electra One MIDI controller can provide information about its hardware and the currently loaded firmware upon request.

This call is useful when you need to check if the connected Electra One is working properly and to retrieve details about the firmware it is running.

For example, the Electra App and the Electra Editor use this call to verify that the controller is connected correctly and to display the connection status 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": "v4.0.0",
  "versionSeq": 400000000,
  "serial": "E02-5301787f",
  "hwRevision": "3.0"
}
```

Get Run-time information

A request call to fetch the run-time information from the Electra firmware. Only the information about free memory is included at the present time.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x7E 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x7E` Run-time information
- `0xF7` SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x7E runtime-json-data 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x01` Data dump
- `0x7E` Run-time information
- `runtime-json-data` JSON document with info about run-time data
- `0xF7` SysEx closing byte

An example of runtime-json-data

```
{
  "freePercentage": 85
}
```

Get Preset

Get preset request retrieves the preset JSON stored in a specific preset slot on the controller. If no bank number or slot number is provided, the controller will return the preset from the currently active slot. If both parameters are provided, the controller will fetch the preset from the specified bank and slot.

A preset is stored as a `preset.json` file in the preset slot.

Request

Retrieve the JSON of the currently active preset:


```
0xF0 0x00 0x21 0x45 0x02 0x01 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x01` Preset file
- `0xF7` SysEx closing byte

Retrieve a preset by specifying its bank number and slot number:

```
0xF0 0x00 0x21 0x45 0x02 0x01 bankNumber slot 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x01` Preset file
- `bankNumber` Bank number (0 .. 5)
- `slot` Slot (0 .. 11)
- `0xF7` SysEx closing byte

Response

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

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

Electra One MIDI controller responds with the SysEx message that has exactly the same format as the Preset upload message. Thus, a SysEx message downloaded with the Get preset call can be used to upload the preset to Electra's active preset slot later on.

Detailed information about `preset-json-data` is provided at [Preset format description](#)

An example of preset-json-data

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

Get Lua script

Get Lua script request retrieves the main Lua script in a specific preset slot on the controller. If no bank number or slot number is provided, the controller will return the Lua script from the currently active slot. If both parameters are provided, the controller will fetch the Lua script from the specified bank and slot.

The main Lua script refers to the script file that runs when the preset is initialized. This request only retrieves the main script, it cannot be used to fetch additional Lua files. Any extra Lua files must be accessed separately using the SysEx File Transfer API.

A Lua script is stored as a `main.lua` file in the preset slot.

Request

Retrieve the Lua script of the currently active preset:

```
0xF0 0x00 0x21 0x45 0x02 0x0C 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x0C` Lua script file
- `0xF7` SysEx closing byte

Retrieve a Lua script by specifying its bank number and slot number:

```
0xF0 0x00 0x21 0x45 0x02 0x0C bankNumber slot 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x02 Query data
0x0C Lua script file
bankNumber Bank number (0 .. 5)
slot Slot (0 .. 11)
0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x0C script-script-code 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x01 Data dump
0x0C Lua Script file
script-script-code bytes representing a script code of the Electra One Lua script application
0xF7 SysEx closing byte

Detailed information about developing Lua script applications is provided at [Electra One Lua Extension](#) documentation.

An example of script-script-code

```
-- Demo application

-- the Setup
clockCounter = 0
beatEnabled = 0

-- User functions
function myPrint(text)
    print("my Lua: " .. text)
end

-- Standard callbacks
function midi.onClock(midiInput)
    if beatEnabled == 1 then
        if math.mod(clockCounter, 24) == 0 then
```

```

        myPrint("midi beat: interface=" .. midiInput.interface)
    end
end
clockCounter = clockCounter + 1
end

function onButtonDown(buttonId)
    myPrint("button " .. buttonId .. " pressed")

    if buttonId == BUTTON_1 then
        myPrint("Beat enabled")
        beatEnabled = 1
    elseif buttonId == BUTTON_4 then
        myPrint("Beat disabled")
        beatEnabled = 0
    end
end
end

```

Get Device overrides

This request retrieves the Device overrides stored in a specific preset slot on the controller. If no bank or slot number is provided, the controller will return the overrides from the currently active preset. If both parameters are provided, it will return the overrides from the specified bank and slot.

A Device override is a custom modification of the devices used in a preset. It allows users to change the MIDI ports and channels assigned to devices without modifying the preset itself, making it easier to adapt presets to different setups or hardware configurations.

A Device overrides definition is stored as a `devices.json` file in the preset slot.

Request

Retrieve the Device overrides of the currently active preset:

```
0xF0 0x00 0x21 0x45 0x02 0x0F 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x0F` Preset devices
- `0xF7` SysEx closing byte

Retrieve a Device overrides by specifying its bank number and slot number:

```
0xF0 0x00 0x21 0x45 0x02 0x0F bankNumber slot 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x02 Query data
0x0F Preset devices
bankNumber Bank number (0 .. 5)
slot Slot (0 .. 11)
0xF7 SysEx closing byte

Response

0xF0 0x00 0x21 0x45 0x01 0x0F preset-devices-json-data 0xF7

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x01 Data dump
0x0F Preset devices
preset-devices-json-data JSON document with a list of preset device overrides
0xF7 SysEx closing byte

An example of preset-devices-json-data

```
{
  "version":1,
  "devices":[
    {
      "id":1,
      "name":"Selection Device",
      "port":"port1",
      "channel":4,
      "rate":0
    },
    {
      "id":2,
      "name":"OP-XY",
      "port":"port1",
      "channel":1,
      "rate":0
    }
  ]
}
```

Get Persisted data

This request retrieves the persisted preset data stored in a specific preset slot on the controller. If no bank or slot number is provided, the controller returns the persisted data from the currently active slot. If both parameters are provided, the data is retrieved from the specified bank and slot.

Persisted preset data is a JSON file that contains a Lua table previously saved using the `persist()` function. Preset developers can use this feature to store custom configuration settings, runtime values, and other important data that should remain available even after the controller is restarted.

A Persisted data is stored as a `data.json` file in the preset slot.

Request

Retrieve persisted data of the currently active preset:

```
0xF0 0x00 0x21 0x45 0x02 0x12 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x12` JSON data file
- `0xF7` SysEx closing byte

Retrieve persisted data by specifying its bank number and slot number:

```
0xF0 0x00 0x21 0x45 0x02 0x12 bankNumber slot 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x12` JSON data file
- `bankNumber` Bank number (0 .. 5)
- `slot` Slot (0 .. 11)
- `0xF7` SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x12 datafile-json-data 0xF7
```

0xF0 SysEx header byte
0x00 0x21 0x45 Electra One MIDI manufacturer Id
0x01 Data dump
0x12 Persisted JSON data file
datafile-json-data JSON data saved with a Lua persist() function
0xF7 SysEx closing byte

An example of datafile-json-data

```

{
  array = { 1, 2, 3 },
  objArray = {
    { key1 = "text" },
    { key2 = 1.2 },
    { key3 = true }
  },
  number = 1.42,
  text = "hello table",
  boolean = false
}

```

Get Performance

Get Performance request retrieves the performance JSON stored in a specific preset slot on the controller. If no bank number or slot number is provided, the controller will return the performance data from the currently active slot. If both parameters are provided, the controller will fetch the performance from the specified bank and slot.

A Performance is a structured JSON file that defines a custom page made up of controls and macro controls that reference existing controls within the preset. It allows users to build a personalized performance view with re-arranged layout, without modifying the original preset.

A Performance is stored as a **performance.json** file in the preset slot.

Request

Retrieve the performance of the currently active preset:

```
0xF0 0x00 0x21 0x45 0x02 0x11 bankNumber slot 0xF7
```

0xF0 SysEx header byte
0x00 0x21 0x45 Electra One MIDI manufacturer Id
0x02 Query data
0x11 Performance

0xF7 SysEx closing byte

Retrieve the performance of the currently active preset:

```
0xF0 0x00 0x21 0x45 0x02 0x11 bankNumber slot 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x02 Query data

0x11 Performance

bankNumber Bank number (0 .. 5)

slot Slot (0 .. 11)

0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x11 performance-json-data 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x01 Data dump

0x11 Performance

performance-json-data Performance JSON data

0xF7 SysEx closing byte

Detailed information about **performance-json-data** is provided at [Performance format description](#)

An example of performance-json-data

```
{
  "version":1,
  "references":[
    {
      "controlSetId":1,
      "potId":1,
      "controlId":1,
      "name":"Fader A"
    },
    {
```



```

    "controlSetId":1,
    "potId":6,
    "valueRefs":[
      {
        "controlId":1,
        "valueId":"value",
        "channel":1,
        "mode":"dataPipe",
        "pipe": {
          "name":"output",
          "bankNumber":5,
          "slot":1
        }
      },
      {
        "controlId":2,
        "valueId":"value",
        "mode":"setValue",
        "depth":50
      }
    ],
    "name":"All faders"
  }
],
"groups":[
  {
    "id":4,
    "pageId":1,
    "name":"GROUP LABEL",
    "color":"ffffff",
    "bounds":[
      14,
      6,
      993,
      171
    ]
  }
]
}

```

Get Configuration

A request to fetch the current Electra One configuration. This configuration file defines the general behavior and settings of the controller

Request

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

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

Detailed information about configuration-json-data is provided at [Configuration format description](#)

An example of configuration-json-data

```
{
  "version": 2,
  "router": {
    ...
  },
  "presetBanks": [
    ...
  ],
  "usbHostAssignments": [
    ...
  ],
  "midiControl": [
    ...
  ]
}
```

Get List of presets

This request retrieves a list of all presets that are currently saved on the controller.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x04 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x02` Query data
- `0x04` Preset list
- `0xF7` SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x04 preset-list-json-data 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x01` Data dump
- `0x04` Preset list
- `preset-list-json-data` JSON document with a list of presets
- `0xF7` SysEx closing byte

An example of preset-list-json-data

```
{
  "version":1,
  "current":{
    "bankNumber":5,
    "slot":0
  },
  "presets":[
    {
      "slot":0,
      "bankNumber":5,
      "name":"EMM Ctrl 10.52",
      "projectId":"4bJi5KIqqQB8th333Na7",
      "hasLua":true,
      "isPinned":false
    },
    {
      "slot":3,
      "bankNumber":5,
```

```

    "name": "VCV Rack 2",
    "projectId": "4rIzUF8a60kXiYsyv1TN",
    "hasLua": true,
    "isPinned": false
  },
  {
    "slot": 11,
    "bankNumber": 5,
    "name": "Rhodes Chroma",
    "projectId": "HxepQNRfBdIo0CyMyCqu",
    "hasLua": false,
    "isPinned": false
  }
]
}

```

Get Preset slot information

This request retrieves information about the Preset slot and the preset stored in it.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x08 bankNumber slot 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x02 Query data
0x04 Preset slot
bankNumber Bank number (0 .. 5)
slot Slot (0 .. 11)
0xF7 SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x04 preset-list-json-data 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x01 Data dump
0x08 Preset slot
preset-slot-json-data JSON document with a preset slot information

0xF7 SysEx closing byte

An example of preset-slot-json-data

```
{
  "version":1,
  "bankNumber":0,
  "slot":0,
  "name":"Demo preset",
  "projectId":"IJopUYMf2TW1PH7GNYxD",
  "hasLua":true,
  "isPinned":false,
  "files":[
    {
      "name":"preset.json",
      "md5":"b58f9ee9391b7e49f471fcbb2deb536c"
    },
    {
      "name":"main.lua",
      "md5":"7f00373c5818f254ef19a82217a18be0"
    },
    {
      "name":"devices.json",
      "md5":"5dec6bf7eebb098dda3d706fe6c2f115"
    }
  ]
}
```

Get List of snapshots

A request to fetch the list of snapshots for a preset associated with a specific projectId.

Request

0xF0 0x00 0x21 0x45 0x02 0x05 snaphost-list-request-json-data 0xF7

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x02 Query data

0x05 Snaphost list

snapshot-list-request-json-data

0xF7 SysEx closing byte

An example of snapshot-list-request-json-data

```
{
  "projectId": "IJopUYMf2TW1PH7GNYxD"
}
```

Response

```
0xF0 0x00 0x21 0x45 0x01 0x05 snapshot-list-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x01 Data dump
 0x05 Snapshot list
 snapshot-list-json-data JSON document with a list of snapshots
 0xF7 SysEx closing byte

An example of snapshot-list-json-data

```
{
  "version":1,
  "projectId":"IJopUYMf2TW1PH7GNYxD",
  "snapshots":[
    {
      "slot":0,
      "bankNumber":0,
      "name":"A0",
      "color":"FFFFFF",
      "filename":"s4380877.json"
    }
  ]
}
```

Get Snapshot data

A request to fetch snapshot data stored in a specific snapshot bank and slot.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x03 snapshot-request-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x02 Query data
 0x03 Snapshot data
 snapshot-request-json-data
 0xF7 SysEx closing byte

An example of snapshot-request-json-data

```
{
  "projectId":"IJopUYMf2TW1PH7GNYxD",
  "bankNumber":0,
  "slot":0
}
```

Response

0xF0 0x00 0x21 0x45 0x01 0x03 snapshot-json-data 0xF7

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x01 Data dump
 0x03 Snapshot data
 snapshot-json-data JSON document with snapshot data
 0xF7 SysEx closing byte

An example of snapshot-json-data

```
{
  "version":1,
  "projectId":"IJopUYMf2TW1PH7GNYxD",
  "parameters":[
    {
      "deviceId":1,
      "messageType":1,
      "parameterNumber":102,
      "midiValue":1
    },
  ],
}
```

```
{
  "deviceId":2,
  "messageType":1,
  "parameterNumber":2,
  "midiValue":38
}
```

Get List of captures

A request to fetch the list of captures for a preset associated with a specific projectId.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x31 capture-list-request-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x02 Query data
 0x31 Capture list
 capture-list-request-json-data
 0xF7 SysEx closing byte

An example of capture-list-request-json-data

```
{
  "projectId": "IJopUYMf2TW1PH7GNYxD"
}
```

Response

```
0xF0 0x00 0x21 0x45 0x01 0x31 capture-list-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x01 Data dump
 0x31 Capture list

`capture-list-json-data` JSON document with a list of snapshots

`0xF7` SysEx closing byte

An example of capture-list-json-data

```
{
  "version":1,
  "projectId":"IJopUYMf2TW1PH7GNYxD",
  "captures":[
    {
      "slot":0,
      "bankNumber":0,
      "name":"A0",
      "color":"FFFFFF",
      "filename":"s5620078.mid",
      "midiInterface":"midiUsbDev",
      "port":1
    }
  ]
}
```

Get Capture data

A request to fetch capture data stored in a specific capture bank and slot.

Request

`0xF0 0x00 0x21 0x45 0x02 0x30 capture-request-json-data 0xF7`

`0xF0` SysEx header byte

`0x00 0x21 0x45` Electra One MIDI manufacturer Id

`0x02` Query data

`0x30` Capture data

`capture-request-json-data`

`0xF7` SysEx closing byte

An example of capture-request-json-data

```
{
  "projectId":"IJopUYMf2TW1PH7GNYxD",
```

```
"bankNumber":0,
"slot":0
}
```

Response

```
0xF0 0x00 0x21 0x45 0x01 0x30 capture-data 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x01** Data dump
- 0x30** Capture data
- capture-data** packed 7-bit MIDI SMF data
- 0xF7** SysEx closing byte

Get USB Host devices

A request to fetch a list of all devices currently connected to the controller's USB Host port.

Request

```
0xF0 0x00 0x21 0x45 0x02 0x10 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x02** Query data
- 0x10** USB Host device list
- 0xF7** SysEx closing byte

Response

```
0xF0 0x00 0x21 0x45 0x01 0x10 usb-host-devices-json-data 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id

0x01 Data dump**0x10** USB Host devices**usb-host-devices-json-data** List of USB Host devices in JSON format**0xF7** SysEx closing byte

An example of usb-host-devices-json-data

```
{
  "version":1,
  "devices":[
    {
      "manufacturer":"ESI",
      "product":"Xjam",
      "serialNumber":"123456",
      "vid":9587,
      "pid":54,
      "ports":[
        {
          "devicePort":1,
          "name":"Port 1",
          "electraPort":"port1"
        }
      ]
    }
  ]
}
```

Uploading data to the controller

The commands in this section are used to upload data to the Electra One controller. They allow you to send presets, Lua scripts, and other data files directly to the device.

Since an upload is a command, the controller will respond with an **ACK** if the operation was successful, or a **NACK** if it failed.

Upload Preset

The preset upload command is used to send a new preset to the Electra One MIDI controller. The preset is always uploaded to the currently selected (active) preset slot.

Once the upload is complete, the preset is immediately activated and ready to use. An uploaded preset is stored as a **preset.json** file in the preset slot.

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

0xF0 SysEx header byte

`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x01` Upload data
`0x01` Preset JSON 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](#)

Upload Lua script

The Lua script upload command is used to upload and execute a new Electra One Lua Extension script. The script is uploaded to the currently selected (active) preset slot.

The Lua script refers to the main script file that runs when the preset is initialized. This command cannot be used to upload additional Lua files. Any extra Lua files must be uploaded separately using the SysEx File Transfer API.

An uploaded Lua script is stored as a `main.lua` file in the preset slot.

```
0xF0 0x00 0x21 0x45 0x01 0x0C script-source-code 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x01` Upload data
`0x0C` Lua Script file
`script-source-code` bytes representing a source code of the Electra One Lua script application
`0xF7` SysEx closing byte

Detailed information about developing Lua script applications is provided at [Electra One Lua script](#) documentation.

Upload Device overrides

The Device Overrides upload command is used to upload and replace the device definitions in the current preset. The overrides are uploaded to the currently selected (active) preset slot.

A Device Override is a custom modification of the devices used in a preset. It allows users to change the MIDI ports and channels assigned to devices without modifying the preset itself, making it easier to adapt presets to different setups or hardware configurations.

An uploaded Devices definition is stored as a `devices.json` file in the preset slot.

```
0xF0 0x00 0x21 0x45 0x01 0x0F preset-devices-json-data 0xF7
```

`0xF0` SysEx header byte

`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x01` Upload data
`0x0F` Preset devices
`preset-devices-json-data` JSON document with a list of preset device overrides
`0xF7` SysEx closing byte

Upload Persisted data

The Persisted data upload command is used to upload and replace the JSON data that will be interpreted as a persisted Lua table in the current preset. The data is uploaded to the currently selected (active) preset slot.

Persisted preset data is a JSON file that contains a Lua table previously saved using the `persist()` function. This data can be loaded back into a Lua table using the `recall()` function. Preset developers can use this feature to store custom configuration settings, runtime values, and other important data that should remain available even after the controller is restarted.

An uploaded Persisted data is stored as a `data.json` file in the preset slot.

```
0xF0 0x00 0x21 0x45 0x01 0x12 datafile-json-data 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x01` Upload data
`0x12` Persisted JSON data file
`datafile-json-data` JSON data saved with a Lua `persist()` function
`0xF7` SysEx closing byte

Upload Performance

The Performance upload command is used to upload and replace the performance JSON data in a specific preset slot on the controller. The data is always uploaded to the currently selected (active) slot.

A Performance is a structured JSON file that defines a custom page made up of controls and macro controls that reference existing controls within the preset. It allows users to build a personalized performance view with re-arranged layout, without modifying the original preset.

An uploaded Performance data is stored as a `performance.json` file in the preset slot.

```
0xF0 0x00 0x21 0x45 0x01 0x11 performance-json-data 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x01` Upload data

`0x11` Performance`performance-json-data` Performance JSON data`0xF7` SysEx closing byte

Upload Configuration

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

```
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](#)

Persistent commands

Persistent commands make permanent changes to the data stored on the controller. This means that any changes made using persistent commands will still be in effect even after the controller is powered off and restarted.

Remove Preset

The Remove Preset command permanently removes a preset identified by its bank number and slot.

```
0xF0 0x00 0x21 0x45 0x05 0x01 bank-number slot 0xF7
```

`0xF0` SysEx header byte`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id`0x05` Remove command`0x01` Preset`bank-number` an identifier of the preset bank (0 .. 5)`slot` an identifier of the preset slot (0 .. 11)`0xF7` SysEx closing byte

Remove Lua script

The Remove Lua Script command permanently deletes the main Lua script file associated with a specific bank number and slot. This command cannot be used to remove additional files; to delete those, use the Remove Preset Slot files command instead.

```
0xF0 0x00 0x21 0x45 0x05 0x0C bank-number slot 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x05` Remove command
- `0x0C` Lua script
- `bank-number` an identifier of the preset bank (0 .. 5)
- `slot` an identifier of the preset slot (0 .. 11)
- `0xF7` SysEx closing byte

Remove Config

The Remove Configuration command permanently deletes the configuration file from the controller.

```
0xF0 0x00 0x21 0x45 0x05 0x02 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x05` Remove command
- `0x02` Configuration file
- `0xF7` SysEx closing byte

Remove Snapshot

The Remove Snapshot command permanently deletes a snapshot from the controller.

```
0xF0 0x00 0x21 0x45 0x05 0x06 snapshot-id-json-data 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x05` Remove command

0x06 Snapshot
snapshot-id-json-data Snapshot identification JSON data
0xF7 SysEx closing byte

An example of the snapshot-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "bankNumber": 2,
  "slot": 5
}
```

Remove Capture

The Remove Capture command permanently deletes a capture from the controller.

```
0xF0 0x00 0x21 0x45 0x05 0x06 capture-id-json-data 0xF7
```

0xF0 SysEx header byte
0x00 0x21 0x45 Electra One MIDI manufacturer Id
0x05 Remove command
0x32 Capture
capture-id-json-data Snapshot identification JSON data
0xF7 SysEx closing byte

An example of the capture-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "bankNumber": 0,
  "slot": 0
}
```

Clear Preset slot

The Remove Preset command permanently removes a preset identified by its bank number and slot.


```
0xF0 0x00 0x21 0x45 0x05 0x01 bank-number slot 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x05 Remove command
0x08 Preset slot
bank-number an identifier of the preset bank (0 .. 5)
slot an identifier of the preset slot (0 .. 11)
0xF7 SysEx closing byte

Update Snapshot

The Update Snapshot command updates the attributes of an existing snapshot.

```
0xF0 0x00 0x21 0x45 0x04 0x06 snapshot-json-data 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x04 Update command
0x06 Snapshot
snapshot-json-data Snapshot JSON data
0xF7 SysEx closing byte

An example of the snapshot-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "bankNumber": 0,
  "slot": 5,
  "name": "House piano",
  "color": "E4660E"
}
```

Update Capture

The Update Capture command updates the attributes of an existing capture.

```
0xF0 0x00 0x21 0x45 0x04 0x06 capture-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x04 Update command
 0x06 Capture
 capture-json-data Capture JSON data
 0xF7 SysEx closing byte

An example of the capture-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "bankNumber": 1,
  "slot": 4,
  "name": "Synths bank",
  "color": "DD1530"
}
```

Swap Snapshots

The Swap Snapshots command exchanges the snapshots between two snapshot slots. If one of the slots is empty, the operation becomes a simple move instead of a swap.

```
0xF0 0x00 0x21 0x45 0x06 0x06 snapshot-ids-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x06 Swap command
 0x06 Snapshot
 snapshot-ids-json-data
 0xF7 SysEx closing byte

An example of the snapshot-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "fromBankNumber": 0,
  "fromSlot": 5,
  "toBankNumber": 0,
  "toSlot": 4
}
```

Swap Captures

The Swap Captures command exchanges the captures between two capture slots. If one of the slots is empty, the operation becomes a simple move instead of a swap.

```
0xF0 0x00 0x21 0x45 0x06 0x06 capture-ids-json-data 0xF7
```

0xF0 SysEx header byte
 0x00 0x21 0x45 Electra One MIDI manufacturer Id
 0x06 Swap command
 0x32 Capture
 capture-ids-json-data
 0xF7 SysEx closing byte

An example of the capture-json-data

```
{
  "projectId": "SCI1mU1v6ojnm8IojuhY",
  "fromBankNumber": 0,
  "fromSlot": 0,
  "toBankNumber": 1,
  "toSlot": 0
}
```

Runtime commands

Runtime commands change how the controller behaves while it's running, but these changes are not saved and will be lost after a restart.

Switch Preset slot

The Preset lot switch command changes the active preset slot. If the selected slot contains a preset, it will be loaded. If the slot is empty, it becomes the active slot and can be used to load a new preset.

```
0xF0 0x00 0x21 0x45 0x09 0x08 bank-number slot 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x09 Switch command

0x08 Preset slot

bank-number an identifier of the preset bank (0 .. 5)

slot an identifier of the preset slot (0 .. 11)

0xF7 SysEx closing byte

Load Preloaded preset

The Load Preloaded preset command copies a preloaded preset into a preset slot and activates it. This allows the controller to quickly load and switch to a prepared preset without using standard upload procedures.

Preloaded presets are stored in special location on the controller. Users can upload presets to these locations either by using the USB mass storage mode in the bootloader or by using the SysEx File Transfer API.

```
0xF0 0x00 0x21 0x45 0x04 0x08 preset-slot-json-data 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x04 Update command

0x08 Preset slot

preset-slot-json-data Preset slot update JSON data

0xF7 SysEx closing byte

An example of the preset-slot-json-data

```
{
  "bankNumber": 5,
  "slot": 1,
  "preset": "xot/ableton/Cabinet"
}
```

Switch Page

The Switch Page command is used to change the active page.

```
0xF0 0x00 0x21 0x45 0x09 0x0A page-number 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x09` Switch command
- `0x0A` Page
- `page-number` an identifier of the page (0 .. 11)
- `0xF7` SysEx closing byte

Switch Control Set

The Switch Control set command changes the currently selected set of knobs assigned to the on-screen controls.

```
0xF0 0x00 0x21 0x45 0x09 0x0B control-set-id 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x09` Switch command
- `0x0B` Control Set
- `control-set-id` an identifier of the page (0 .. 2)
- `0xF7` SysEx closing byte

Set Preset slot

The Set Preset Slot command changes the currently selected preset bank and slot. However, it does not activate or load the preset in that slot, unlike the Switch Preset Slot command. Instead, Set Preset Slot simply arms the slot as selected for subsequent operations, such as uploading preset files.

```
0xF0 0x00 0x21 0x45 0x14 0x08 bank-number slot 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x09` Update runtime command
- `0x08` Preset slot
- `bank-number` an identifier of the preset bank (0 .. 5)

`slot` an identifier of the preset slot (0 .. 11)

`0xF7` SysEx closing byte

Set Snapshot slot

The Set Snapshot Slot command changes the currently selected snapshot bank and slot. The selected slot is then armed for use with subsequent snapshot operations.

```
0xF0 0x00 0x21 0x45 0x14 0x09 snapshot-slot-json-data 0xF7
```

`0xF0` SysEx header byte

`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id

`0x09` Update runtime command

`0x09` Snapshot slot

`snapshot-slot-json-data` JSON document with data to identify the snapshot slot

`0xF7` SysEx closing byte

An example of snapshot-slot-json-data

```
{
  "projectId":"4bJi5KIqqQB8th333Na7",
  "bankNumber":0,
  "slot":3
}
```

Set Capture slot

The Set Capture Slot command changes the currently selected capture bank and slot. The selected slot is then armed for use with subsequent capture operations.

```
0xF0 0x00 0x21 0x45 0x14 0x33 capture-slot-json-data 0xF7
```

`0xF0` SysEx header byte

`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id

`0x09` Update runtime command

`0x09` Snapshot slot

`capture-slot-json-data` JSON document with data to identify the capture slot

0xF7 SysEx closing byte

An example of capture-slot-json-data

```
{
  "projectId":"4bJi5KIqqQB8th333Na7",
  "bankNumber":5,
  "slot":11
}
```

Execute Lua command

The Run Lua Command executes arbitrary Lua commands, effectively serving as an API endpoint for controlling Electra One presets from external devices and applications.

It allows you to remotely manage Electra One presets using Lua commands, offering a powerful way to interact with the controller from external sources. The maximum allowed length of a Lua command is 65,535 bytes.

However, we recommend keeping commands short — commands shorter than 65 bytes are executed significantly faster than longer ones.

To optimize performance, it is better to use this SysEx call to trigger Lua functions defined in a previously uploaded Lua script, rather than sending large blocks of arbitrary Lua code.

```
0xF0 0x00 0x21 0x45 0x08 0x0D lua-command-text 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x08 Execute command

0x0D Function

lua-command-text ASCII bytes representing the Lua command

0xF7 SysEx closing byte

For backwards compatibility, the following message structure is supported too:

```
0xF0 0x00 0x21 0x45 0x08 0x0C lua-command-text 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x08 Execute command

- `0x0C` Lua file
- `lua-command-text` ASCII bytes representing the Lua command
- `0xF7` SysEx closing byte

The `lua-command-text` is free form string containing Lua command to be executed. The maximum length is limited to 128 characters. It is recommended to call predefined functions.

An example of the lua-command-text

```
hideControl (1)
```

or

```
print ("Hello MIDI world!")
```

Reload Preset slot

The Reload Preset Slot command reinitializes and restarts the preset stored in the specified preset slot. The preset instance currently running in that slot will be terminated. If available, associated Lua scripts, Device Overrides, and Performance data will also be reinitialized.

```
0xF0 0x00 0x21 0x45 0x08 0x08 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x08` Execute command
- `0x08` Preset slot
- `bankNumber` Bank number (0 .. 5)
- `slot` Slot (0 .. 11)
- `0xF7` SysEx closing byte

Update control

A call to update the name, color, and visibility of a control. These changes are applied at runtime only, which means they will be lost when the Electra One is powered off.

```
0xF0 0x00 0x21 0x45 0x14 0x07 control-id-lsb control-id-msb control-update-json-data 0xF7
```


`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x14` Update runtime command
`0x07` Control
`control-id-lsb` a LSB of a controlId
`control-id-msb` a MSB of a controlId
`control-update-json-data`
`0xF7` SysEx closing byte

The `controlId` is split into two 7-bit parts: a most significant byte (MSB) and a least significant byte (LSB), using the following logic:

```
control-id-msb = controlId >> 7
control-id-lsb = controlId & 0x7F
```

The `control-update-json-data` may include up to four optional attributes: `name`, `color`, `visibility`, and `value`. When the control update command is received, any provided attributes will be applied to the control. You only need to include the attributes you want to change — all others can be left out.

Updating the `value` attribute allows you to set `value.text` only, which is equivalent to using the SysEx call for [overriding the value text](#).

An example of the control-json-data

change all attributes:

```
{
  "name": "Track 1",
  "color": "FFFFFF",
  "visible": true
}
```

one attribute only:

```
{
  "name": "Track 2"
}
```

overriding a value text:

```
{
  "value": {
    "id": "value",
    "text": "6.2dB"
  }
}
```

```
}
}
```

Note, when overriding a value text the `"id": "value"` is not required for single value controls, such as faders, pads, and relative controls. The `text` is text string of printable ASCII characters, maximum length is 15 characters. Setting the `text` string with 0 bytes length cancels the value override. When cancelled, the controller will display the current value according to its settings.

Override value text

The Override Value Text command replaces the control's current displayed value with custom text. It gives developers full control over what is shown on the screen, which is especially useful when working with Relative Control Change messages.

The custom text also overrides the output from [Lua Value formatters](#).

Although value texts can also be overridden using the Control Update command, the Override Value Text command is a more performance-optimized option, as it avoids the overhead of JSON parsing and valueId translation.

```
0xF0 0x00 0x21 0x45 0x14 0x0E control-id-lsb control-id-msb numeric-value-id text 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x14` Update runtime command
`0x0E` Override text value
`control-id-lsb` a LSB of a controlId
`control-id-msb` a MSB of a controlId
`numeric-value-id` a numeric identifier of the value within the control
`text` a text string to be displayed as value
`0xF7` SysEx closing byte

The `controlId` is split into two 7-bit parts: a most significant byte (MSB) and a least significant byte (LSB), using the following logic:

```
control-id-msb = controlId >> 7
control-id-lsb = controlId & 0x7F
```

The `numeric-value-id` identifies Electra One's MIDI port as follows. Note: the value Ids must be selected according to the type of control being used.

- `0x00` default value of single value controls (fader, pads, and relative controls)
- `0x01` attack, l1, x
- `0x02` decay, hold, release, r1, y
- `0x03` sustain, decay, break, release, l2
- `0x04` release, sustain, slope, r2

- `0x05` release, sustain, l3
- `0x06` release, r3
- `0x07` l4
- `0x08` r4

The `text` is text string of printable ASCII characters, maximum length is 15 characters. Setting the `text` string with 0 bytes length cancels the value override. When cancelled, the controller will display the current value according to its settings.

Set Bottom Bar text

The Set Bottom Bar Text command replaces the default text shown in the status bar at the bottom of the screen. The custom text remains visible until the command is called again with a string of 0 bytes in length, which clears the text and restores the default display.

```
0xF0 0x00 0x21 0x45 0x14 0x77 text 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x14` Update runtime command
- `0x77` Override text value
- `text` a text string to be displayed as value
- `0xF7` SysEx closing byte

The `text` is text string of printable ASCII characters, maximum length is 40 characters. Setting the `text` string with 0 bytes length cancels the value override. When cancelled, the controller will display the current value according to its settings.

Set Events MIDI port

The Set Event Port command sets the MIDI port used to transmit event notifications triggered by user actions on the controller (e.g., page switching).

```
0xF0 0x00 0x21 0x45 0x14 0x7B port-number 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x14` Update runtime command
- `0x7B` Select the port to transmit UI driven events on
- `port-number` a MIDI port to be used
- `0xF7` SysEx closing byte

The `port-number` identifies Electra's MIDI port as follows:

- `0x00` Port 1
- `0x01` Port 2
- `0x02` CTRL

Subscribe Events

The Subscribe Events command tells the controller which SysEx event messages should be sent out when specific events occur.

```
0xF0 0x00 0x21 0x45 0x14 0x79 event-flags 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x14` Update runtime command
`0x79` Event subscriptions
`event-flags` a byte where each bit represents specific type of event
`0xF7` SysEx closing byte

The `event-flags` a byte with the following bits (flags). The individual flags must be ORed to produce the final byte value:

- `0x00` None
- `0x01` (`bit 0`) Page events
- `0x02` (`bit 1`) Control Set events
- `0x04` (`bit 2`) USB Host events
- `0x08` (`bit 3`) Pots events
- `0x10` (`bit 4`) Touch events
- `0x20` (`bit 5`) Button events
- `0x40` (`bit 6`) Window events

Note, currently only Page events and Pots events are supported.

to reset the subscribed events, send a message with the flags set to `0x00` (None).

Control Logger Output

This system call is used to control whether Electra One sends debugging log messages. The command sets a non-volatile flag inside the controller, meaning the logger's status remains saved even after the controller is powered off.

However, startup log messages are always sent, regardless of the logger's enabled or disabled state.

Debug log messages generated by the Lua `print()` function are always sent out.

```
0xF0 0x00 0x21 0x45 0x7F 0x7D status 0x00 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x7F System call
0x7D Logger
status desired state of the logger (see below)
log-level level of verbosity of log messages (0 .. 3)
0xF7 SysEx closing byte

List of possible **status** values:

- **0x00** disable the logger
- **0x01** enable the logger

The **log-level** sets the verbosity of log messages sent by the Electra One controller. Higher log levels add extra messages to stream of log messages. The **log-level** parameter is ignored when the **status** parameter is set to **0x00**.

- **0x00** critical messages (that cannot be disabled)
- **0x01** warning messages
- **0x02** informative messages
- **0x03** tracing messages

Set Logger MIDI port

The Set Log Port command sets the USB device MIDI port used to transmit log messages. By default, log messages are sent to the **Electra Controller CTRL** port.

Note: Although log messages are considered a type of controller event, they do not follow the Event Port settings. Instead, they use their own dedicated port, which is configured using this SysEx command.

```
0xF0 0x00 0x21 0x45 0x14 0x7D port-number reserved 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x14 System call
0x7D Logger
port-number a MIDI port to be used
reserved an optional parameter. It is not currently used.

`0xF7` SysEx closing byte

The `port-number` identifies Electra's MIDI port as follows:

- `0x00` Port 1
- `0x01` Port 2
- `0x02` CTRL

Control Window repaints

The Window Repaint command provides control over the graphic component repainting process. It can be used to accumulate multiple individual repaint requests into a single repaint operation, improving overall performance.

```
0xF0 0x00 0x21 0x45 0x7F 0x7A command reserved 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x7F` System call
`0x7A` Window
`command` a command to execute
`reserved` an optional parameter. It is not currently used.
`0xF7` SysEx closing byte

The `command` must be one of the following:

- `0x00` Stop the window repainting process
- `0x01` Repaint the window and resume the window repainting process

Note: When repaints are stopped, the controller does not update any graphics on the screen and may appear unresponsive.

Control Debugging

The Debug command allows developers to enable or disable Lua script debugging. The feature is in its early initial phase. When enabled, the controller will report every single executed line of the Lua script code using the logger text message. The feature is intended to implement a full blown Lua debugger in the future.

```
0xF0 0x00 0x21 0x45 0x7C command 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id

- `0x7C` Debug
- `command` a command to execute
- `0xF7` SysEx closing byte

The `command` must be one of the following:

- `0x00` Disable debugging
- `0x01` Enable debugging

Control Midi learn

The Set MIDI Learn command enables or disables the MIDI Learn functionality on the controller. When enabled, the controller sends MIDI Learn event messages back to the host for all incoming MIDI messages.

While MIDI Learn is active, incoming MIDI messages are not processed in the standard way.

The MIDI Learn event message is described in the [Controller events](#) section.

```
0xF0 0x00 0x21 0x45 0x03 status 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x03` Midi Learn
- `status` Desired state of the MIDI learn functionality (see below)
- `0xF7` SysEx closing byte

List of possible `status` values:

- `0x00` disable the MIDI learn
- `0x01` enable the MIDI learn

Reboot

The Reboot command restarts the controller.

```
0xF0 0x00 0x21 0x45 0x7F 0x78 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x7F` System call
- `0x78` Reboot

`0xF7` SysEx closing byte

Controller events

Controller events are sent from the controller to the host computer. Their primary purpose is to keep the host informed about important actions or changes occurring on the controller, such as page switches, preset changes, knob touches, and device connections.

By default, controller events that are triggered by user actions (not initiated by SysEx API commands) are transmitted through the `Electra Controller CTRL` MIDI port. This default behavior can be changed using the Set Event Port command, allowing developers to route these user-driven event messages to a different USB device MIDI port if needed — keeping event traffic separated from other MIDI streams.

However, events triggered as a response to SysEx API commands are always sent back over the same MIDI port on which the original SysEx API command was received. This ensures that responses remain properly linked to their initiating requests, even if a custom event port has been configured.

ACK

Acknowledged. Informs the host that the last operation was successfully completed.

```
0xF0 0x00 0x21 0x45 0x7E 0x01 transaction-id-lsb transaction-id-msb 0xF7
```

`0xF0` SysEx header byte
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x7E` Controller event
`0x01` ACK (acknowledged)
`transaction-id-lsb` transaction Id LSB
`transaction-id-msb` transaction Id MSB
`0xF7` SysEx closing byte

Where:

- `transaction-id-lsb` is the least significant 7 bits (LSB) of the transaction Id
- `transaction-id-msb` is the most significant 7 bits (MSB) of the transaction Id

The `Transaction Id` is split into two 7-bit parts: a most significant byte (MSB) and a least significant byte (LSB), using the following logic:

```
transaction-id-msb = transactionId >> 7
transaction-id-lsb = transactionId & 0x7F
```


If a **Transaction Id** is included in the Command, the corresponding **ACK** or **NACK** response will also include the same two bytes, allowing you to match the response to the original command.

Example

The ACK with transaction Id 4183 should be transferred as

```
0xF0 0x00 0x21 0x45 0x7E 0x01 0x77 0x20 0xF7
```

If no Transaction ID was included in the request, a Transaction ID of 0 will be present in the ACK response.

NACK

Not acknowledged. Informs the host that the last operation did not succeed.

```
0xF0 0x00 0x21 0x45 0x7E 0x00 transaction-id-lsb transaction-id-msb 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x7E Controller event
0x00 NACK (not acknowledged)
transaction-id-lsb transaction Id LSB
transaction-id-msb transaction Id MSB
0xF7 SysEx closing byte

Where:

- **transaction-id-lsb** is the least significant 7 bits (LSB) of the transaction Id
- **transaction-id-msb** is the most significant 7 bits (MSB) of the transaction Id

The **Transaction Id** is split into two 7-bit parts: a most significant byte (MSB) and a least significant byte (LSB), using the following logic:

```
transaction-id-msb = transactionId >> 7
transaction-id-lsb = transactionId & 0x7F
```

If a **Transaction Id** is included in the Command, the corresponding **ACK** or **NACK** response will also include the same two bytes, allowing you to match the response to the original command.

Example

The NACK with transaction Id 4183 should be transferred as

```
0xF0 0x00 0x21 0x45 0x7E 0x00 0x77 0x20 0xF7
```

If no Transaction ID was included in the request, a Transaction ID of 0 will be present in the NACK response.

Preset switch

The Preset switch event informs the host that the user has changed the preset on the controller.

```
0xF0 0x00 0x21 0x45 0x7E 0x02 bank-number slot 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x7E` Controller event
- `0x02` Preset switch
- `bank-number` Current bank number (0 .. 5)
- `slot` Current preset slot (0 .. 11)
- `0xF7` SysEx closing byte

Snapshot list change

The Snapshot list change Event informs the host that the list of snapshots has been modified. It is sent whenever a snapshot is added, updated, or removed.

```
0xF0 0x00 0x21 0x45 0x7E 0x03 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x7E` Controller event
- `0x03` Snapshot list change
- `0xF7` SysEx closing byte

Capture list change

The Capture list change Event informs the host that the list of capture has been modified. It is sent whenever a snapshot is added, updated, or removed.

```
0xF0 0x00 0x21 0x45 0x7E 0x31 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x7E Controller event
0x31 Capture list change
0xF7 SysEx closing byte

Pot touch

The Pot touch event informs the host when the user touches or releases a potentiometer (knob) on the controller.

```
0xF0 0x00 0x21 0x45 0x7E 0x0A pot-id control-id-lsb control-id-msb touched 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x7E Controller event
0x0A Pot touch activity
pot-id an identifier of the pot (0 .. 11)
control-id-lsb a LSB of a controlId
control-id-msb a MSB of a controlId
touched is set to **true** for initial touch, and **false** when pot is released
0xF7 SysEx closing byte

Preset list change

The Preset list change event informs the host that the list of presets has been modified. It is sent whenever a preset is added, updated, or removed.

```
0xF0 0x00 0x21 0x45 0x7E 0x05 0xF7
```

0xF0 SysEx header byte
0x00 **0x21** **0x45** Electra One MIDI manufacturer Id
0x7E Controller event
0x05 Preset list change
0xF7 SysEx closing byte

Page switch

The Page switch event informs the host that the user has changed the active page on the controller.

```
0xF0 0x00 0x21 0x45 0x7E 0x06 page-number 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x7E** Controller event
- 0x06** Page switch
- page-id** Current page number as defined in the preset (0 .. 11)
- 0xF7** SysEx closing byte

Control Set switch

The Control Set switch event informs the host that the user has changed the active Control Set on the controller.

```
0xF0 0x00 0x21 0x45 0x7E 0x07 control-set-number 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x7E** Controller event
- 0x07** Control set switch
- control-set-number** Current active control Set (0 .. 2)
- 0xF7** SysEx closing byte

Preset bank switch

The Preset bank switch event informs the host that the user has changed the active preset bank on the controller.

```
0xF0 0x00 0x21 0x45 0x7E 0x08 preset-bank-number 0xF7
```

- 0xF0** SysEx header byte
- 0x00** **0x21** **0x45** Electra One MIDI manufacturer Id
- 0x7E** Controller event
- 0x08** preset bank switch
- preset-bank-number** Current page number (0 .. 5)

0xF7 SysEx closing byte

USB Host change notification

Informs the host that a new device was connected or an existing device was disconnected from the USB Host port.

```
0xF0 0x00 0x21 0x45 0x7E 0x08 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x7E Controller event

0x08 USB Host change

0xF7 SysEx closing byte

Snapshot change

Informs the host that the user made change regarding the snapshots. Upon receiving this event the host might want to query the snapshot list information.

```
0xF0 0x00 0x21 0x45 0x7E 0x03 0xF7
```

- `0xF0` SysEx header byte - `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id - `0x7E` Controller event -
`0x03` Snapshot change - `0xF7` SysEx closing byte

Snapshot bank switch

Informs the host that the user changed current snapshot bank.

```
0xF0 0x00 0x21 0x45 0x7E 0x04 bank-number 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x7E Controller event

0x04 Snapshot bank switch

bank-number Snapshot bank (0 .. 11)

0xF7 SysEx closing byte

Midi learn info

When Electra has the MIDI learn enabled it sends a MIDI message with description of MIDI messages received on user ports to.

```
0xF0 0x00 0x21 0x45 0x03 midilearn-json-data 0xF7
```

0xF0 SysEx header byte

0x00 **0x21** **0x45** Electra One MIDI manufacturer Id

0x03 Midi learn

midilearn-json-data a JSON data that describe detected MIDI message

0xF7 SysEx closing byte

An example of midilearn-json-data

non-SysEx:

```
{
  "port": 0,
  "msg": "cc7",
  "channel": 2,
  "parameterId": 10,
  "value": 119
}
```

SysEx:

```
{
  "port": 0,
  "msg": "sysex",
  "data": [ 67, 32, 0 ]
}
```

Log message

A log message is a text that is transmitted to the host computer in order to provide the user with information what is happening in the controller. The log messages are generated either by the firmware or user's Lua script.

```
0xF0 0x00 0x21 0x45 0x7F 0x00 log-message 0xF7
```

0xF0 SysEx header byte

`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
`0x7F` System call
`0x00` Log message
`log-message` ASCII bytes representing the log message
`0xF7` SysEx closing byte

The `log-message` is a text string that start with a number representing milliseconds from the start of the controller, followed by the space, and then the text of the message.

An example of log-message

```
147362 ElectraApp: preset successfully loaded
```