CV40 API

The programmer guide

Version 1.3.0, 21/07/2025

Table of Contents

Chapter 1. Introduction

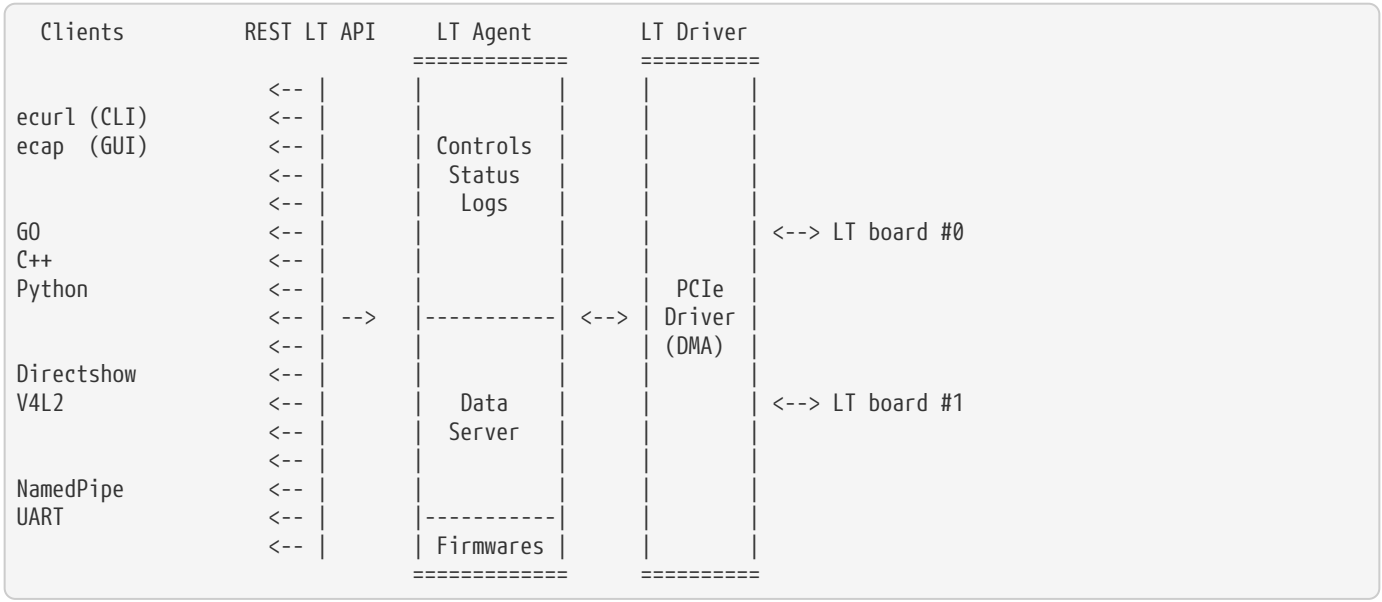
The **LT API** is built around the REST architecture (Representational State Transfer). REST defines a structured approach for exposing system functionalities via a consistent interface.

A REST API is typically accessed using predefined URLs, which represent various resources returned as JSON objects. These resources support standard methods such as `GET`, `POST`, and `DELETE`.

The **LT API** requests are processed by a host service called **cv40agent**, which utilizes standard OS mechanisms such as IPC, SG-DMA, and shared memory. This ensures minimal latency when handling API requests. To optimize performance, video and audio data are shared among consumers using memory segments, allowing large data buffers and concurrent access.

With **LT boards** designed to be seen as *hardware-as-a-service* approach, the **LT API** abstracts hardware-specific implementations behind a unified API. This allows users to focus on core functionalities such as capturing, playing, recording, and streaming audio/video data.

The **LT API** is accessible through various clients, including the **ecurl** command line interface (CLI) and the **ecam** graphical user interface (GUI). The API can also be accessed directly through C++, Python, DirectShow, V4L2, NamedPipe, and UART.



Chapter 2. Installation

2.1. Windows

Download and run the latest **cv40install_x.x.x.exe** to install the **cv40 family** drivers, tools and services. If necessary the previous version will be uninstalled.

Once installed, the **cv40agent** service will be started automatically and will be (re)started automatically with each system (re)boot.

The **cv40agent** can be controlled using the Windows Services Manager or the command line. For more details, see [control_service].

The **LT boards** plugged into the host should appear in the Windows Device Manager under the **Sound, video and game controllers**.

Directshow

All the boards inputs are accessible through directshow filters, and then are available in any directshow compatible application. The directshow filters are managed by the **cv40agent** service, so you can use them only if the service is running.

NOTE To uninstall the **cv40 family** drivers, tools and services, run the installer and choose "Remove" or use "Apps & features" menu.

2.2. Linux

Download and extract the latest **cv40install_x.x.x.tar.gz**, then run the **cv40_install.sh** script to install the **cv40 family** drivers, tools and services. If necessary, the previous version will be uninstalled.

Upon successful installation, the **cv40agent** daemon will start automatically and will also (re)started with each system (re)boot. If the installation fails, contact us and check the generated **cv40install.log** file for details.

The **cv40agent** can be controlled using the command line. For more details, see [control_service].

V4L2

All the boards inputs are accessible through V4L2 drivers, and then are available in any V4L2/GStreamer compatible application. The V4L2 drivers are managed by the **cv40agent** service, so you can use them only if the

```
service is running.
```

NOTE	To uninstall the **cv40 family** drivers, tools and services, run cv40_uninstall.sh from the installation directory.

2.3. SDK

Download the latest **cv40sdk_x.x.x.zip** or **cv40sdk_x.x.x.tar.gz** archive and extract it anywhere you want. The SDK contains the API documentation, the API libraries and examples for the following languages **Go**, **C++** and **Python**.

Please navigate through the examples to learn how to program the API.

You can also create scripts using the **ecurl** command-line tool. For more details, see [ecurl].

2.4. Tools

Two tools are installed along with the **cv40agent** service:

- **ecurl** - a command-line tool to send REST API requests to the **cv40agent**.
- **ecam** - a graphical user interface to control and/or test LT boards.

By default, the service and tools are added to the **PATH** environment variable, allowing you to use them from any command line.

2.5. cv40agent Controls

To access the **LT API**, ensure that the **cv40agent** is installed and running on your host system.

You can use the following commands, which require administrative privileges, to control the **cv40agent**.

Start the cv40agent

```
$ cv40agent start
```

Stop the cv40agent

```
$ cv40agent stop
```

Check the cv40agent and boards firmware version

```
$ cv40agent version
```

Check the cv40agent status

```
$ cv40agent status
```

2.6. Board Firmware

If the boards installed in the host require a firmware update, the **cv40agent** service will automatically update them when the service starts. This process may take up to 2 minutes, depending on the board type. The service will be unavailable until the update is completed.

Please use the command below if you want to check the firmware version of the boards.

Check the cv40agent and boards firmware version

```
$ cv40agent version
```

It is also possible to manually update the board firmware.

Update the boards firmware

```
$ cv40agent update
```

NOTE	To update the board firmware, the **cv40agent** must be stopped.

Chapter 3. ecurl (CLI)

The **ecurl** program is a developer tool to help you make requests on the **LT API** directly from your terminal. The tool is deployed along with the **cv40agent** at the installation stage. The tool has been developed with our SDK and is available for both Linux and Windows platforms.

You can use the **ecurl** CLI to:

- Create, retrieve, update or delete **LT API** objects.
- Play and record any video or audio resources.
- Use the multi-channel feature of the **LT boards**.
- Control and test the installed **LT boards**.

NOTE	The **cv40agent** must be running otherwise **ecurl** will not work.

3.1. GET command

```
$ ecurl get <url>
```

Send a GET request to retrieve a specific API object identified by the <**url**>.

▼ **Examples**

Retrieve information about the cv40 family agent

```
$ ecurl get cv40:/
```

Retrieve informaton from cv40 board located at index 0

```
$ ecurl get cv40:/0
```

Make a JPEG capture

```
$ ecurl get cv40:/0/{in}/0/file -d type=image/jpeg
```

3.2. POST command

```
$ ecurl post <url> [-d @file.json] [-d field=value] [-d data=@file.bin]
```

Create or modify the resource designated by the <**url**>.

Arguments may be added to the request with the **-d** optional flags. It is possible to use a file content as input by preceding the filename with the @ character. By preceding the filename with the **$** charater, relative path will be translated to the absolute full path.

▼ **Examples**

Create a virtual video input from a mp4 file

```
$ ecurl post cv40:/canvas/0/init -d source=$video.mp4
```

Load a custom HDMI-IN Edid

```
$ ecurl post cv40:/0/hdmi-in/0/edid -d data=@custom.edid
```

3.3. DELETE Command

```
$ ecurl delete <url>
```

Delete or reset the resource pointed by the <**url**>.

▼ **Examples**

Unplug virtual video input

```
$ ecurl delete cv40:/canvas/0
```

3.4. PLAY Command

```
$ ecurl play <url> [-d]
```

Play video or audio source until `Ctrl` + `c` is pressed.
Arguments may be added to the request with the **-d** optional flags.

▼ **Examples**

Live display of camera video

```
$ ecurl play cv40:/0/camera/0/data -d media=video/yuyv
```

Live playing of camera video

```
$ ecurl play cv40:/0/camera/0
```

3.5. REC Command

```
$ ecurl rec <url> [-d]
```

Record video or audio source until `Ctrl` + `c` is pressed.
Arguments may be added to the request with the **-d** optional flags.

▼ **Examples**

Record camera video into a mp4 file

```
$ ecurl rec cv40:/0/camera/0/file -d media=video/mp4
```

Record camera video using NVENC hardware encoder and hevc codec

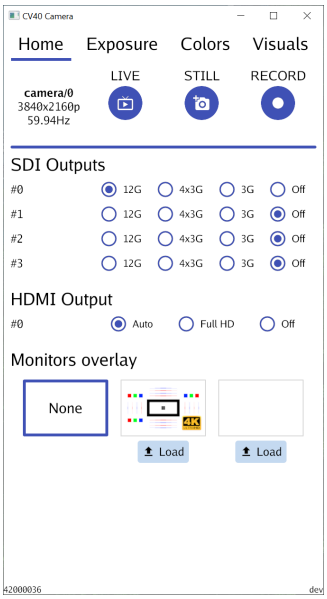
```
$ ecurl rec cv40:/0/camera/0 -d media=video/mp4 -d extra.hw=nvenc -d extra.codec=hevc
```

Chapter 4. ecam (GUI)

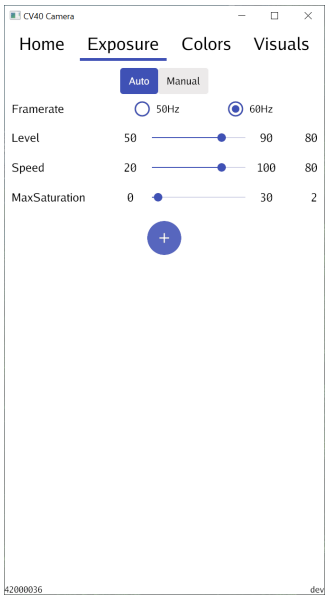
The **ecam** program is a simple graphical user interface developed with our SDK and is available for Linux and Windows platforms.

You can use the **ecam** application to:

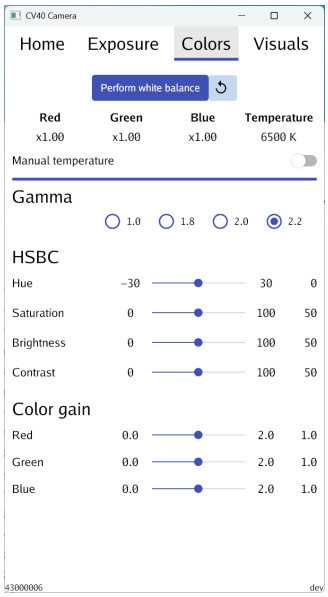
- Play and record the camera input. You can also control HDMI and SDI output configuration and test the overlay feature.

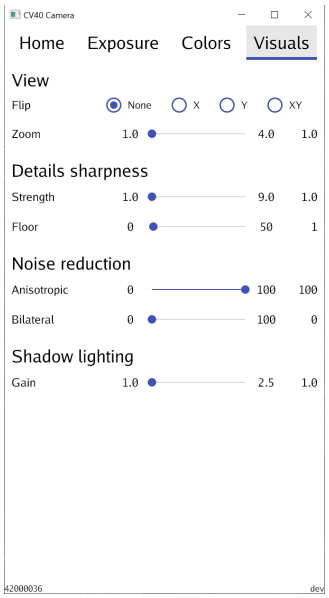


- Switch between automatic and manual exposure, and adjust the exposure settings.



- Adjust the colors settings (HSBC and gamma correction), perform white balance or modify the color temperature.
- Control the visuals settings like edge enhancement, noise reduction.





NOTE The **cv40agent** has to be running otherwise **ecam** will not work.

Chapter 5. API description

An API endpoint is a URL where the API processes requests for a specific resource. Endpoints are accessed via URLs using the syntax `scheme:/path`. It consists of:

- A non-empty scheme component followed by a colon `cv40:`.
- A path component made up of path segments separated by slashes `/`.

For clarity, the URLs endpoints are presented in tables where:

- The **scheme `cv40:` is omitted**.
- **Tables vertical separators replace slashes `/`.**
- Each method listed in a cell represents an available operation for that path.
- An **empty cell** means that no method exists for that path.

5.1. Agent

The **agent** endpoint allows to retrieve the **cv40agent** software version.

/
GET

```
GET    /
```

5.1.1. Agent Configuration File

The agent can be configured using a optional configuration file. You can place the `cv40agent.config` file in the same directory as the application executable. Configuration options:

Option	Description
`WD`	Working directory for the agent (default: current working directory)
`numCanvases`	Number of canvas objects to create (default 4)
`noVideoSignal`	Settings for when no video signal is detected (default: gray background with "NO SIGNAL" text)
`dshowFilters`	Enable/disable DirectShow filters (default: true)
`v4l2Filters`	Enable/disable V4l2 filters (default: false)
`devMode`	Enable/disable development mode (default: false)

This file is included in the agent folder. The configuration file is optional and can be omitted if not needed.

5.1.2. Agent Object

revision `string`

Commit hash.

time `string`

Commit timestamp.

version `string`

Commit tagged version.

Agent object

```
{
    "revision": "...hash...",
    "time": "...time...",
    "version": "1.3.0"
}
```

5.1.3. View cv40agent Information

Retrieves the **cv40agent** software version.

Parameters

None.

Response

Returns the agent object if the request succeeded.

GET /

request

```
{
    "method": "GET",
    "url": "cv40:/",
    "body": null
}
```

response

```
{
    "revision": "...hash...",
    "time": "...time...",
    "version": "1.3.0"
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/
```

▼ *GO*

```
var response lt.Agent // struct to store the response
err := lt.Get("cv40:/", &response)
```

▼ *C++*

```
lt::Agent response; // struct to store the response
lt::error err = lt::Get("cv40:/", response);
```

▼ *Python*

```
response, err = Get("cv40:/")
```

5.2. Board

The **board** endpoint allows to retrieve information on boards installed into the host. These information are described in the board object section.

`:board`	`buttons`	`:pin`
`GET`	`GET`	`GET`

`GET` `/:board`

`GET` `/:board/buttons/:pin`

`board`
: Device position into the host `[0 .. 1]`.

`pin`
: Button position `[0 .. 4]`.

5.2.1. Board Object

model `string`

Board model identifier. If no devices is found, the value is left empty.

sn `int`

Board serial number.

cpu `uint`

Board CPU datecode.

fpga `uint`

Board FPGA datecode.

bridge `uint`

Board bridge datecode.

Board object

```
{
    "model" : "cv42",
    "sn" : 64000000,
    "cpu" : 0,
    "fpga" : 0,
    "bridge" : 0
}
```

5.2.2. View Board Information

To retrieve the board information at a given position, send a GET request to the **/:board** endpoint.

Parameters

None.

Response

Returns the board object if the request succeeded.

GET /:board

request

```
{
    "method": "GET",
    "url": "cv40:/0",
    "body": null
}
```

response

```
{
    "model" : "cv42",
    "sn" : 64000000,
    "cpu" : 0,
    "fpga" : 0,
    "bridge" : 0
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0
```

▼ *GO*

```
var response lt.Board // struct to store the response
err := lt.Get("cv40:/0", &response)
```

▼ *C++*

```
lt::Board response; // object to store the response
lt::error err = lt::Get("cv40:/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0")
```

5.2.3. Button Object

The **button** object allows to capture the button pin state.

description `string`

Description of the button pin.

pressed `bool`

Is the button currently 'pressed'.

pressedCount `int`

Number of times the button has been pressed since the last 'release' event.

timestamp `int64`

The unix timestamp of the last 'press' or 'release' event in microseconds.

Button object

```
{
    "description": "0/buttons/0",
    "pressed": false,
    "pressedCount": 0,
    "timestamp": 0
}
```

5.2.4. View Button State

To retrieve the button status, send a GET request to the **button/:pin** endpoint.

Parameters

None.

Response

Returns the button object if the request succeeded.

GET /0/buttons/:pin

request

```
{
    "method": "GET",
    "url": "cv40:/0/buttons/0",
    "body": null
}
```

response

```
{
    "description": "0/buttons/0",
    "pressed": false,
    "pressedCount": 0,
    "timestamp": 0
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/buttons/0
```

▼ *golang*

```
var response lt.Button // struct to store the response
err := lt.Get("cv40:/0/buttons/0", &response)
```

▼ *C++*

```
lt::Button response; // struct to store the response
lt::error err = lt::Get("cv40:/0/buttons/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/buttons/0")
```

5.2.5. Buttons Object

The **buttons** object is a collection of button objects. It allows to capture the state of multiple buttons at once.

5.2.6. View Buttons State

To retrieve the status of all buttons at once, send a GET request to the **buttons** endpoint.

Parameters

None.

Response

Returns the buttons object if the request succeeded.

GET /0/buttons

request

```
{
    "method": "GET",
    "url": "cv40:/0/buttons",
    "body": null
}
```

response

```
{
    {
        "description": "0/buttons/0",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/buttons/1",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/buttons/2",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/buttons/3",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/buttons
```

▼ *golang*

```
var response lt.Buttons // struct to store the response
err := lt.Get("cv40:/0/buttons", &response)
```

▼ *C++*

```
lt::Buttons response; // struct to store the response
lt::error err = lt::Get("cv40:/0/buttons", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/buttons/0")
```

5.3. Camera

This endpoint describes how to use **camera** inputs.
Native data can be accessed via the **format** path enumerator `yuyv` (video).
Some of the proposed **format** might require to use the host CPU and/or GPU before being delivered.

The `pci` endpoint allows to limit the maximum **framerate** coming through the PCIe bus to save bandwidth and ensure best quality of service for **multi-channel** scenarios.

Some camera has GPIOs buttons that can be used to trigger a capture, a recording or zoom control.

`:board` `GET`	`camera`	`:id` `GET`	`data` `POST`	
			`file` `POST`	
			`net` `POST`	
			`white` `GET` `POST`	
			`colors` `GET` `POST`	
			`exposure` `GET` `POST`	
			`visuals` `GET` `POST`	
			`buttons` `GET`	`:pin` `GET`

```
GET     /:board/camera/:id
POST    /:board/camera/:id/data
POST    /:board/camera/:id/file
POST    /:board/camera/:id/net
GET     /:board/camera/:id/white
POST    /:board/camera/:id/white
GET     /:board/camera/:id/colors
POST    /:board/camera/:id/colors
GET     /:board/camera/:id/exposure
POST    /:board/camera/:id/exposure
GET     /:board/camera/:id/visuals
POST    /:board/camera/:id/visuals
GET     /:board/camera/:id/buttons/:pin
```

board
: Device position into the host [0 .. 1].

id
: camera index number [0 .. 1].

pin
: Button position [0 .. 4].

5.3.1. Camera Object

model `string`

Camera head model. If no devices is found, the value is left empty.

sn `int`

Camera head serial number.

cpu `int`

Camera head CPU datecode.

fpga `int`

Camera head FPGA datecode.

---

audio `json`

Audio signal object.

→ **description** `string`

A short description of the audio signal.

→ **format** `string`

The audio sample format `pcm`.

→ **channels** `int`

The number of audio channels.

→ **samplerate** `int`

The number of audio samples per second.

→ **depth** `int`

The number of bits per audio sample.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

Camera object

```
{
    "cpu": 0,
    "fpga": 0,
    "model": "",
    "sn": 0,
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

video `json`

Video signal object.

→ **description** `string`

The video signal short description.

→ **format** `string`

The pixel color format `rgb444`, `yuv444` or `yuv422`.

→ **framerate** `float`

The number of video frames per second.

→ **size** `[2]int`

The video frame width and height in pixel units.

→ **interlaced** `bool`

The video frame interlaced status.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

5.3.2. View Camera Status

To retrieve the camera signal status, send a GET request to the **camera/:id** endpoint

Parameters

None.

Response

Returns the camera object if the request succeeded.

GET /:board/camera/:id

request

```
{
    "method": "GET",
    "url": "cv40:/0/buttons/0",
    "body": null
}
```

response

```
{
    "cpu": 0,
    "fpga": 0,
    "model": "",
    "sn": 0,
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/buttons/0
```

▼ *GO*

```
var response lt.Camera // struct to store the response
err := lt.Get("cv40:/0/buttons/0", &response)
```

▼ *C++*

```
lt::Camera response; // struct to store the response
lt::error err = lt::Get("cv40:/0/buttons/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/buttons/0")
```

5.3.3. White Object

The **white** object is used to control the white balance and color temperature of the camera.

balance `[3]float64`

White balance red, green and blue gains `[0.0 .. 2.0]`.

temperature `int`

Color temperature in Kelvin `[3500 .. 7500]`.

White Object

```
{
    "balance": [1.0, 1.0, 1.0],
    "temperature": 6500,
}
```

5.3.4. View White Settings

To retrieve the camera white parameters, send a GET request to the **white** endpoint.

Parameters

None.

Response

Returns the white object if the request succeeded.

GET /:board/camera/:id/white

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/white",
    "body": null
}
```

response

```
{
    "balance": [1.0, 1.0, 1.0],
    "temperature": 6500,
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/white
```

▼ *GO*

```
var response lt.CameraWhite // struct to store the response
err := lt.Get("cv40:/0/camera/0/white", &response)
```

▼ *C++*

```
lt::CameraWhite response;
lt::error err = lt::Get("cv40:/0/camera/0/white", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/white")
```

5.3.5. Update White Settings

To update the white settings, send a POST request with the parameters you want to update to the **white** endpoint.
If you want to perform a white balance operation, send an empty body. In that case, the camera will automatically adjust the white balance gains and temperature.

Parameters

None to perform a white balance operation.
Otherwise, include parameters of the white object you want to update.
In the example, the `temperature` is set to `3800`.

Response

Returns the white object if the request succeeded.

POST /:board/camera/:id/white

request

```
{
    "method": "POST",
    "url": "cv40:/0/camera/0/white",
    "body": {
        "temperature": 3800
    }
}
```

response

```
{
    "balance": [1.0, 1.0, 1.0],
    "temperature": 3800,
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/camera/0/white -d temperature=3800
```

▼ *GO*

```
body := lt.JSON{
    temperature: 3800,
}
var response lt.CameraWhite // struct to store the response
err := lt.Post("cv40:/0/camera/0/white", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"temperature", 3800}
};
lt::CameraWhite response;
lt::error err = lt::Post("cv40:/0/camera/0/white", body, response);
```

▼ *Python*

```
body = {
    "temperature": 3800
}
response, err = Post("cv40:/0/camera/0/white", body)
```

5.3.6. Colors Object

The **colors** object is used to customize the color saturation, hue, brightness, contrast and gamma correction.

hue `int`

Color hue `[-30 .. 30]`.

saturation `int`

Color saturation `[0 .. 100]`.

brightness `int`

Color brightness `[0 .. 100]`.

contrast `int`

Color contrast `[0 .. 100]`.

gamma `float64`

Color contrast `[1.0 .. 2.4]`.

colorGain `[3]float64`

Color gain red, green and blue `[0.0 .. 2.0]`.

Colors Object

```
{
    "hue": 0,
    "saturation": 50,
    "brightness": 50,
    "contrast": 50,
    "gamma": 2.2,
    "colorGain": [1.0, 1.0, 1.0]
}
```

5.3.7. View Colors Settings

To retrieve the camera colors parameters, send a GET request to the **colors** endpoint.

Parameters

None.

Response

Returns the colors object if the request succeeded.

GET /:board/camera/:id/colors

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/colors",
    "body": null
}
```

response

```
{
    "hue": 0,
    "saturation": 50,
    "brightness": 50,
    "contrast": 50,
    "gamma": 2.2,
    "colorGain": [1.0, 1.0, 1.0]
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/colors
```

▼ *GO*

```
var response lt.CameraColors // struct to store the response
err := lt.Get("cv40:/0/camera/0/colors", &response)
```

▼ *C++*

```
lt::CameraColors response;
lt::error err = lt::Get("cv40:/0/camera/0/colors", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/colors")
```

5.3.8. Update Colors Settings

To update the colors settings, send a POST request with the parameters you want to update to the **colors** endpoint.

Parameters

Include parameters of the colors object you want to update.
In the example, the `saturation` is set to `80` and the `brightness` to `60`.

Response

Returns the colors object if the request succeeded.

POST /:board/camera/:id/colors

request

```
{
    "method": "POST",
    "url": "cv40:/0/camera/0/colors",
    "body": {
        "brightness": 60,
        "saturation": 80
    }
}
```

response

```
{
    "hue": 0,
    "saturation": 80,
    "brightness": 60,
    "contrast": 50,
    "gamma": 2.2,
    "colorGain": [1.0, 1.0, 1.0]
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/camera/0/colors -d saturation=80 -d brightness=60
```

▼ *GO*

```
body := lt.JSON{
    Saturation: 80,
    Brightness: 60,
}
var response lt.CameraColors // struct to store the response
err := lt.Post("cv40:/0/camera/0/colors", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"saturation", 80},
    {"brightness", 60}
};
lt::CameraColors response;
lt::error err = lt::Post("cv40:/0/camera/0/colors", body, response);
```

▼ *Python*

```
body = {
    "saturation": 80,
    "brightness": 60
}
response, err = Post("cv40:/0/camera/0/colors", body)
```

5.3.9. Exposure Object

The **exposure** object is used to control the manual and auto exposure settings of the camera.

framerate `string`

Camera framerate `50` or `60`.

---

shutter `float64`

Sensor shutter gain `[1.0 .. 240.0]`.

gain `float64`

Sensor analog gain `[1.0 .. 22.5]`.

binning `float64`

Sensor binning gain `[1.0 .. 8.0]`.

lowLightGain `float64`

Artificially increase the luminosity level of the camera with a non linear gain on darker pixels `[1.0 .. 2.45]`.

---

isAuto `bool`

Enable/disable auto exposure.

level `float64`

Luminance target when auto exposure is enabled `[50 .. 90]`.

speed `float64`

Auto exposure convergence speed `[20 .. 100]`.

maxSaturation `float64`

Auto exposure maximum saturated pixel tolerance `[0 .. 30]` percent.

window `[4]int`

Auto exposure window position and size: **x**, **y**, **width**, **height**.

Exposure object


```
{
    "framerate": 50,
    "shutter": 153.777777,
    "gain": 1,
    "binning": 0,
    "lowLightGain": 0,
    "isAuto": true,
    "level": 80,
    "speed": 80,
    "maxSaturation": 5,
    "window": [0, 72, 3840, 2048],
    "shutterLimits": [1, 240.0],
    "gainLimits": [1, 22.5],
    "binningLimits": [1, 1],
    "lowLightGainLimits": [1, 1],
}
```

shutterLimits `[2]float64`

Sensor shutter auto exposure minimum and maximum limits.

gainLimits `[2]float64`

Sensor analog gain auto exposure minimum and maximum limits.

binningLimits `[2]float64`

Sensor binning gain auto exposure minimum and maximum limits.

lowLightGainLimits `[2]float64`

Low light gain auto exposure minimum and maximum limits.

5.3.10. View Exposure Settings

To retrieve the camera exposure parameters, send a GET request to the **exposure** endpoint.

Parameters

None.

Response

Returns the exposure object if the request succeeded.

GET /:board/camera/:id/exposure

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/exposure",
    "body": null
}
```

response

```
{
    "framerate": 50,
    "shutter": 153.777777,
    "gain": 1,
    "binning": 0,
    "lowLightGain": 0,
    "isAuto": true,
    "level": 80,
    "speed": 80,
    "maxSaturation": 5,
    "window": [0, 72, 3840, 2048],
    "shutterLimits": [1, 240.0],
    "gainLimits": [1, 22.5],
    "binningLimits": [1, 1],
    "lowLightGainLimits": [1, 1],
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/exposure
```

▼ *GO*

```
var response lt.CameraExposure // struct to store the response
err := lt.Get("cv40:/0/camera/0/exposure", &response)
```

▼ *C++*

```
lt::CameraExposure response;
lt::error err = lt::Get("cv40:/0/camera/0/exposure", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/exposure")
```

5.3.11. Update Exposure Settings

Change the exposure parameters as required by your application by sending a POST request to the **exposure** endpoint.

Parameters

Include settings of the exposure object you want to update.
In the example, the `gainLimits` is set to `[2.5, 15]`.

Response

Returns the exposure object if the request succeeded.

POST /:board/camera/:id/exposure

request

```
{
    "method": "POST",
    "url": "cv40:/0/camera/0/exposure",
    "body": {
        "gainLimits": [2.5, 15.0]
    }
}
```

response

```
{
    "framerate": 50,
    "shutter": 153.777777,
    "gain": 1,
    "binning": 0,
    "lowLightGain": 0,
    "isAuto": true,
    "level": 80,
    "speed": 80,
    "maxSaturation": 5,
    "window": [0, 72, 3840, 2048],
    "shutterLimits": [1, 240.0],
    "gainLimits": [2.5, 15],
    "binningLimits": [1, 1],
    "lowLightGainLimits": [1, 1],
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/camera/0/exposure -d gainLimits=2.5,15.0
```

▼ *GO*

```
body := lt.JSON{
    GainLimits: [2.5, 15.0],
}
var response lt.CameraExposure // struct to store the response
err := lt.Post("cv40:/0/camera/0/exposure", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"gainLimits", {2.5, 15.0}}
};
lt::CameraExposure response;
lt::error err = lt::Post("cv40:/0/camera/0/exposure", body, response);
```

▼ *Python*

```
body = {
    "gainLimits": [2.5, 15.0]
}
```

```
response, err = Post("cv40:/0/camera/0/exposure", body)
```

5.3.12. Visuals Object

The **visuals** object regroups parameters to enhance the camera picture. It includes sharpness, anisotropic and bilateral denoising filters, shadow lighting effect, flip and zoom settings.

anisotropic `int`

Strength of the anisotropic denoising filter `[0 .. 100]`.

bilateral `int`

Strength of the bilateral denoising filter `[0 .. 100]`.

flip `string`

Flip **Horizontally** and/or **Vertically** the camera. `none`, `x`, `y`, `xy`.

shadow lighting gain `float64`

Artificially increase the luminosity level of the camera with a non linear gain on darker pixels `[1.0 .. 2.5]`.

sharpness `float64`

Strength of the sharpness filter `[1.0 .. 9.0]`.

sharpnessFloor `int`

Floor threshold at which the sharpness filter begin to take effect `[0 .. 50]`.

zoom `float64`

Digital zoom `[1.0 .. 4.0]`.

Visuals object

```
{
    "anisotropic": 0,
    "bilateral": 0,
    "flip": "none",
    "shadowLightingGain": 1,
    "sharpness": 3.1,
    "sharpnessFloor": 0,
    "zoom": 1
}
```

5.3.13. View Visuals Settings

To retrieve the camera visuals parameters, send a GET request to the **visuals** endpoint.

Parameters

None.

Response

Returns the visuals object if the request succeeded.

GET /:board/camera/:id/visuals

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/visuals",
    "body": null
}
```

response

```
{
    "anisotropic": 0,
    "bilateral": 0,
    "flip": "none",
    "shadowLightingGain": 1,
    "sharpness": 3.1,
    "sharpnessFloor": 0,
    "zoom": 1
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/visuals
```

▼ *GO*

```
var response lt.CameraVisuals // struct to store the response
err := lt.Get("cv40:/0/camera/0/visuals", &response)
```

▼ *C++*

```
lt::CameraVisuals response;
lt::error err = lt::Get("cv40:/0/camera/0/visuals", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/visuals")
```

5.3.14. Update Visuals Settings

Change the camera visuals as required by your application by sending a POST request to the **visuals** endpoint.

Parameters

Include settings of the visuals object you want to update.
In the example, the `sharpness` is set to `2.5`.

Response

Returns the visuals object if the request succeeded.

POST /:board/camera/:id/visuals

request

```
{
    "method": "POST",
    "url": "cv40:/0/camera/0/visuals",
    "body": {
        "sharpness": 2.5
    }
}
```

response

```
{
    "anisotropic": 0,
    "bilateral": 0,
    "flip": "none",
    "shadowLightingGain": 1.0,
    "sharpness": 2.5,
    "sharpnessFloor": 0,
    "zoom": 1
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/camera/0/visuals -d sharpness=2.5
```

▼ *GO*

```
body := lt.JSON{
    Sharpness: 2.5,
}
var response lt.CameraVisuals // struct to store the response
err := lt.Post("cv40:/0/camera/0/visuals", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"sharpness", 2.5}
};
lt::CameraVisuals response;
lt::error err = lt::Post("cv40:/0/camera/0/visuals", body, response);
```

▼ *Python*

```
body = {
    "sharpness": 2.5
}
response, err = Post("cv40:/0/camera/0/visuals", body)
```

5.3.15. Button Object

The **button** object allows to capture the button pin state.

description `string`

Description of the button pin.

pressed `bool`

Is the button currently 'pressed'.

pressedCount `int`

Number of times the button has been pressed since the last 'release' event.

timestamp `int64`

The unix timestamp of the last 'press' or 'release' event in microseconds.

Button object

```
{
    "description": "0/camera/0/buttons/0",
    "pressed": false,
    "pressedCount": 0,
    "timestamp": 0
}
```

5.3.16. View Button State

To retrieve the button status, send a GET request to the **button/:pin** endpoint.

Parameters

None.

Response

Returns the button object if the request succeeded.

GET /0/camera/0/buttons/:pin

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/buttons/0",
    "body": null
}
```

response

```
{
    "description": "0/camera/0/buttons/0",
    "pressed": false,
    "pressedCount": 0,
    "timestamp": 0
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/buttons/0
```

▼ *golang*

```
var response lt.Button // struct to store the response
err := lt.Get("cv40:/0/camera/0/buttons/0", &response)
```

▼ *C++*

```
lt::Button response; // struct to store the response
lt::error err = lt::Get("cv40:/0/camera/0/buttons/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/buttons/0")
```

5.3.17. Buttons Object

The **buttons** object is a collection of button objects. It allows to capture the state of multiple buttons at once.

5.3.18. View Buttons State

To retrieve the status of all buttons at once, send a GET request to the **buttons** endpoint.

Parameters

None.

Response

Returns the buttons object if the request succeeded.

GET /0/camera/0/buttons

request

```
{
    "method": "GET",
    "url": "cv40:/0/camera/0/buttons",
    "body": null
}
```

response

```
{
    {
        "description": "0/camera/0/buttons/0",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/camera/0/buttons/1",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/camera/0/buttons/2",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
    {
        "description": "0/camera/0/buttons/3",
        "pressed": false,
        "pressedCount": 0,
        "timestamp": 0
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/camera/0/buttons
```

▼ *golang*

```
var response lt.Buttons // struct to store the response
err := lt.Get("cv40:/0/camera/0/buttons", &response)
```

▼ *C++*

```
lt::Buttons response; // struct to store the response
lt::error err = lt::Get("cv40:/0/camera/0/buttons", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/camera/0/buttons/0")
```

5.4. HDMI Output

Allows to configure the physical hdmi outputs on a LT board.

:board	hdmi-out	:id
GET		GET POST

```
GET     /:board/hdmi-out/:id
POST    /:board/hdmi-out/:id
```

board

Device position into the host [0 .. 1].

id

hdmi-out index number 0.

5.4.1. HDMI Output Object

source `string`

The hdmi-out source. Values can be `auto`, `camera/:id`.

overlay `string`

The overlay source applied on the hdmi-out. Values can be `canvas/:id` or `none` for no overlay.

overlayMode `string`

Controls how overlay content is processed on the output device. This parameter affects the quality and performance of overlays. Values can be `performance` or `quality`.

format `string`

The color format applied to the output. Values can be `auto`, `rgb444`, `yuv444` and `yuv422`.

link `string`

The hdmi-out link carrier. Values can be `auto`, `fhd` or `off`.

---

audio `object`

The audio object for the HDMI Output.

video `object`

The video object for the HDMI Output.

HDMI Output object


```
{
    "source": "auto",
    "overlay": "none",
    "overlayMode": "performance",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "none",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```


5.4.2. View HDMI Output Status

To retrieve the hdmi-out configuration, send a GET request to the **hdmi-out/:id** endpoint.

Parameters

None.

Response

Returns the hdmi-out object if the request succeeded.

GET /:board/hdmi-out/:id

request

```
{
    "method": "GET",
    "url": "cv40:/0/hdmi-out/0",
    "body": null
}
```

response

```
{
    "source": "auto",
    "overlay": "none",
    "overlayMode": "performance",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "none",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/hdmi-out/0
```

▼ *GO*

```
var response lt.Output // struct to store the response
err := lt.Get("cv40:/0/hdmi-out/0", &response)
```

▼ *C++*

```
lt::HdmiOutput response; // struct to store the response
lt::error err = lt::Get("cv40:/0/hdmi-out/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/hdmi-out/0")
```

5.4.3. Configure HDMI Output

In order to update the hdmi-out configuration, send a POST request to the **hdmi-out/:id** endpoint with the desired settings.

Parameters

Include settings of the hdmi-out object you want to update.
In the example, the hdmi-out `overlay` is set to `canvas/0`.

Response

Returns the hdmi-out object if the request succeeded.

POST /:board/hdmi-out/:id

request

```
{
    "method": "POST",
    "url": "cv40:/0/hdmi-out/0",
    "body": {
        "overlay": "canvas/0"
    }
}
```

response

```
{
    "source": "auto",
    "overlay": "canvas/0",
    "overlayMode": "performance",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/hdmi-out/0 -d overlay=canvas/0
```

▼ *GO*

```
body := lt.JSON{
    "overlay": "canvas/0",
}
var response lt.Output // struct to store the response
```

```
err := lt.Post("cv40:/0/hdmi-out/0", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"overlay", "canvas/0"}
};
lt::HdmiOutput response; // struct to store the response
lt::error err = lt::Post("cv40:/0/hdmi-out/0", body, response);
```

▼ *Python*

```
body = {
    "overlay": "canvas/0"
}
response, err = Post("cv40:/0/hdmi-out/0", body)
```

5.5. SDI Output

Allows to configure the physical sdi outputs on a LT board.

:board	sdi-out	:id
GET		GET POST

GET /:board/sdi-out/:id

POST /:board/sdi-out/:id

board

Device position into the host [0 .. 1].

id

sdi-out index number 0.

5.5.1. SDI Output Object

source `string`

The sdi-out source. Values can be `auto`, `camera/:id`.

overlay `string`

The overlay source applied on the sdi-out. Values can be `none` or `canvas/:id`.

format `string`

The color format applied to the output. Values is `yuv422`.

link `string`

The sdi-out link carrier. Values can be `auto`, `uhd_4x3g`, `fhd` or `off`.

---

audio `json`

Audio signal object.

→ **description** `string`

A short description of the audio signal.

→ **format** `string`

The audio sample format `pcm`.

→ **channels** `int`

The number of audio channels.

→ **samplerate** `int`

The number of audio samples per second.

→ **depth** `int`

The number of bits per audio sample.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

SDI Output object

```
{
    "source": "auto",
    "overlay": "none",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "framerate": 0,
        "size": [0, 0],
        "interlaced": false,
        "signal": "none"
    }
}
```

video `json`

Video signal object.

→ **description** `string`

The video signal short description.

→ **format** `string`

The pixel color format `rgb444`, `yuv444` or `yuv422`.

→ **framerate** `float`

The number of video frames per second.

→ **size** `[2]int`

The video frame width and height in pixel units.

→ **interlaced** `bool`

The video frame interlaced status.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

5.5.2. View SDI Output Status

To retrieve the sdi-out configuration, send a GET request to the **sdi-out/:id** endpoint.

Parameters

None.

Response

Returns the sdi-out object if the request succeeded.

GET /:board/sdi-out/:id

request

```
{
    "method": "GET",
    "url": "cv40:/0/sdi-out/0",
    "body": null
}
```

response

```
{
    "source": "auto",
    "overlay": "none",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "framerate": 0,
        "size": [0, 0],
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/0/sdi-out/0
```

▼ *GO*

```
var response lt.Output // struct to store the response
err := lt.Get("cv40:/0/sdi-out/0", &response)
```

▼ *C++*

```
lt::SdiOutput response; // struct to store the response
lt::error err = lt::Get("cv40:/0/sdi-out/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/0/sdi-out/0")
```

5.5.3. Configure SDI Output

In order to update the sdi-out configuration, send a POST request to the **sdi-out/:id** endpoint with the desired settings.

Parameters

Include settings of the sdi-out object you want to update.
In the example, the sdi-out `overlay` is set to `canvas/0`.

Response

Returns the sdi-out object if the request succeeded.

POST /:board/sdi-out/:id

request

```
{
    "method": "POST",
    "url": "cv40:/0/sdi-out/0",
    "body": {
        "overlay": "canvas/0"
    }
}
```

response

```
{
    "source": "auto",
    "overlay": "canvas/0",
    "format": "auto",
    "link": "auto",
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "framerate": 0,
        "size": [0, 0],
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/0/sdi-out/0 -d overlay=canvas/0
```

▼ *GO*

```
body := lt.JSON{
    "overlay": "canvas/0",
}
var response lt.Output // struct to store the response
err := lt.Post("cv40:/0/sdi-out/0", body, &response)
```

▼ *C++*

```
lt::json body = {
    {"overlay", "canvas/0"}
};
lt::SdiOutput response; // struct to store the response
lt::error err = lt::Post("/0/sdi-out/0", body, response);
```

▼ *Python*

```
body = {
    "overlay": "canvas/0"
}
response, err = Post("cv40:/0/sdi-out/0", body)
```

5.6. Canvas

The **canvas** endpoint is both a virtual audio/video source and a dynamic synthetic image generator which supports draw operations. It could be used to emulate the LT boards video inputs and to send overlay images onto the hdmi and/or sdi outputs.

Data operations

canvas	:id	data
	GET DELETE	POST
		file
		POST

```
GET     /canvas/:id
POST    /canvas/:id/:format/data
POST    /canvas/:id/:format/file
```

id

canvas index number `[0 .. 3]`.

Draw operations

canvas	:id GET	init POST
		text POST
		line POST
		ellipse POST
		rectangle POST
		image POST
		video POST
		clear POST
		op POST
		ops POST

```
GET     /canvas/:id
POST    /canvas/:id/init
POST    /canvas/:id/line
POST    /canvas/:id/ellipse
POST    /canvas/:id/rectangle
POST    /canvas/:id/image
POST    /canvas/:id/video
POST    /canvas/:id/clear
POST    /canvas/:id/op
POST    /canvas/:id/ops
```

id

canvas index number `[0 .. 3]`.

5.6.1. Canvas Object

audio `json`

Audio signal object.

→ **description** `string`

A short description of the audio signal.

→ **format** `string`

The audio sample format `pcm`.

→ **channels** `int`

The number of audio channels.

→ **samplerate** `int`

The number of audio samples per second.

→ **depth** `int`

The number of bits per audio sample.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

Canvas object

```
{
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

video `json`

Video signal object.

→ **description** `string`

The video signal short description.

→ **format** `string`

The pixel color format `rgb444`, `yuv444` or `yuv422`.

→ **framerate** `float`

The number of video frames per second.

→ **size** `[2]int`

The video frame width and height in pixel units.

→ **interlaced** `bool`

The video frame interlaced status.

→ **signal** `string`

`none` (not found), or `locked` (ready to use).

5.6.2. View Canvas Status

To retrieve the canvas signal status, send a GET request to the **canvas/:id** endpoint

Parameters

None.

Response

Returns the canvas object if the request succeeded.

GET /canvas/:id

request

```
{
    "method": "GET",
    "url": "cv40:/canvas/0",
    "body": null
}
```

response

```
{
    "cpu": 0,
    "fpga": 0,
    "model": "",
    "sn": 0,
    "audio": {
        "description": "",
        "format": "",
        "channels": 0,
        "samplerate": 0,
        "depth": 0,
        "signal": "none"
    },
    "video": {
        "description": "",
        "format": "",
        "size": [0, 0],
        "framerate": 0,
        "interlaced": false,
        "signal": "none"
    }
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl get cv40:/canvas/0
```

▼ *GO*

```
var response lt.Camera // struct to store the response
err := lt.Get("cv40:/canvas/0", &response)
```

▼ *C++*

```
lt::Camera response; // struct to store the response
lt::error err = lt::Get("cv40:/canvas/0", response);
```

▼ *Python*

```
response, err = Get("cv40:/canvas/0")
```

5.6.3. Delete Operation

Clear the canvas to a "NO SIGNAL" equivalent. Helps to simulate a video input loss.

Parameters

None.

Response

Returns an error if the request failed.

DELETE /canvas/:id

request

```
{
    "method": "DELETE",
    "url": "cv40:/canvas/0",
    "body": null
}
```

response

```
{

}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl delete cv40:/canvas/0
```

▼ *GO*

```
err := lt.Delete("cv40:/canvas/0", nil, nil)
```

▼ *C++*

```
lt::error err = lt::Delete("cv40:/canvas/0", nullptr, nullptr);
```

▼ *Python*

```
err = Delete("cv40:/canvas/0")
```

5.6.4. Init Operation

Clear the canvas and fill the background with the specified file, pattern or color.

Parameters

op `string`

Operation identifier `init`.

source `string`

The canvas *size* and *framerate* is derived from the file content. Supported formats are `jpeg`, `png`, `bmp` and `mp4` files.
In the example, the **source** is a mp4 video file.

color `[4]int`

The RGBA background color with transparency. Default `[0,0,0,0]`.

size `[2]int`

Set the canvas width and height. Default `[3840,2160]`.

framerate `float`

Set the canvas refresh rate. Default `30.0`.

Response

Returns the init operation parameters if the request succeeded.

POST /canvas/:id/init

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/init",
    "body": {
        "source": "video.mp4",
    }
}
```

response

```
{
    "op": "init",
    "source": "video.mp4",
    "color": [0,0,0,0],
    "size": [3840,2160],
    "framerate": 30.0
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/init -d source=video.mp4
```

▼ *GO*

```
body := lt.JSON{
    "source": "video.mp4",
}
err := lt.Post("cv40:/canvas/0/init", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "video.mp4"}
};
lt::error err = lt::Post("cv40:/canvas/0/init", body, nullptr);
```

▼ *Python*

```
body = {
```

```
    "source": "video.mp4"
}
resp, err = Post("cv40:/canvas/0/init", body)
```

5.6.5. Text Operation

Draw text onto the canvas.

Parameters

op `string`

Operation identifier `text`.

text `string`

Text to draw. **This parameter is mandatory.**

align `string`

Set the text position into the container.The possible values are `top-left`, `top`, `top-right`, `left`, `center`, `right`, `bottom-left`, `bottom` and `bottom-right`. Default is `center`.

font `string`

Font type. Default is `regular`, could also be `mono` and `smallcaps`.

fontSize `int`

Font size in pt unit. Default is `32`.

italic `bool`

Draw the text with the ***italic*** attribute. Default `false`.

bold `bool`

Draw the text with the ***bold*** attribute. Default `false`.

color `[4]int`

The RGBA shape color with transparency. Default `{0,0,0,255}`.

angle `float`

Rotation angle in degree unit. Default `0`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**. ---

POST /canvas/:id/text

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/text",
    "body": {
        "text": "hello world!",
        "size": [3840,2160],
    }
}
```

response

```
{
    "op": "text",
    "text": "hello world!",
    "align": "center",
    "font": "regular",
    "fontSize": 32,
    "italic": false,
    "bold": false,
    "color": [255, 255, 255, 255],
    "angle": 0,
    "position": [0, 0],
    "size": [3840, 2160],
    "anchor": [0, 0]
}
```

Response

Returns the text operation parameters if the request succeeded.

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/text -d text="hello world!"
```

▼ *GO*

```
body := lt.JSON{
    "text": "hello world!",
}
err := lt.Post("cv40:/canvas/0/text", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"text", "hello world!"}
};
lt::error err = lt::Post("cv40:/canvas/0/text", body, nullptr);
```

▼ *Python*

```
body = {
    "text": "hello world!"
}
resp, err = Post("cv40:/canvas/0/text", body)
```

5.6.6. Line Operation

Draw a line whose top left anchor is `(x,y)` coordinates.

Parameters

op `string`

Operation identifier `line`.

width `int`

The shape width size in pixel unit. Default `1`.

color `[4]int`

The RGBA shape color with transparency. Default `{0,0,0,255}`.

pattern `[]int`

The dash size pattern in pixel units. The pattern is repeated. Default no dash pattern: `{}`.

angle `float`

Rotation angle in degree unit. Default `0`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**. **This parameter is mandatory.**

Response

Returns the line operation parameters if the request succeeded.

POST /canvas/:id/line

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/line",
    "body": {
        "position": [0,0],
        "size": [3840,2160],
        "color": [255,0,0,255]
    }
}
```

response

```
{
    "op": "line",
    "width": 1,
    "color": [255, 0, 0, 255],
    "pattern": null,
    "angle": 0,
    "position": [0, 0],
    "size": [3840, 2160],
    "anchor": [0, 0]
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/line \
    -d position=0,0 \
    -d size=3840,2160 \
    -d color=255,0,0,255
```

▼ *GO*

```
body := lt.JSON{
    "position": [0,0],
    "size": [3840,2160],
    "color": [255,0,0,255]
}
```

```
err := lt.Post("cv40:/canvas/0/line", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"position", {0,0}},
    {"size", {3840,2160}},
    {"color", {255,0,0,255}}
};
lt::error err = lt::Post("cv40:/canvas/0/line", body, nullptr);
```

▼ *Python*

```
body = {
    "position": [0, 0],
    "size": [3840, 2160],
    "color": [255, 0, 0, 255]
}
resp, err = Post("cv40:/canvas/0/line", body)
```

5.6.7. Ellipse Operation

Draw an ellipse whose top left anchor is `(x,y)` coordinates.

Parameters

op `string`

Operation identifier `ellipse`.

width `int`

The shape width size in pixel unit. Default `1`.

color `[4]int`

The RGBA shape color with transparency. Default `{0,0,0,255}`.

pattern `[]int`

The dash size pattern in pixel units. The pattern is repeated. Default no dash pattern: `{}`.

fill `[4]int`

Fill the shape with a RGBA color. Default is `{0,0,0,0}`.

angle `float`

Rotation angle in degree unit. Default `0`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**. **This parameter is mandatory.**

Response

Returns the ellipse operation parameters if the request succeeded.

▼ **Examples**

POST /canvas/:id/ellipse

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/ellipse",
    "body": {
        "position": [0,0],
        "size": [3840,2160],
        "color": [255,0,0,255],
        "fill": [0,255,0,255]
    }
}
```

response

```
{
    "op": "ellipse",
    "width": 10,
    "color": [255, 0, 0, 255],
    "pattern": null,
    "fill": [0, 255, 0, 255],
    "angle": 0,
    "position": [0, 0],
    "size": [3840, 2160],
    "anchor": [0, 0]
}
```

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/ellipse \
    -d position=0,0 \
    -d size=3840,2160 \
    -d color=255,0,0,255 \
    -d fill=0,255,0,255
```

▼ *GO*

```
body := lt.JSON{
    "position": [0,0],
    "size": [3840,2160],
    "color": [255,0,0,255],
    "fill": [0,255,0,255]
}
err := lt.Post("cv40:/canvas/0/ellipse", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"position", {0,0}},
    {"size", {3840,2160}},
    {"color", {255,0,0,255}},
    {"fill", {0,255,0,255}}
};
lt::error err = lt::Post("cv40:/canvas/0/ellipse", body, nullptr);
```

▼ *Python*

```
body = {
    "position": [0,0],
    "size": [3840,2160],
    "color": [255,0,0,255],
    "fill": [0,255,0,255]
}
resp, err = Post("cv40:/canvas/0/ellipse", body)
```

5.6.8. Rectangle Operation

Draw a rectangle whose top left anchor is `(x,y)` coordinates.

Parameters

op `string`

Batch operation identifier `rectangle`.

width `int`

The shape width size in pixel unit. Default `1`.

color `[4]int`

The RGBA shape color with transparency. Default `{0,0,0,255}`.

pattern `[]int`

The dash size pattern in pixel units. The pattern is repeated. Default no dash pattern: `{}`.

fill `[4]int`

Fill the shape with a RGBA color. Default is `{0,0,0,0}`.

rounded `int`

The shape corner radius in pixel unit. Default `0`.

angle `float`

Rotation angle in degree unit. Default `0`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**. **This parameter is mandatory.**

Response

Returns the rectangle operation parameters if the request succeeded.

POST /canvas/:id/rectangle

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/rectangle",
    "body": {
        "position": [100,100],
        "size": [400,400],
        "fill": [0,0,255,255]
    }
}
```

response

```
{
    "op": "rectangle",
    "width": 1,
    "color": [255, 255, 255, 255],
    "pattern": null,
    "fill": [0, 0, 255, 255],
    "rounded": 0,
    "angle": 0,
    "position": [100, 100],
    "size": [400, 400],
    "anchor": [0, 0]
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/rectangle \
```

```
    -d position=100,100 \
    -d size=400,400 \
    -d fill=0,0,255,255
```

▼ *GO*

```
body := lt.JSON{
    "position": [100,100],
    "size": [400,400],
    "fill": [0,0,255,255]
}
err := lt.Post("cv40:/canvas/0/rectangle", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"position", {100,100}},
    {"size", {400,400}},
    {"fill", {0,0,255,255}}
};
lt::error err = lt::Post("cv40:/canvas/0/rectangle", body, nullptr);
```

▼ *Python*

```
body = {
    "position": [100,100],
    "size": [400,400],
    "fill": [0,0,255,255]
}
resp, err = Post("cv40:/canvas/0/rectangle", body)
```

5.6.9. Image Operation

There are two ways to draw an image on the canvas:

- Using a file path with the `source` parameter. The `format`, `data`, `width` and `height` parameters are ignored.
- Using a data buffer with the `data` parameter. The `format` parameter is mandatory and if a raw format is used (i.e. `rgba` or `rgb`), the `width` and `height` parameters are required too.

Parameters

op `string`

Operation identifier `image`.

source `string`

Filepath. Supported formats are `jpeg`, `png` and `bmp` files.

angle `float`

Rotation angle in degree unit. Default `0`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**. **This parameter is mandatory.**

---

format `string`

The image data format. Could be `rgba`, `rgb`, `bmp`, `jpeg` or `png`.

data `[]byte`

Image data buffer.

width `int`

Image width. Mandatory for `rgba` or `rgb` data buffer.

height `int`

Image height. Mandatory for `rgba` or `rgb` data buffer.

Response

Returns the image operation parameters if the request succeeded.

▼ **Examples**

POST /canvas/:id/image

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/image",
    "body": {
        "source": "C:\\image.png",
        "position": [0,0],
        "size": [640,480]
    }
}
```

response

```
{
    "op": "image",
    "source": "C:\\image.png",
    "angle": 0,
    "position": [0, 0],
    "size": [640, 480],
    "anchor": [0, 0],
    "format": "",
    "data": null,
    "width": 0,
    "height": 0
}
```

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/image \
    -d source=C:\\image.png \
    -d position=0,0 \
    -d size=640,480
```

▼ *GO*

```
body := lt.JSON{
    "source": "C:\\image.png",
    "position": [0,0],
    "size": [640,480]
}
err := lt.Post("cv40:/canvas/0/image", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "C:\\image.png"},
    {"position", {0,0}},
    {"size", {640,480}}
};
lt::error err = lt::Post("cv40:/canvas/0/image", body, nullptr);
```

▼ *Python*

```
body = {
    "source": "C:\\image.png",
    "position": [0,0],
    "size": [640,480]
}
resp, err = Post("cv40:/canvas/0/image", body)
```

5.6.10. Video Operation

Place a video on the canvas.

Parameters

op `string`

Batch operation identifier `video`.

source `string`

Supported sources are `:board/camera/:id` and `canvas/:id`.

position `[2]int`

The top left corner **{x, y}** position of the container. Default is `{0,0}`.

size `[2]int`

The container size **{width, height}**.

Response

Returns the video operation parameters if the request succeeded.

POST /canvas/:id/video

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/video",
    "body": {
        "source": "0/camera/0",
        "position": [0,0],
        "size": [1920,1080]
    }
}
```

response

```
{
    "op": "video",
    "source": "0/camera/0",
    "position": [0, 0],
    "size": [1920, 1080],
    "anchor": [0, 0]
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/canvas/0/video \
    -d source=0/camera/0 \
    -d position=0,0 \
    -d size=1920,1080
```

▼ *GO*

```
body := lt.JSON{
    "source": "0/camera/0",
    "position": [0,0],
    "size": [1920,1080]
}
err := lt.Post("cv40:/canvas/0/video", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "0/camera/0"},
    {"position", {0,0}},
    {"size", {1920,1080}}
};
lt::error err = lt::Post("cv40:/canvas/0/video", body, nullptr);
```

▼ *Python*

```
body = {
    "source": "0/sdi-in/0",
    "position": [0,0],
```

```
    "size": [1920,1080]
}
resp, err = Post("cv40:/canvas/0/video", body)
```

5.6.11. Clear Operation

Clear the canvas and fill the background with the specified color or remove video sources from the canvas.

Parameters

op `string`

Operation identifier `clear`.

source `string`

Supported sources are `:board/sdi-in/:id`, `:board/hdmi-in/:id`, `canvas/:id` and `all`.

color `[4]int`

The RGBA background color with transparency. Default `[0,0,0,0]`.

position `[2]int`

The top left corner position for partial clearing. Default is `[0,0]`.

size `[2]int`

Area dimensions to clear. If omitted, clears the entire canvas.

thickness `int`

Expand the area to clear by this amount of pixels on all sides. Default `0`.

Response

Returns the clear operation parameters if the request succeeded.

POST /canvas/:id/op

request

```
{
    "method": "POST",
    "url": "cv40:/canvas/0/op",
    "body": {
        "op": "clear",
        "source": "0/sdi-in/0"
    }
}
```

response

```
{
    "op": "clear",
    "source": "0/sdi-in/0",
    "color": [0,0,0,0],
    "position": [0,0],
    "size": [0,0],
    "thickness": 0
}
```

▼ **Examples**

▼ *ecurl - Clear entire canvas*

```
$ ecurl post cv40:/canvas/0/op -d op=clear -d color=0,0,0,255
```

▼ *ecurl - Remove specific video source*

```
$ ecurl post cv40:/canvas/0/op -d op=clear -d source=0/hdmi-in/0
```

▼ *ecurl - Remove all video sources*

```
$ ecurl post cv40:/canvas/0/op -d op=clear -d source=all
```

▼ *GO*

```
body := lt.JSON{
    "op": "clear",
    "source": "0/sdi-in/0",
}
err := lt.Post("cv40:/canvas/0/op", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"op", "clear"},
    {"source", "0/sdi-in/0"}
};
lt::error err = lt::Post("cv40:/canvas/0/op", body, nullptr);
```

5.6.12. Batch Operations

Draw operations in batch.

Parameters

ops `[]JSON`

Array of canvas operations.

Response

Returns the operations if succeeded.

POST /canvas/:id/ops

request

```
$ ecurl post cv40:/canvas/0/ops \
    -d ops=@draw.json
```

response

```
{
    "ops": { ... },
}
```

5.7. Client

The **client** endpoint retains the connection context, the living memory references and the running workers. Once a **client** is done with a resource, it has to delete it. If the **client** dies or ceases to communicate, the **cv40agent** will automatically collect the resources and clean them.

client	jobs	:id GET	start POST
			stop POST
			pause POST
	refs	:id DELETE	

```
GET     /client/jobs/:id
POST    /client/jobs/:id/start
POST    /client/jobs/:id/pause
POST    /client/jobs/:id/stop
DELETE /client/refs/:id
```

id

Object identifier.

5.7.1. Fetch Worker Updates

The long running task(s) (eg: recording a mp4) are child processe(s) of the **client(s)** which have initiated the request(s). These task(s) are processed by worker(s) that are attached into the **client(s)** context(s) with an **unique ID**.

Retrieving the updates periodically ensures that the tasks are properly processed and allow to fetch the data out of the **cv40agent**.

GET /client/job/:id

Please go to Section 5.8, “Workers” to learn the complete workflow usage.

5.7.2. Release Referenced Memory

Clients and **cv40agent** communicate by exchanging references on shared memory blocks.

Once processed, it is recommended to expressly release the references, otherwise the **cv40agent** memory pool can run out of shared memory blocks.

Depending to your development language (Garbage Collected or not), the SDK wrapper might automatically release the memory for you.

DELETE /client/ref/:id

Please go to Section 5.8, "Workers" to learn the complete workflow usage.

5.8. Workers

All the API processing is based on **Worker objects** created by the server (on behalf of the clients requests) to serve data or metadata packets. The workers creation endpoints are easily recognizable by their URLs patterns:

:url	data POST
	file POST

```
POST    /:url/data
POST    /:url/file
```

`url`

URL can be any valid API resource that point toward a `data` or a `file` endpoint.

The workers can serve streams under 3 types:

- `data` the de facto interface to **process data** into a third party application. These kind of workers use the host shared memory mechanisms with pooled buffers to distribute large chunk of data to multiple concurrent consumers.
- `file` this helps you **record files** onto the host hard drive. These workers supports splitting and containerized formats like mp4, asf, or avi. Each time that a file is finished or split, the **completed** field of the Worker object is set to true. The next request will point toward a new file via a redirected URL.

Finally, the Workers transfer packets from the LT agent to the LT clients. Packets may contains data, metadata, video, audio, ...

5.8.1. Worker Creation

A **type** has to be submitted to the **data** or **file** endpoint to create a worker. The type is a string that describes the data class `audio`, `image` and `video` and the data format. The type is a mandatory field and must be set to a valid value.

Audio: `audio/pcm`, `audio/wav`, `audio/aac`.

Image: `image/yuyv`, `image/yuv422`, `image/nv12`, `image/rgba`, `image/rgb`, `image/jpeg`, `image/png`, `image/bmp`.

Video: `video/yuyv`, `video/yuv422`, `video/nv12`, `video/rgba`, `video/rgb`, `video/jpeg`, `video/png`, `video/bmp`, `video/h264`, `video/mp4`.

5.8.1.1. Audio Data Worker

Create a worker object that serves audio data packets.

Parameters

media `string`

Media type identifier. Could be `audio/pcm` or `audio/aac`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

channels `int`

The number of audio channels. Default `2`.

samplerate `int`

The audio sample rate. Default `48000`.

depth `int`

The audio sample depth. Default `16`.

Response

Returns the location of the worker object onto the form of a **redirect** error.

▼ **Examples**

POST /:url/data

request

```
{
    "method": "POST",
    "url": "cv40:/:url/data",
    "body": {
        "media": "audio/pcm",
        "source": "0/hdmi-in/0",
        "channels": 2,
        "samplerate": 48000,
        "depth": 16
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

▼ *GO*

```
err := lt.Post("cv40:/:url/data", lt.AudioDataWorker{Media: "audio/pcm"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/data", lt::AudioDataWorker{ "audio/pcm" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/data", {'media': "audio/pcm"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.1.2. Image Data Worker

Create a worker object that serves one image data packet.

Parameters

media `string`

Media type identifier. Could be `image/yuyv`, `image/yuv422`, `image/nv12`, `image/rgba`, `image/rgb`, `image/jpeg`, `image/png` and `image/bmp`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

size `[2]int`

The image frame size. Let empty to use the default size.

Response

Returns the location of the worker object onto the form of a **redirect** error.

POST /:url/data

request

```
{
    "method": "POST",
    "url": "cv40:/:url/data",
    "body": {
        "media": "video/nv12",
        "source": "0/hdmi-in/0",
        "size": [1920, 1080]
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

▼ **Examples**

▼ *GO*

```
err := lt.Post("cv40:/:url/data", lt.ImageDataWorker{Media: "image/jpeg"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/data", lt::ImageDataWorker{ "image/jpeg" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/data", {'media': "image/jpeg"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.1.3. Video Data Worker

Create a worker object that continuously serves video data packets.

Parameters

media `string`

Media type identifier. Could be `video/yuyv`, `video/yuv422`, `video/nv12`, `video/rgba`, `video/rgb`, `video/jpeg`, `video/png`, `video/bmp`, `video/h264`, `video/hevc` and `video/mp4`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

size `[2]int`

The image frame size. Let empty to use the default size.

framerate `float`

The video frame rate. Let empty to use the default framerate.

Response

Returns the location of the worker object onto the form of a **redirect** error.

POST /:url/data

request

```
{
    "method": "POST",
    "url": "cv40:/:url/data",
    "body": {
        "media": "video/nv12",
        "source": "0/hdmi-in/0",
        "size": [1920, 1080],
        "framerate": 30
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

▼ **Examples**

▼ *GO*

```
err := lt.Post("cv40:/:url/data", lt.VideoDataWorker{Media: "video/nv12"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/data", lt::VideoDataWorker{ "video/nv12" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/data", {'media': "video/nv12"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.1.4. Audio File Worker

Create a worker object that records an audio file. The file is split when the file length or the file duration is reached.

Parameters

media `string`

Media type identifier. Could be `audio/pcm`, `audio/wav` or `audio/aac`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

channels `int`

The number of audio channels. Default `2`.

samplerate `int`

The audio sample rate. Default `48000`.

depth `int`

The audio sample depth. Default `16`.

location `string`

The file location.

duration `int`

The file duration in milliseconds to record. Default `0` (infinite).

splitSize `int`

The file split length in bytes. Default `0` (no split).

splitDuration `int`

The file split duration in milliseconds. Default `0` (no split).

Response

Returns the location of the worker object onto the form of a **redirect** error.

POST /:url/file

request

```
{
    "method": "POST",
    "url": "cv40:/:url/file",
    "body": {
        "media": "audio/wav",
        "source": "0/hdmi-in/0",
        "channels": 2,
        "samplerate": 48000,
        "depth": 16,
        "location": "/path/to/audio/directory",
        "duration": 0,
        "splitSize": 0,
        "splitDuration": 0
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/:url/file -d media=audio/wav
```

▼ *GO*

```
err := lt.Post("cv40:/:url/file", lt.AudioFileWorker{Media: "audio/wav"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/file", lt::AudioFileWorker{ "audio/wav" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/file", {'media': "audio/wav"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.1.5. Image File Worker

Create a worker object that records one image file.

Parameters

media `string`

Media type identifier. Could be `image/yuyv`, `image/yuv422`, `image/nv12`, `image/rgba`, `image/rgb`, `image/jpeg`, `image/png` and `image/bmp`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

size `[2]int`

The image frame size. Let empty to use the default size.

location `string`

The file location.

POST /:url/file

request

```
{
    "method": "POST",
    "url": "cv40:/:url/file",
    "body": {
        "media": "video/nv12",
        "source": "0/hdmi-in/0",
        "size": [1920, 1080],
        "location": "/path/to/image/directory"
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

Response

Returns the location of the worker object onto the form of a **redirect** error.

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/:url/file -d media=image/jpeg
```

▼ *GO*

```
err := lt.Post("cv40:/:url/file", lt.ImageFileWorker{Media: "image/jpeg"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/file", lt::ImageFileWorker{ "image/jpeg" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/file", {'media': "image/jpeg"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.1.6. Video File Worker

Create a worker object that records a video file. The file is split when the file length or the file duration is reached.

Parameters

media `string`

Media type identifier. Could be `video/yuyv`, `video/yuv422`, `video/nv12`, `video/rgba`, `video/rgb`, `video/jpeg`, `video/png`, `video/h264`, `video/hevc` and `video/mp4`.

source `string`

The audio board input source: `:board/hdmi-in/:id`, `:board/sdi-in/:id` and `canvas/:id`.

size `[2]int`

The image frame size. Let empty to use the default size.

framerate `float`

The video frame rate. Let empty to use the default framerate.

location `string`

The file location.

duration `int`

The file duration in seconds to record. Default `0` (infinite).

splitSize `int`

The file split length in bytes. Default `0` (no split).

splitDuration `int`

The file split duration in seconds. Default `0` (no split).

POST /:url/file

request

```
{
    "method": "POST",
    "url": "cv40:/:url/file",
    "body": {
        "media": "video/mp4",
        "source": "0/hdmi-in/0",
        "size": [1920, 1080],
        "framerate": 30,
        "location": "/path/to/video/directory",
        "duration": 0,
        "splitSize": 0,
        "splitDuration": 0
        "extra": {
            "hw": "",
            "bitrate": 0,
            "quality": 0,
            "gop": 0,
            "codec": "",
            "preset": ""
        }
    }
}
```

response

```
{
    "location": "cv40:/client/jobs/...",
    "error": "redirect"
}
```

Extra `string`

Video encoder configuration parameters that control the encoding process:

- `hw` - Hardware encoder to use: `"qsv"` (Intel), `"nvenc"` (NVIDIA), or `"amf"` (AMD)
- `bitrate` - Target bitrate in bits per second (e.g., 5000000 for 5 Mbps)
- `quality` - Quality/compression level (19-24, lower values = higher quality)
- `gop` - Group of Pictures - keyframe interval in frames
- `codec` - Video codec to use: `"h264"` or `"hevc"`
- `preset` - Preset for the encoder (e.g., `"ultrafast"`, `"faster"`, `"fast"`, `"medium"`, `"slow"`, `` `"slower"``, `"veryslow"`)

> **NOTE** Use either `bitrate` or `quality` for rate control, but not both simultaneously. Using `bitrate` creates a constant bitrate encoding, while `quality` creates variable bitrate encoding with consistent visual quality.

Response

Returns the location of the worker object onto the form of a **redirect** error.

▼ **Examples**

▼ *ecurl*

```
$ ecurl post cv40:/:url/file -d media=video/mp4
```

▼ *GO*

```
err := lt.Post("cv40:/:url/file", lt.VideoFileWorker{Media: "video/mp4"}, nil)
if !errors.Is(err, lt.ErrRedirect) {
    log.Fatal("worker creation failed:", err)
}
workerURL := lt.RedirectLocation(err)
```

▼ *C++*

```
lt::error err = lt::Post("cv40:/:url/file", lt::VideoFileWorker{ "video/mp4" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

▼ *Python*

```
resp, err = Post("cv40:/:url/file", {'media': "video/mp4"})
if not lt.ErrorIs(err, lt.ErrRedirect):
    exit(err)
workerURL = lt.RedirectLocation(err)
```

5.8.2. Worker Object

The Worker object is the result of a GET request onto a worker endpoint. It contains the worker status, data packets and metadata. A Worker might process one or multiples tracks and the SDK provides helpers functions to automatically parse the worker into a comprehensive structure with the contained audio and video packets.

name `string`

Name.

location `string`

Location.

start `int64`

Unix timestamp at which the worker started.

duration `int64`

Elapsed time since the worker started.

size `int`

Quantity of byte processed since the segment started.

status `string`

`running`, `paused`, `break` (file split) or `completed`.

packets `map[int]packet`

Packets maps packet or shared packet of video, audio or text data samples and/or metadata samples.

Worker object

```
{
    "name": "",
    "location": "",
    "start": 1644248369455566,
    "duration": 16667,
    "size": 4147200,
    "status": "completed",
    "packets": {
        "0": {
           "... packet object #0 ..."
        }
    }
}
```

5.8.3. Packet Object

The packet object wraps the data and the metadata of an audio, video, ... track.

The SDK provides a helper function to automatically parse the packets into a comprehensive structure.

track `int`

The track ID of the packet if the worker process multiple tracks.

type `string`

The packet type and format.

signal `string`

`none` (not found), or `locked` (ready to use).

timestamp `int64`

Unix timestamp at which the packet has been sampled.

data `[]byte`

The packet plain data buffer.

meta `JSON`

The metadata fields for audio and video. See audio and video metadata objects.

Packet object

```
{
    "track": 0,
    "type": "audio/pcm",
    "signal": "none",
    "timestamp": 1695816377020822,
    "data": "...",
    "meta": {
        "channels": 2,
        "samplerate": 48000,
        "depth": 16,
        "samples": 1600
    },
}
```

5.8.3.1. SharedPacket Object

This has the same description as the Packet object, use only for reference. To lower the cpu consumption and the latency, big data blocks are transmitted to the user using the OS standard shared memory mechanisms. No memory copy is involved in the packet transmission.

The SDK provides a helper function to automatically parse the shared packets into a comprehensive structure.

track `int`

The track ID of the packet if the worker process multiple tracks.

type `string`

The packet type and format.

signal `string`

`none` (not found), or `locked` (ready to use).

timestamp `int64`

Unix timestamp at which the packet has been sampled.

meta `JSON`

The packet metadata fields for video, audio, ...

ref `string`

The shared memory reference to be deleted once the data has been used.

client `string`

The client id which has made the request.

handle `string`

The handle that allows to access the shared memory.

size `int`

The shared memory block total capacity.

ptr `int`

The pointer at which the shared buffer start inside the shared memory block.

len `int`

The shared buffer length inside the shared memory block.

SharedPacket object


```
{
    "track": 0,
    "type": "video/yuyv",
    "signal": "locked",
    "timestamp": 1695815814430318,
    "len": 16588800,
    "meta": {
        "size": [1920, 1080],
        "framerate": 30,
        "interlaced": false,
        "keyframe": true
    },
    "ref": "cv40:/client/ref/...",
    "client": "q5jrzd2OIQuxCq1IJWICuA",
    "handle": "cv40_global_24",
    "size": 1275592704,
    "ptr": 478347264
}
```


5.8.3.2. Audio Metadata

Packets with `audio/*` type.

channels `int`

The number of channels.

samplerate `int`

The number of samples per second.

depth `int`

The number of bits per sample.

Samples `int`

The number of samples contained into the buffer.

Audio metadata

```
{
    "channels": 2,
    "samplerate": 48000,
    "depth": 16,
    "samples": 800,
}
```

5.8.3.3. Image Metadata

Packets with `image/*` type.

size `[2]int`

The image frame size.

Image metadata

```
{
    "size": [1920, 1080],
}
```

5.8.3.4. Video Metadata

Packets with `video/*` type.

size `[2]int`

The video frame size.

framerate `float`

The number of video frame per second.

interlaced `bool`

Is the frame interlaced.

keyframe `bool`

Is the frame intra coded.

Video metadata

```
{
    "size": [1920, 1080],
    "framerate": 60,
    "interlaced": false,
    "keyframe": true
}
```

5.8.4. Data Worker Workflow

Create a worker object that serves data packets. Data packets could be of type audio, image or video.

Create Worker

`> POST cv40:/canvas/0/yuyv/data`

- A worker is created.
- Check non-null error.
- Retrieve worker location with 'redirect' URL.

`> GET workerURL`

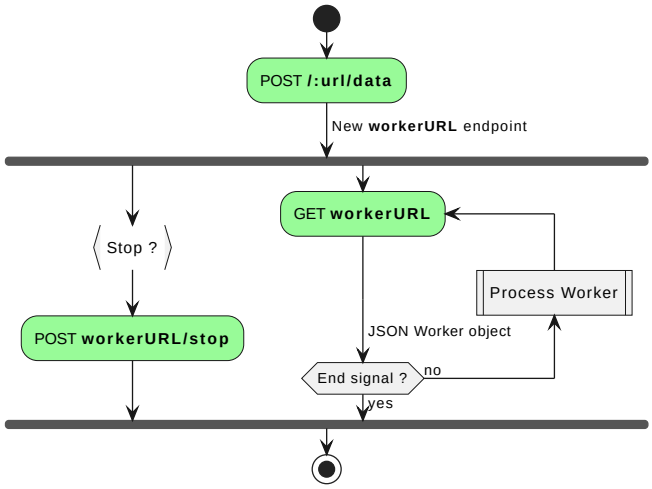
- JSON Worker object is returned.
- Check non-null error.

`> Check EndOfStream signal`

- Exit or pass to the next step.

`> Process Worker`

- See data worker processing below.
- Continue the worker loop until the EndOfStream signal is met.



> Monitor the Worker progress

- Start, duration, total processed bytes

> Loop over the Worker Packets

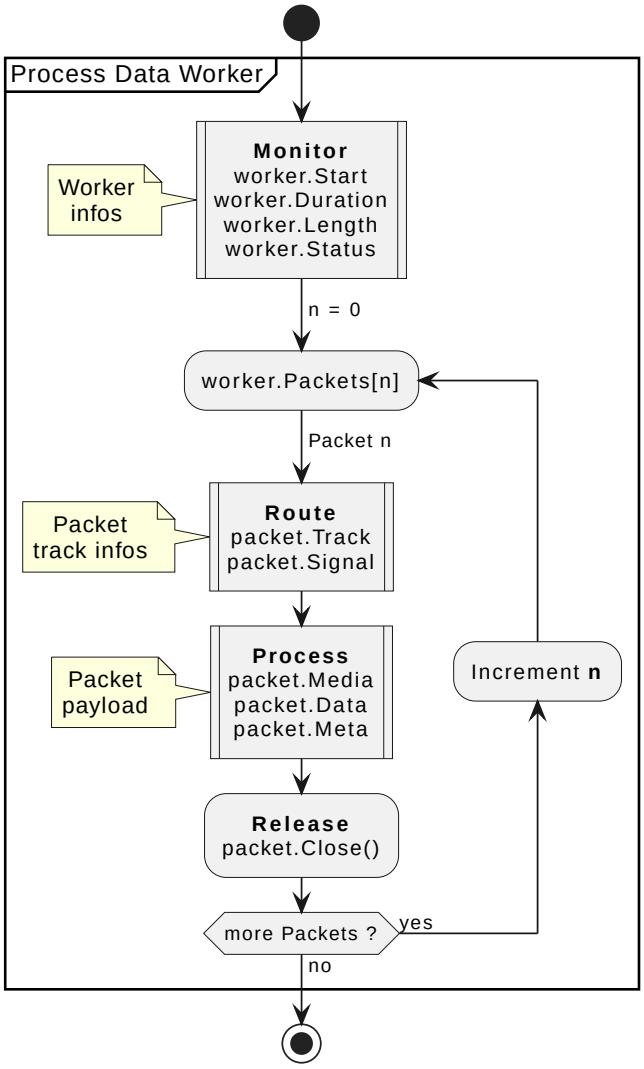
- Grab one packet.
- Check the packet track ID.
- Check the packet track signal.

> Process the Packet

- Check the media type field.
- Parse and load the metadata.
- Use the data.

> Release the Packet

- Call the packet.Close() function.
- Shared memory reference is released.
- Check non-null error.



5.8.5. File Worker Workflow

Create a worker object that records a file. The file is split when the file length or the file duration is reached. Files could be of type audio, image or video.

Create file worker

`> POST cv40:/canvas/0/png/file`

- A worker is created.
- Check non-null error.

`> GET worker`

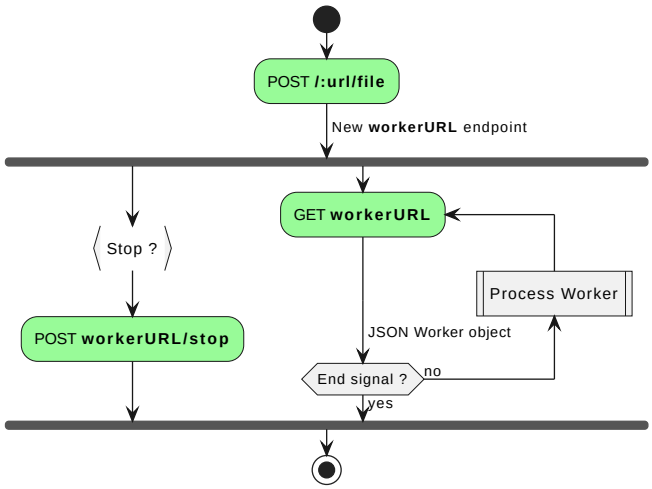
- JSON Worker object is returned.
- Check non-null error.

`> Check EndOfStream signal`

- Exit or pass to the next step.

`> Process Worker`

- See process Worker workflow above.
- Continue the worker loop



Process file worker

```
> GET cv40:/canvas/0/png/file
```

- JSON Worker object is returned.
- Check non-null error.

```
> Monitor the Worker progress
```

- Name, location
- Start, duration, total processed bytes, completed

```
> Loop over the Worker Packets
```

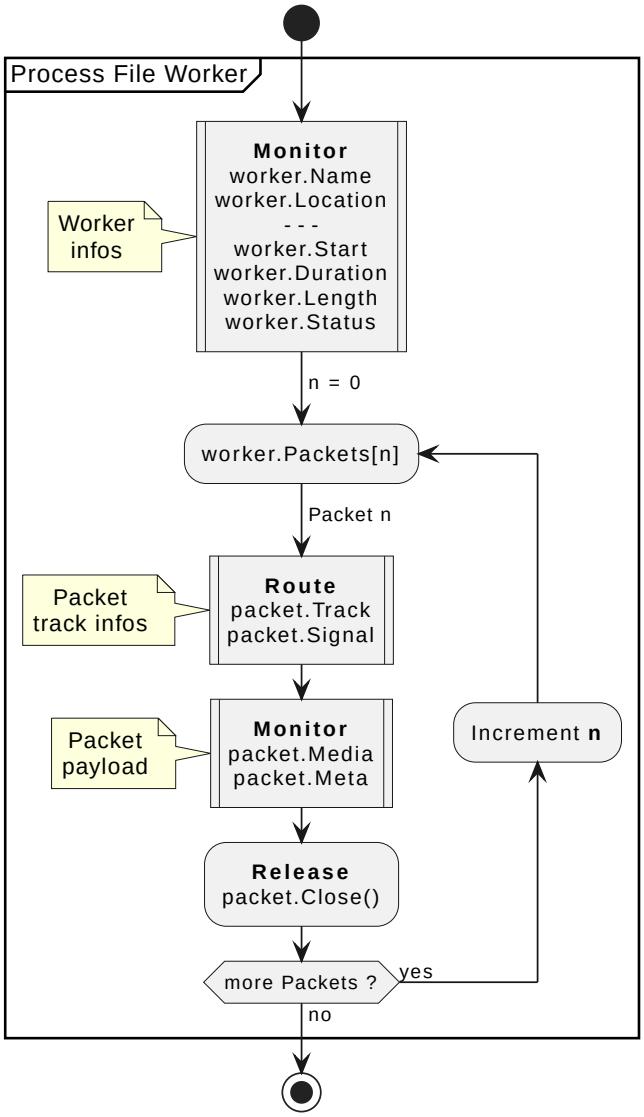
- Grab one packet.
- Check the packet track ID.
- Check the packet track signal.

```
> Process the Packet
```

- Check the media type field.
- Parse and load the metadata.

```
> Release the Packet
```

- Call the packet.Close() function.
- Check non-null error.



Chapter 6. Cheatsheet

/
GET

:board	buttons	:pin
GET	GET	GET

:board	camera	:id	data	
GET		GET	POST	
			file	
			POST	
			net	
			POST	
			white	
			GET POST	
			colors	
			GET POST	
			exposure	
			GET POST	
			visuals	
			GET POST	
			buttons	:pin
			GET	GET

:board	hdmi-out	:id
GET		GET POST

:board	sdi-out	:id
GET		GET POST

canvas	:id GET DELETE	data POST
		file POST
canvas	:id GET	init POST
		text POST
		line POST
		ellipse POST
		rectangle POST
		image POST
		video POST
		clear POST
		op POST
		ops POST

client	jobs	:id GET	start POST
			stop POST
			pause POST
	refs	:id DELETE	

Chapter 7. Changelog

```
# 1.3.0 (21/07/2025):
- Add agent configuration file
- Add NV12 native support
- Add independent RGB gain
- Add video encoder parameters (hardware accelerators, codecs, ...)
- Add dead pixels correction
- Improve overlay performance
- Improve agent reliability
- Improve programmation reliability
- Improve Linux support
- Update sdk
- Remove Windows 7 & 8 support
- Minor fixes

# 1.2.1 (11/02/2025):
- Improve player colorimetry
- Minor fixes

# 1.2.0 (22/01/2025):
- Add low light boost
- Add shadow lighting
- Add OSD feature into server
- Add overlay scaling option
- Improve DirectShow filters
- Update sdk
- Minor fixes

# 1.1.0 (22/08/2024):
- Add OSD feature (only works with UART interface)

# 1.0.0 (19/07/2024):
- First official release
```