

# File Transfer SysEx API

The File Transfer SysEx API is an extension of the Electra One [SysEx Implementation](#). It adds a way for you to upload, download, list, and delete files stored inside the Electra One controller.

This API is especially useful when you need to manage files like presets, Lua scripts, performances, or snapshots directly through MIDI SysEx messages. While the general SysEx Implementation provides essential tools for basic file management, the File Transfer SysEx API extends that functionality, adding features such as:

- Transferring large files in smaller chunks
- Verifying file integrity using MD5 checks
- Managing multiple Lua script files within a preset slot

And more advanced file operations

The File Transfer API uses the same message structure as the general Electra One SysEx Implementation. If you are new to SysEx communication or the Electra One message format, we recommend starting with the [Electra One SysEx API documentation](#) first.

## Note

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

## Internal file storage

The Electra One controller uses an SD card to store presets, Lua scripts, and other files. It also stores important system resources such as fonts, graphical assets, and configuration files. However, the SD card is not used to store the firmware — the controller's firmware is stored separately in dedicated internal flash memory.

The SD card can be accessed through the slot located on the side of the controller enclosure. On older revisions of the Electra One MK2, the SD card slot is not externally accessible and is located under the plastic lid on the bottom of the controller.

## File system structure

The files are organized on the SD card in the following file structure:

```
-- assets
|
+- boot
|
+- cache
|
+- ctrlv2 -- captures
    |
    +- lua
    |
    +- presets
    |
    +- slots -- b00 -- p00
```

```

|      |      |
|      |      +- p01
|      |      |
|      |      +- ...
|      |      |
|      +- b01
|      |
|      +- ...
|
+- snaps
|
+ configv4.cfg

```

## assets

System folder for fonts and graphical data files.

## boot

System folder used for applying firmware and bootloader updates.

## cache

System folder for storing file transfers that are in progress.

## ctrlv2

Folder that holds all the files needed to run the MIDI controller firmware.

### ctrlv2/captures

Folder for saving captures, collections of MIDI messages. Capture data files are organized into subfolders based on project ID (projectId).

### ctrlv2/lua

Folder for storing preloaded Lua modules. It contains `namespace` subfolders, typically named after the developer's nickname. Developers are responsible for managing their own folders. Lua modules stored here can be imported in preset Lua scripts using Lua's `require` function.

Developers who would like their Lua modules to be included in official Electra One releases and distributed with new controllers - should contact Electra One support.

### ctrlv2/presets

Folder for storing preloaded presets. Similar to the Lua folder, it uses `namespace` subfolders based on developer nicknames. Preloaded presets can be used to provision preset slots using the Load Preloaded Preset SysEx command.

Developers who would like their preloaded presets to be included in official Electra One releases and distributed with new controllers - should contact Electra One support.

### ctrlv2/slots

Folder for storing the 72 available preset slots (6 banks of 12 presets each). Files are organized into:

- `bnn` folders (one for each bank)
- `pnn` folders (one for each preset slot within a bank)

Each pnn preset slot folder can contain:

- `preset.json` — the preset's main JSON file
- `main.lua` — the Lua script that runs when the preset is loaded
- `devices.json` — device override definitions
- `data.json` — persisted Lua table data
- `performance.json` — performance configuration

### ctrlv2/snaps

Folder for storing snapshots, saved values of preset controls. Snapshot files are organized into subfolders based on project ID (projectId).

There are other system folders and files on the SD card, but the ones listed above are the most important for developers to know about and use.

## USB Disk mode

The files on the SD card can also be accessed using the USB Mode in the controller's bootloader. When enabled, the controller appears as an external disk drive on your computer. However, it is important to keep in mind that the controller is not a full-featured USB mass storage device — operations may take longer than usual, and formatting the SD card is not possible through Disk Mode.

## Protocol Handshake

The File Transfer protocol is built around the idea of a transfer cache — a temporary location where files are transferred in chunks, assembled, and then moved to their final destination.

The protocol supports transferring multiple files at once and distributing them to their final locations as a single atomic transaction, meaning either all files are successfully transferred together, or none are.

The general workflow is as follows:

1. Open a file transfer cache transaction.
2. Register the individual files that will be transferred in the transaction.
3. Transfer the file chunks for all registered files.
4. Commit the transaction, verifying file integrity and copying the completed files to their final locations.

All commands use the standard ACK/NACK handshake to confirm success or failure. In addition, during chunk transfers, the controller reports progress information back to the host.

### Open the cache

Starts a new file transfer transaction. Any previously transferred data or unfinished transactions are cleared, and a new transaction is initialized.

```
0xF0 0x00 0x21 0x45 0x01 0x2D 0xF7
```

`0xF0` SysEx header byte  
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id  
`0x01` Upload command  
`0x2D` Staged cache  
`0xF7` SysEx closing byte

## Register the files

Registers individual files for transfer. Each file is identified by a numeric `fileId` (0 .. 127). For each file, its total size must also be provided. The size is split into four 7-bit bytes (following the standard MIDI SysEx 7-bit encoding).

If multiple files are to be transferred in a single transaction, this command must be sent separately for each file, using a unique `fileId` and providing the correct size for each file.

```
0xF0 0x00 0x21 0x45 0x01 0x2E file-id size-0 size-1 size-2 size-3 F7
```

`0xF0` SysEx header byte  
`0x00` `0x21` `0x45` Electra One MIDI manufacturer Id  
`0x01` Upload command  
`0x2E` Staged cache header  
`file-id` Numeric fileId (0 .. 127)  
`size-0` Least significant 7 bits of the file size  
`size-1` Bits 7–13 of the file size  
`size-2` Bits 14–20 of the file size  
`size-4` Most significant bits 21–27 of the file size  
`0xF7` SysEx closing byte

The file `size` must be split into four 7-bit parts using the following logic:

```
size-0 = fileSize & 0x7F
size-1 = (fileSize >> 7) & 0x7F
size-2 = (fileSize >> 14) & 0x7F
size-3 = (fileSize >> 21) & 0x7F
```

## Transfer chunks

Transfers a data chunk for a previously registered file. Each chunk is associated with a file using its `fileId`. The chunk can carry 7-bit encoded data of arbitrary length.

After a chunk is transferred, its length is subtracted from the total remaining size of all files registered in the transaction. The controller continuously reports the total amount of data transferred so far using the Transfer Report event message.

```
0xF0 0x00 0x21 0x45 0x01 0x2F file-id 7-bit-data 0xF7
```

**0xF0** SysEx header byte  
**0x00** **0x21** **0x45** Electra One MIDI manufacturer Id  
**0x01** Upload command  
**0x2F** Staged cache chunk  
**file-id** Numeric fileId (0 .. 127)  
**7-bit-data** 7-bit data of arbitrary length  
**0xF7** SysEx closing byte

## Commit the transaction

After all chunks are successfully transferred, the Commit command must be called to finalize the transfer. This command distributes the files to their final locations.

Before moving the files, the controller verifies the integrity of each file using its provided MD5 checksum.

The instructions for committing the transaction are included in a JSON document attached to the command. This JSON document associates each fileId with its target type, destination location, and corresponding MD5 digest.

```
0xF0 0x00 0x21 0x45 0x04 0x2D file-commit-json 0xF7
```

**0xF0** SysEx header byte  
**0x00** **0x21** **0x45** Electra One MIDI manufacturer Id  
**0x04** Update command  
**0x2D** Staged cache  
**file-commit-json** JSON describing the file distribution  
**0xF7** SysEx closing byte

## An example of the file-commit-json

```
{
  "files":[
    {
      "id":1,
      "location":"slots",
      "type":"luaModule",
      "path":"test",
      "bankNumber":0,
```

```

        "slot":0,
        "md5":"1c40e4876067a51b9ed5ee73b7a32f09"
    }
]
}

```

## files

The `files` element is an array containing the individual files to be committed.

## id

The `id` field is the file identifier that was registered using the Register command.

## location

The `location` field is an enum that specifies where the file will be moved after the transfer is completed. Files can only be moved to predefined locations, which are:

- `slots` - preset slots
- `updates` - boot folder for firmware and bootloader updates
- `assets` - graphical assets and system files
- `modules` - preloaded Lua modules
- `presets` - preloaded presets
- `root` - firmware root (used for configuration files)

Some locations may require additional parameters to be provided:

## slots

The slots location requires `bankNumber` and `slot` to be provided. For additional Lua source files, `path` must be provided.

- `bankNumber` - a numeric id of the preset bank (0 .. 5)
- `slot` - a numeric id of the slot within the bank (0 .. 11)
- `path` - file name of the Lua file, excluding the .lua extension. path is a plain filename and cannot include subfolders.

## modules

The preloaded Lua modules require `namespace` and `path` to be provided.

- `namespace` - name of the collection of Lua modules, usually a nickname of the developer
- `path` - name of the Lua file, excluding the .lua extension, path is a plain filename and cannot include subfolders.

## presets

The presloaded presets require `namespace` and `path` to be provided.

- `namespace` - name of the collection of preloaded presets, usually a nickname of the developer
- `path` - name of the preloaded preset, excluding the .json extension, path is a plain filename and cannot include subfolders.

## type

Specifies the type of file to be saved to the location. Only predefined file types are allowed, which are:

- `firmware` — A firmware file to be uploaded to the internal flash
- `bootloader` — A bootloader firmware file to be uploaded to the internal flash
- `preset` — A preset file to be uploaded to a slot or to preloaded presets
- `lua` — A Lua script file to be executed
- `luaModule` — A Lua module
- `ui` — A graphical assets file
- `config` — A configuration file
- `deviceList` — A Device Overrides definition
- `datafile` — A persisted Lua table data file
- `performance` — A performance file

## MD5

MD5 digest used to verify the integrity of the transferred file. If the MD5 digest calculated over the transferred file differs from the MD5 digest provided in the commit JSON, the entire distribution process will be canceled. In that case, the transferred files will remain stored in the transfer cache.

# Querying data from the controller

This section explains how to query file information from various storage locations. The File Transfer SysEx API uses a JSON document to describe each query.

If you only need to work with files stored in preset slots, you can use the simpler [Preset Slot Information](#) query described in the SysEx Implementation document.

## Get Location files

A request to fetch a list of files stored in a specific location. The location to be queried is specified using a JSON document. Some locations may require additional parameters, following the same rules described earlier in the [Commit the transaction](#) section.

### Request

```
0xF0 0x00 0x21 0x45 0x02 0x34 location-query-json-data 0xF7
```

`0xF0` SysEx header byte

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

`0x02` Query data

`0x34` Location

`location-query-json-data` JSON document specifying the location to be queried

`0xF7` SysEx closing byte

## An example of location-query-json-data

```
{
  "location":"slots",
  "bankNumber":0,
  "slot":0
}
```

## Response

```
0xF0 0x00 0x21 0x45 0x01 0x34 location-json-data 0xF7
```

0xF0 SysEx header byte  
 0x00 0x21 0x45 Electra One MIDI manufacturer Id  
 0x01 Data dump  
 0x34 Location  
 location-json-data JSON document with a list of files stored in location  
 0xF7 SysEx closing byte

## An example of location-json-data

```
{
  "version":1,
  "path":"/ctrlv2/slots/b00/p00",
  "exists":true,
  "files":[
    {
      "name":"preset.json",
      "md5":"b58f9ee9391b7e49f471fcbb2deb536c"
    },
    {
      "name":"main.lua",
      "md5":"7f00373c5818f254ef19a82217a18be0"
    },
    {
      "name":"devices.json",
      "md5":"5dec6bf7eebb098dda3d706fe6c2f115"
    }
  ]
}
```



# Remove files

The File Transfer SysEx API provides a command to clear the contents of a specified location. The location where files should be removed is specified using a JSON document. If you only need to clear files from preset slots, you can use the simpler [Clear preset slot](#) command described in the SysEx Implementation document.

## Remove files from location

The Remove Files command deletes all files stored in a specific location. The location is specified using the same JSON document format as the [Get Location files](#) query.

```
0xF0 0x00 0x21 0x45 0x05 0x34 location-query-json-data 0xF7
```

**0xF0** SysEx header byte

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

**0x05** Remove command

**0x34** Location

**location-query-json-data** bytes representing ASCII bytes of the preset file

**0xF7** SysEx closing byte

### An example of location-query-json-data

```
{
  "location": "modules",
  "namespace": "xot",
  "path": "ableton"
}
```

# Controller events

While transferring file chunks, the controller reports the amount of data transferred so far. Developers can use this information, together with the total size provided during the initial file registration, to track the overall transfer progress.

By default, controller events 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.

## Report progress

The Report Progress event informs the host about the amount of data that has been transferred so far within the current file transfer transaction.

```
0xF0 0x00 0x21 0x45 0x7E 0x2D size-0 size-1 size-2 size-3 0xF7
```

- `0xF0` SysEx header byte
- `0x00` `0x21` `0x45` Electra One MIDI manufacturer Id
- `0x7E` Controller event
- `0x2D` Staged cache
- `size-0` Least significant 7 bits of the file size
- `size-1` Bits 7–13 of the file size
- `size-2` Bits 14–20 of the file size
- `size-4` Most significant bits 21–27 of the file size
- `0xF7` SysEx closing byte

To reconstruct the size of data from the four 7-bit size parts received (size-0, size-1, size-2, and size-3), use the following logic:

```
size = size-0 + (size-1 << 7) + (size-2 << 14) + (size-3 << 21)
```

## Example

The Progress report with size of 138 bytes:

```
0xF0 0x00 0x21 0x45 0x7E 0x2D 0x10 0x01 0x00 0x00 0xF7
```