LT100 API

The programmer guide

Version 3.2.1, 28/10/2024

Table of Contents

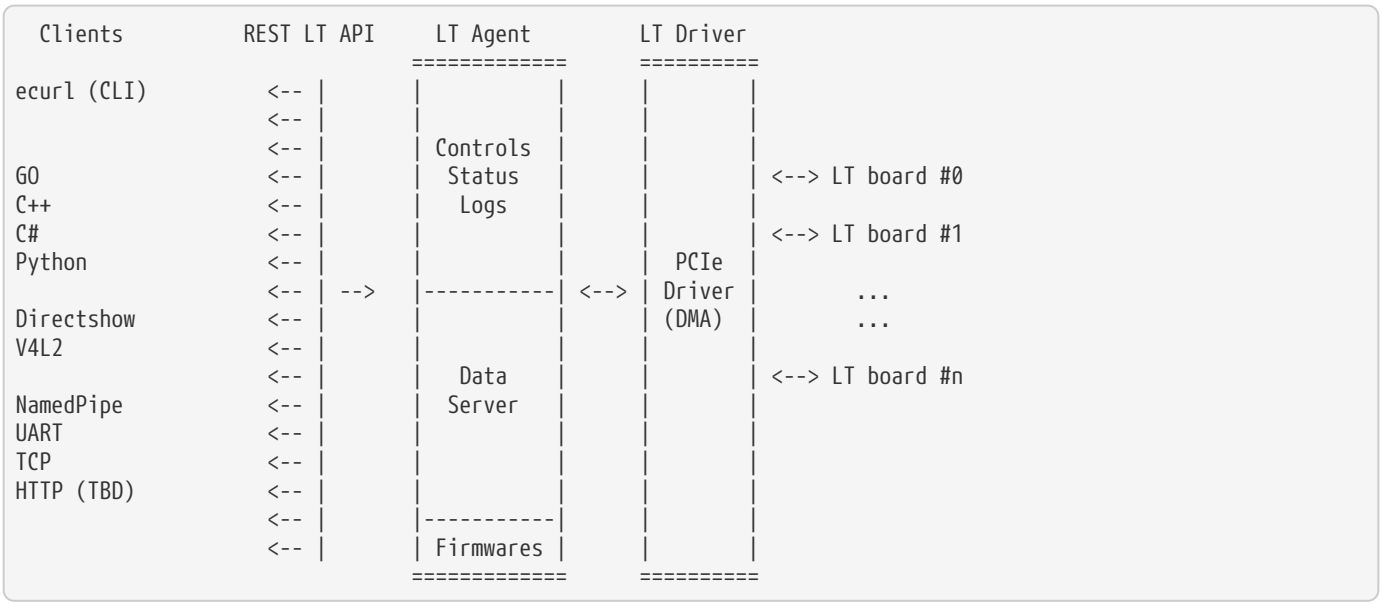
Chapter 1. Introduction

The **LT API** is organized around REST which stands for Representational State Transfer. This is an architectural pattern that describes how systems can expose a consistent interface. When people use the term REST API, they are generally referring to an API accessed with a predefined set of URLs.

These URLs represent various resources which are returned as JSON objects. Resources have one or more methods like `GET`, `POST` or `DELETE`.

The **LT API** requests are processed by a host service called **lt100agent**. The **lt100agent** uses the standard Operating System IPC, SG-DMA and shared memory mechanisms to serve the **LT API** requests with the minimum possible latency. Video and Audio data is distributed to the various consumers with shared memory segments to increase the performance with either the large data buffers and the concurrent access.

The **LT API** is structured in a way that the **LT boards** can be seen as *hardware as a service*. The hardware specific implementations are hidden behind a unique general API that let the users to focus only on what the boards are most useful for: grab, play, record and stream audio and video data.



Chapter 2. Installation

2.1. Windows

Download and run the latest **lt100install_x.x.x.exe** to deploy the **lt100 family** drivers, tools and services. If necessary the previous version will be uninstalled.

Once installed, the **lt100agent** service will be started automatically and will be (re)started automatically with each system (re)boot.

The **lt100agent** can be controlled using the Windows Services Manager or the **lt100agent** command line interface. Please 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 **lt100agent** service, so you can use them only if the service is running.

NOTE	To uninstall the **lt100 family** drivers, tools and services, run the uninstall script located in the installation directory.

2.2. Linux

Download and extract the latest **lt100install_x.x.x.tar.gz**, then execute the **lt100install.sh** script to deploy the **lt100 family** drivers, tools and services. If necessary the previous version will be uninstalled.

After a successful installation, the **lt100agent** daemon will be started automatically and will be (re)started automatically with each system (re)boot. If the installation failed, please check the **lt100install.log** file.

The **lt100agent** can be controlled using the **lt100agent** command line interface. Please see [control_service].

The **LT boards** plugged into the host should appear in the **/proc** directory. Please type the command below to check the boards status.

Get installed boards status

```
$ TBD
```

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 **lt100agent** service, so you can use them only if the service is running.

NOTE To uninstall the **lt100 family** drivers, tools and services, run the uninstall script located in the installation directory.

2.3. SDK

Download the latest **lt100sdk_x.x.x_{windows,linux}.zip** archive and extract it anywhere you want. The SDK contains the API documentation and the API libraries for the following languages **Go**, **C++**, **Python** and **C#**.

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

You can also make scripts with the **ecurl** command line interface tool. Please see [ecurl].

2.4. Tools

Two tools are installed along with the **lt100agent** service:

- **ecurl**, a command line tool to send REST API requests to the **lt100agent**.
- **ecam**, a graphical user interface to control and/or test the LT boards.

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

2.5. LT Board Firmware

If the boards installed into the host need a firmware update, the **lt100agent** service will automatically make an update when the service start. This step can take up to 2 minutes depending to the board type. The service availability will be delayed until the update is completed.

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

Boards status

```
$ lt100agent version
```

This is also possible to manually update the board firmwares.

Boards status with update

```
$ lt100agent update
```

2.6. lt100agent Controls

To use the **LT API** you need to ensure that the **lt100agent** is running on your host system.

The following commands will help you to control the **lt100agent**.

Install the service

```
$ lt100agent install
```

Uninstall the service

```
$ lt100agent uninstall
```

Check the service version

```
$ lt100agent version
```

Check the service status

```
$ lt100agent status
```

Start the service

```
$ lt100agent start
```

Stop the service

```
$ lt100agent stop
```

Restart the service

```
$ lt100agent restart
```

Run interactively (for debugging purpose, does not start the service)

```
$ lt100agent run
```

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 **lt100agent** at the installation stage. The tool has been developed with our SDK and is available for 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 **lt100agent** has to be running otherwise **ecurl** will not work.

3.1. GET command

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

Perform a GET requests to retrieve an individual API object designed by the <**url**>.

Example 1. GET

Retrieve lt100 family agent infos

```
$ ecurl get lt100:/
```

Retrieve lt100 family board infos located at index 0

```
$ ecurl get lt100:/0
```

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.

Example 2. POST

Create a virtual video input from a mp4 file

```
$ ecurl post lt100:/canvas/0/init -d source=video.mp4
```

3.3. DELETE Command

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

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

Example 3. DELETE

Unplug virtual video input

```
$ ecurl delete lt100:/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.

Example 4. PLAY

Live display of sdi-in video

```
$ ecurl play lt100:/0/sdi-in/0/data -d type=video/yuyv
```

Live listening of sdi-in audio

```
$ ecurl play lt100:/0/sdi-in/0/data -d type=audio/pcm
```

Live playing of sdi-in audio and video

```
$ ecurl play lt100:/0/sdi-in/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.

Example 5. REC

Record sdi-in video into a mp4 file

```
$ ecurl rec lt100:/0/sdi-in/0/file -d type=video/mp4
```

Chapter 4. API description

An API endpoint is a URL where the API receives requests about a specific resource. The endpoints are accessed with URLs with the following syntax `scheme:/path`.

It comprises:

- A non-empty scheme component followed by a colon (`lt100:`).
- A path component consisting of a sequence of path segments separated by a slash (`/`).

For convenience, the URLs endpoints are described in tables where the scheme is omitted (`lt100:`) and the vertical separators replace the path slash (`/`). Paths are read from left to right. Methods written into a cell show the path available methods. An empty cell means that no method exists on the path.

4.1. Agent

The **agent** endpoint allows to retrieve the **lt100agent** software version.

/
`GET`

```
GET     /
```

4.1.1. Agent Object

revision `string`

VCS hash.

time `string`

VCS time.

version `string`

VCS tagged version.

Agent object

```
{
    "revision": "...hash...",
    "time": "...time...",
    "version": "3.2.1"
}
```

4.1.2. View lt100agent Information

Retrieves the **lt100agent** software version.

Parameters

None.

Response

Returns the agent object if the request succeeded.

GET /

request

```
{
    "method": "GET",
    "url": "lt100:/",
    "body": null
}
```

response

```
{
    "revision": "...hash...",
    "time": "...time...",
    "version": "3.2.1"
}
```

Examples

▼ *ecurl*

```
$ ecurl get lt100:/
```

▼ *GO*

```
var response lt.Agent // struct to store the response
err := lt.Get("lt100:/", &response)
```

▼ *C++*

```
lt::Agent response; // struct to store the response
lt::error err = lt::Get("lt100:/", response);
```

4.2. Board

The **board** endpoint allows to retrieve information on boards installed into the host.

```
:board

GET
```

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

4.2.1. Board Object

model `string`

Board model identifier. Could be `lt101`, `lt102`, `lt122` or `lt124`. If no devices is found, the value is left empty.

sn `uint`

Board serial number.

cpu `uint`

Embedded processing cpu tagged time.

fpga `uint`

Processing fpga tagged time.

bridge `uint`

Bridge fpga tagged time.

Board object

```
{
    "model" : "lt101",
    "sn" : 64000000,
    "cpu" : 0,
    "fpga" : 0,
    "bridge" : 0
}
```

4.2.2. 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": "lt100:/0",
    "body": null
}
```

response

```
{
    "model" : "lt101",
    "sn" : 64000000,
    "cpu" : 0,
    "fpga" : 0,
    "bridge" : 0
}
```

Examples

▼ *ecurl*

```
$ ecurl get lt100:/0
```

▼ *GO*

```
var response lt.Board // struct to store the response
err := lt.Get("lt100:/0", &response)
```

▼ *C++*

```
lt::Board response; // object to store the response
lt::error err = lt::Get("lt100:/0", response);
```

4.3. CVBS Input

This endpoint describes how to use **cvbs-in** inputs.

Native data can be accessed via the **format** path enumerator `yuyv` (video) and `pcm` (audio).

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

Some of the proposed **formats** might require to use the host CPU and/or GPU before being delivered.

`:board`	`cvbs-in`	`:id`	`data`
`GET`		`GET`	`POST`
			`file`
			`POST`
			`net`
			`POST`

`GET`	`/:board/cvbs-in/:id`
`POST`	`/:board/cvbs-in/:id/data`
`POST`	`/:board/cvbs-in/:id/file`
`POST`	`/:board/cvbs-in/:id/net`

`board`

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

`id`

cvbs-in index number `[0]`.

4.3.1. CVBS Input 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).

CVBS Input 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).

4.3.2. View CVBS Input Status

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

Parameters

None.

Response

Returns the cvbs-in object if the request succeeded.

GET /:board/cvbs-in/:id

request

```
{
    "method": "GET",
    "url": "lt100:/0/cvbs-in/0",
    "body": null
}
```

response

```
{
    "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 lt100:/0/cvbs-in/0
```

▼ *GO*

```
var response lt.Input // struct to store the response
err := lt.Get("lt100:/0/cvbs-in/0", &response)
```

▼ *C++*

```
lt::Input response; // struct to store the response
lt::error err = lt::Get("lt100:/0/cvbs-in/0", response);
```

4.3.3. CVBS Input from the command line

Please use our dedicated tool **ecurl** to access or test the cvbs-in capabilities from the command line.

These `ecurl` samples are provided for convenience and are not exhaustive. If you want to learn more

about LT API programming, please download our SDK and look at the provided examples.

View the cvbs-in signal status

```
$ ecurl get lt100:/0/cvbs-in/0
```

Play audio (PCM)

```
$ ecurl play lt100:/0/cvbs-in/0/data -d media=audio/pcm
```

Play video (YUYV)

```
$ ecurl play lt100:/0/cvbs-in/0 -d media=video/yuyv
```

Play both audio and video

```
$ ecurl play lt100:/0/cvbs-in/0
```

Make a JPEG capture

```
$ ecurl rec lt100:/0/cvbs-in/0 -d media=image/jpeg
```

Make a PNG capture

```
$ ecurl rec lt100:/0/cvbs-in/0 -d media=image/png
```

Record a movie clip

```
$ ecurl rec lt100:/0/cvbs-in/0
```

4.4. SVIDEO Input

This endpoint describes how to use **svideo-in** inputs.

Native data can be accessed via the **format** path enumerator `yuyv` (video) and `pcm` (audio).

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

Some of the proposed **formats** might require to use the host CPU and/or GPU before being delivered.

:board	svideo-in	:id	data
GET		GET	POST
			file
			POST
			net
			POST

GET	/:board/svideo-in/:id	
POST	/:board/svideo-in/:id/data	
POST	/:board/svideo-in/:id/file	
POST	/:board/svideo-in/:id/net	

board

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

id

svideo-in index number [0].

4.4.1. SVIDEO Input 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).

SVIDEO Input 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).

4.4.2. View SVIDEO Input Status

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

Parameters

None.

Response

Returns the svideo-in object if the request succeeded.

GET /:board/svideo-in/:id

request

```
{
    "method": "GET",
    "url": "lt100:/0/svideo-in/0",
    "body": null
}
```

response

```
{
    "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 lt100:/0/svideo-in/0
```

▼ *GO*

```
var response lt.Input // struct to store the response
err := lt.Get("lt100:/0/svideo-in/0", &response)
```

▼ *C++*

```
lt::Input response; // struct to store the response
lt::error err = lt::Get("lt100:/0/svideo-in/0", response);
```

4.4.3. SVIDEO Input from the command line

Please use our dedicated tool **ecurl** to access or test the svideo-in capabilities from the command line.

These `ecurl` samples are provided for convenience and are not exhaustive. If you want to learn more

about LT API programming, please download our SDK and look at the provided examples.

View the svideo-in signal status

```
$ ecurl get lt100:/0/svideo-in/0
```

Play audio (PCM)

```
$ ecurl play lt100:/0/svideo-in/0/data -d media=audio/pcm
```

Play video (YUYV)

```
$ ecurl play lt100:/0/svideo-in/0 -d media=video/yuyv
```

Play both audio and video

```
$ ecurl play lt100:/0/svideo-in/0
```

Make a JPEG capture

```
$ ecurl rec lt100:/0/svideo-in/0 -d media=image/jpeg
```

Make a PNG capture

```
$ ecurl rec lt100:/0/svideo-in/0 -d media=image/png
```

Record a movie clip

```
$ ecurl rec lt100:/0/svideo-in/0
```

4.5. DVI Input

This endpoint describes how to use **dvi-in** inputs.

Native data can be accessed via the **format** path enumerator `yuyv` (video) and `pcm` (audio).

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

Some of the proposed **formats** might require to use the host CPU and/or GPU before being delivered.

:board	dvi-in	:id	data
GET		GET	POST
			file
			POST
			net
			POST

```
GET     /:board/dvi-in/:id
POST    /:board/dvi-in/:id/data
POST    /:board/dvi-in/:id/file
POST    /:board/dvi-in/:id/net
```

board

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

id

dvi-in index number [0 .. 1].

4.5.1. DVI Input 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).

DVI Input 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).

4.5.2. View DVI Input Status

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

Parameters

None.

Response

Returns the dvi-in object if the request succeeded.

GET /:board/dvi-in/:id

request

```
{
    "method": "GET",
    "url": "lt100:/0/dvi-in/0",
    "body": null
}
```

response

```
{
    "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 lt100:/0/dvi-in/0
```

▼ *GO*

```
var response lt.Input // struct to store the response
err := lt.Get("lt100:/0/dvi-in/0", &response)
```

▼ *C++*

```
lt::Input response; // struct to store the response
lt::error err = lt::Get("lt100:/0/dvi-in/0", response);
```

4.5.3. DVI Input from the command line

Please use our dedicated tool **ecurl** to access or test the dvi-in capabilities from the command line.

These `ecurl` samples are provided for convenience and are not exhaustive. If you want to learn more

about LT API programming, please download our SDK and look at the provided examples.

View the dvi-in signal status

```
$ ecurl get lt100:/0/dvi-in/0
```

Play audio (PCM)

```
$ ecurl play lt100:/0/dvi-in/0/data -d media=audio/pcm
```

Play video (YUYV)

```
$ ecurl play lt100:/0/dvi-in/0 -d media=video/yuyv
```

Play both audio and video

```
$ ecurl play lt100:/0/dvi-in/0
```

Make a JPEG capture

```
$ ecurl rec lt100:/0/dvi-in/0 -d media=image/jpeg
```

Make a PNG capture

```
$ ecurl rec lt100:/0/dvi-in/0 -d media=image/png
```

Record a movie clip

```
$ ecurl rec lt100:/0/dvi-in/0
```

4.6. SDI Input

This endpoint describes how to use **sdi-in** inputs.

Native data can be accessed via the **format** path enumerator `yuyv` (video) and `pcm` (audio).

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

Some of the proposed **formats** might require to use the host CPU and/or GPU before being delivered.

:board	sdi-in	:id	data
GET		GET	POST
			file
			POST
			net
			POST

GET	/:board/sdi-in/:id	
POST	/:board/sdi-in/:id/data	
POST	/:board/sdi-in/:id/file	
POST	/:board/sdi-in/:id/net	

board

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

id

sdi-in index number [0 .. 1].

4.6.1. SDI Input 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).

SDI Input 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).

4.6.2. View SDI Input Status

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

Parameters

None.

Response

Returns the sdi-in object if the request succeeded.

GET /:board/sdi-in/:id

request

```
{
    "method": "GET",
    "url": "lt100:/0/sdi-in/0",
    "body": null
}
```

response

```
{
    "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 lt100:/0/sdi-in/0
```

▼ *GO*

```
var response lt.Input // struct to store the response
err := lt.Get("lt100:/0/sdi-in/0", &response)
```

▼ *C++*

```
lt::Input response; // struct to store the response
lt::error err = lt::Get("lt100:/0/sdi-in/0", response);
```

4.6.3. SDI Input from the command line

Please use our dedicated tool **ecurl** to access or test the sdi-in capabilities from the command line.

These `ecurl` samples are provided for convenience and are not exhaustive. If you want to learn more

about LT API programming, please download our SDK and look at the provided examples.

View the sdi-in signal status

```
$ ecurl get lt100:/0/sdi-in/0
```

Play audio (PCM)

```
$ ecurl play lt100:/0/sdi-in/0/data -d media=audio/pcm
```

Play video (YUYV)

```
$ ecurl play lt100:/0/sdi-in/0 -d media=video/yuyv
```

Play both audio and video

```
$ ecurl play lt100:/0/sdi-in/0
```

Make a JPEG capture

```
$ ecurl rec lt100:/0/sdi-in/0 -d media=image/jpeg
```

Make a PNG capture

```
$ ecurl rec lt100:/0/sdi-in/0 -d media=image/png
```

Record a movie clip

```
$ ecurl rec lt100:/0/sdi-in/0
```

4.7. 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 GET DELETE	data POST
		file POST
		net POST

GET	/canvas/:id	**id**
POST	/canvas/:id/:format/data	canvas index number [0 .. 7].
POST	/canvas/:id/:format/file	
POST	/canvas/:id/:format/net	

Draw operations

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

GET	/canvas/:id	**id**
POST	/canvas/:id/init	canvas index number [0 .. 7].
POST	/canvas/:id/line	
POST	/canvas/:id/ellipse	
POST	/canvas/:id/rectangle	
POST	/canvas/:id/image	
POST	/canvas/:id/video	
POST	/canvas/:id/ops	

4.7.1. 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).

4.7.2. 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": "lt100:/canvas/0",
    "body": null
}
```

response

```
{
    "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 lt100:/canvas/0
```

▼ *GO*

```
var response lt.Input // struct to store the response
err := lt.Get("lt100:/canvas/0", &response)
```

▼ *C++*

```
lt::Input response; // struct to store the response
lt::error err = lt::Get("lt100:/canvas/0", response);
```

4.7.3. 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": "lt100:/canvas/0",
    "body": null
}
```

response

```
{

}
```

Examples

▼ *ecurl*

```
$ ecurl delete lt100:/canvas/0
```

▼ *GO*

```
err := lt.Delete("lt100:/canvas/0", nil, nil)
```

▼ *C++*

```
lt::error err = lt::Delete("lt100:/canvas/0", nullptr, nullptr);
```

4.7.4. 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": "lt100:/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 lt100:/canvas/0/init -d source=video.mp4
```

▼ *GO*

```
body := lt.JSON{
    "source": "video.mp4",
}
err := lt.Post("lt100:/canvas/0/init", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "video.mp4"}
};
lt::error err = lt::Post("lt100:/canvas/0/init", body, nullptr);
```

4.7.5. Text Operation

Draw text onto the canvas.

Parameters

op `string`

Operation identifier `text`.

text `string`

Text to draw.

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}**.

anchor `[2]float`

Move container along to the horizontal and

POST /canvas/:id/text

request

```
{
    "method": "POST",
    "url": "lt100:/canvas/0/text",
    "body": {
        "text": "hello world!",
    }
}
```

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]
}
```

Examples

▼ *ecurl*

```
$ ecurl post lt100:/canvas/0/text -d text="hello world!"
```

▼ *GO*

```
body := lt.JSON{
    "text": "hello world!",
}
err := lt.Post("lt100:/canvas/0/text", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"text", "hello world!"}
};
lt::error err = lt::Post("lt100:/canvas/0/text", body, nullptr);
```

4.7.6. 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}**.

anchor `[2]float`

Move container along to the horizontal and vertical **{x, y}** anchor in % of the container size **{width, height}**. Default is `{0,0}`.

Response

Returns the line operation parameters if the request succeeded.

POST /canvas/:id/line

request

```
{
    "method": "POST",
    "url": "lt100:/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 lt100:/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("lt100:/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("lt100:/canvas/0/line", body, nullptr);
```

4.7.7. 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}**.

anchor `[2]float`

Move container along to the horizontal and vertical **{x, y}** anchor in % of the container size **{width, height}**. Default is `{0,0}`.

Response

Returns the ellipse operation parameters if the request succeeded.

POST /canvas/:id/ellipse

request

```
{
    "method": "POST",
    "url": "lt100:/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]
}
```

Examples

▼ *ecurl*

```
$ ecurl post lt100:/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("lt100:/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("lt100:/canvas/0/ellipse", body, nullptr);
```

4.7.8. 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}**.

anchor `[2]float`

Move container along to the horizontal and vertical **{x, y}** anchor in % of the container size **{width, height}**. Default is `{0,0}`.

Response

Returns the rectangle operation parameters if the request succeeded.

POST /canvas/:id/rectangle

request

```
{
    "method": "POST",
    "url": "lt100:/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 lt100:/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("lt100:/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("lt100:/canvas/0/rectangle", body, nullptr);
```

4.7.9. 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}**.

anchor `[2]float`

Move container along to the horizontal and vertical **{x, y}** anchor in % of the container size **{width, height}**. Default is `{0,0}`.

---

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.

POST /canvas/:id/image

request

```
{
    "method": "POST",
    "url": "lt100:/canvas/0/image",
    "body": {
        "source": "image.png",
        "position": [0,0],
        "size": [640,480]
    }
}
```

response

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

Examples

▼ *ecurl*

```
$ ecurl post lt100:/canvas/0/image \
    -d source=image.png \
    -d position=0,0 \
    -d size=640,480
```

▼ *GO*

```
body := lt.JSON{
    "source": "image.png",
    "position": [0,0],
    "size": [640,480]
}
err := lt.Post("lt100:/canvas/0/image", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "image.png"},
    {"position", {0,0}},
    {"size", {640,480}}
};
lt::error err = lt::Post("lt100:/canvas/0/image", body, nullptr);
```

4.7.10. Video Operation

Place a video on the canvas.

Parameters

op `string`

Batch operation identifier `video`.

source `string`

Supported sources are `:board/sdi-in/:id`, `:board/hdmi-in/: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}**.

anchor `[2]float`

Move container along to the horizontal and vertical **{x, y}** anchor in % of the container size **{width, height}**. Default is `{0,0}`.

POST /canvas/:id/video

request

```
{
    "method": "POST",
    "url": "lt100:/canvas/0/video",
    "body": {
        "source": "0/sdi-in/0",
        "position": [0,0],
        "size": [1920,1080]
    }
}
```

response

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

Response

Returns the video operation parameters if the request succeeded.

Examples

▼ *ecurl*

```
$ ecurl post lt100:/canvas/0/video \
    -d source=0/sdi-in/0 \
    -d position=0,0 \
    -d size=1920,1080
```

▼ *GO*

```
body := lt.JSON{
    "source": "0/sdi-in/0",
    "position": [0,0],
    "size": [1920,1080]
}
err := lt.Post("lt100:/canvas/0/video", body, nil)
```

▼ *C++*

```
lt::json body = {
    {"source", "0/sdi-in/0"},
    {"position", {0,0}},
    {"size", {1920,1080}}
};
```

```
lt::error err = lt::Post("lt100:/canvas/0/video", body, nullptr);
```

4.7.11. 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 lt100:/canvas/0/ops \
    -d ops=@draw.json
```

response

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

4.7.12. Canvas from the command line

Please use our dedicated tool **ecurl** to access or test the canvas capabilities from the command line.

These `ecurl` samples are provided for convenience and are not exhaustive. If you want to learn more about LT API programming, please download our SDK and look at the provided examples.

View the canvas signal status

```
$ ecurl get lt100:/canvas/0
```

Play audio (PCM)

```
$ ecurl play lt100:/canvas/0/data -d media=audio/pcm
```

Play video (YUYV)

```
$ ecurl play lt100:/canvas/0 -d media=video/yuyv
```

Play both audio and video

```
$ ecurl play lt100:/canvas/0
```

Make a JPEG capture

```
$ ecurl rec lt100:/canvas/0 -d media=image/jpeg
```

Make a PNG capture

```
$ ecurl rec lt100:/canvas/0 -d media=image/png
```

Record a movie clip

```
$ ecurl rec lt100:/canvas/0
```

4.8. 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 **lt100agent** will automatically collect the resources and clean them.

<table>
<tr><td rowspan="4">client</td><td rowspan="3">jobs
DELETE</td><td rowspan="3">:id
GET DELETE</td><td>start
POST</td></tr>
<tr><td>stop
POST</td></tr>
<tr><td>pause
POST</td></tr>
<tr><td>refs</td><td colspan="2">:id
DELETE</td></tr>
</table>

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

id
: Object identifier.

4.8.1. 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 **lt100agent**.

GET /client/job/:id

Please go to Section 4.9, “Workers” to learn the complete workflow usage.

4.8.2. Terminate Worker

Terminate a worker task.

DELETE /client/job/:id

Please go to Section 4.9, “Workers” to learn the complete workflow usage.

4.8.3. Terminate ALL Workers

Terminate all the client workers tasks.

DELETE /client/job

Please go to Section 4.9, “Workers” to learn the complete workflow usage.

4.8.4. Release Referenced Memory

Clients and **lt100agent** communicate by exchanging references on shared memory blocks.

Once processed, it is recommended to expressly release the references, otherwise the **lt100agent** 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 4.9, “Workers” to learn the complete workflow usage.

4.9. 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
	net POST

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

`url`

URL can be any valid API resource that point toward a `data`, a `file` or a `net` 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.
- `net` this maps any data source to the **network adapter**. Not ready at this time. TBD

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

4.9.1. Worker Creation

A **type** has to be submitted to the **data**, **file** or **net** 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`.

4.9.1.1. 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/dvi-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`.

POST /:url/data

request

```
{
    "method": "POST",
    "url": "lt100:/:url/data",
    "body": {
        "media": "audio/pcm",
        "source": "0/dvi-in/0",
        "channels": 2,
        "samplerate": 48000,
        "depth": 16
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Response

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

Examples

▼ *GO*

```
err := lt.Post("lt100:/: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("lt100:/:url/data", lt::AudioDataWorker{ "audio/pcm" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
```

```
string workerURL = lt::RedirectLocation(err);
```

4.9.1.2. 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/dvi-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": "lt100:/:url/data",
    "body": {
        "media": "video/nv12",
        "source": "0/dvi-in/0",
        "size": [1920, 1080]
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Examples

▼ *GO*

```
err := lt.Post("lt100:/: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("lt100:/:url/data", lt::ImageDataWorker{ "image/jpeg" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

4.9.1.3. 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` and `video/mp4`.

source `string`

The audio board input source: `:board/dvi-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": "lt100:/:url/data",
    "body": {
        "media": "video/nv12",
        "source": "0/dvi-in/0",
        "size": [1920, 1080],
        "framerate": 30
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Examples

▼ *GO*

```
err := lt.Post("lt100:/: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("lt100:/:url/data", lt::VideoDataWorker{ "video/nv12" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

4.9.1.4. 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/dvi-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).

splitLength `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/data

request

```
{
    "method": "POST",
    "url": "lt100:/:url/data",
    "body": {
        "media": "audio/wav",
        "source": "0/dvi-in/0",
        "channels": 2,
        "samplerate": 48000,
        "depth": 16,
        "location": "/path/to/audio/directory",
        "duration": 0,
        "splitLength": 0,
        "splitDuration": 0
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Examples

▼ *GO*

```
err := lt.Post("lt100:/:url/data", 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("lt100:/:url/data", lt::AudioFileWorker{ "audio/wav" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

4.9.1.5. 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/dvi-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.

Response

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

POST /:url/data

request

```
{
    "method": "POST",
    "url": "lt100:/:url/data",
    "body": {
        "media": "video/nv12",
        "source": "0/dvi-in/0",
        "size": [1920, 1080],
        "location": "/path/to/image/directory"
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Examples

▼ *GO*

```
err := lt.Post("lt100:/:url/data", 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("lt100:/:url/data", lt::ImageFileWorker{ "image/jpeg" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

4.9.1.6. 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/bmp`, `video/h264` and `video/mp4`.

source `string`

The audio board input source: `:board/dvi-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 milliseconds to record. Default `0` (infinite).

splitLength `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/data

request

```
{
    "method": "POST",
    "url": "lt100:/:url/data",
    "body": {
        "media": "video/mp4",
        "source": "0/dvi-in/0",
        "size": [1920, 1080],
        "framerate": 30,
        "location": "/path/to/video/directory",
        "duration": 0,
        "splitLength": 0,
        "splitDuration": 0
    }
}
```

response

```
{
    "location": "lt100:/client/jobs/...",
    "error": "redirect"
}
```

Examples

▼ *GO*

```
err := lt.Post("lt100:/:url/data", 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("lt100:/:url/data", lt::VideoFileWorker{ "video/mp4" }, nullptr);
if (!lt::ErrorIs(err, lt::ErrRedirect)) {
    logFatal("worker creation failed:" + err);
}
string workerURL = lt::RedirectLocation(err);
```

4.9.1.7. Audio Net Worker

TBD

4.9.1.8. Image Net Worker

TBD

4.9.1.9. Video Net Worker

TBD

4.9.2. 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 ..."
        }
    }
}
```

4.9.3. 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
    },
}
```

4.9.3.1. 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": "lt100:/client/ref/...",
    "client": "q5jrzd2OIQuxCq1IJWICuA",
    "handle": "lt100_global_24",
    "size": 1275592704,
    "ptr": 478347264
}
```


4.9.3.2. 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,
}
```

4.9.3.3. Image Metadata

Packets with `image/*` type.

size `[2]int`

The image frame size.

Image metadata

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

4.9.3.4. 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
}
```

4.9.4. Data Worker Workflow

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

Create Worker

`> POST lt100:/canvas/0/yuyv/data`

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

`> GET workerURL`

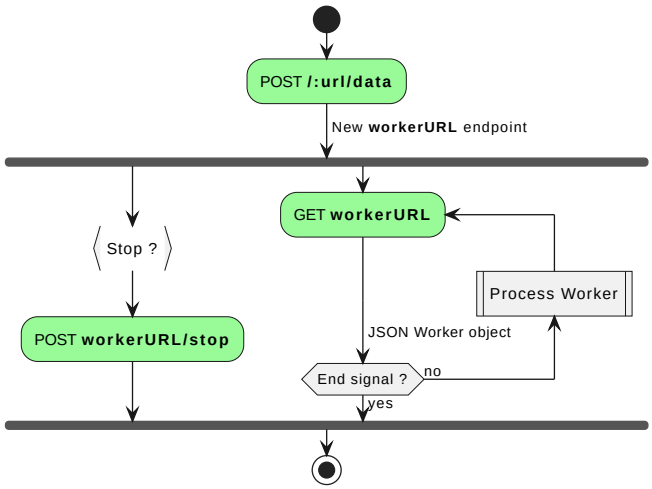
- JSON Worker object is returned.
- Check non-null error.

`> 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

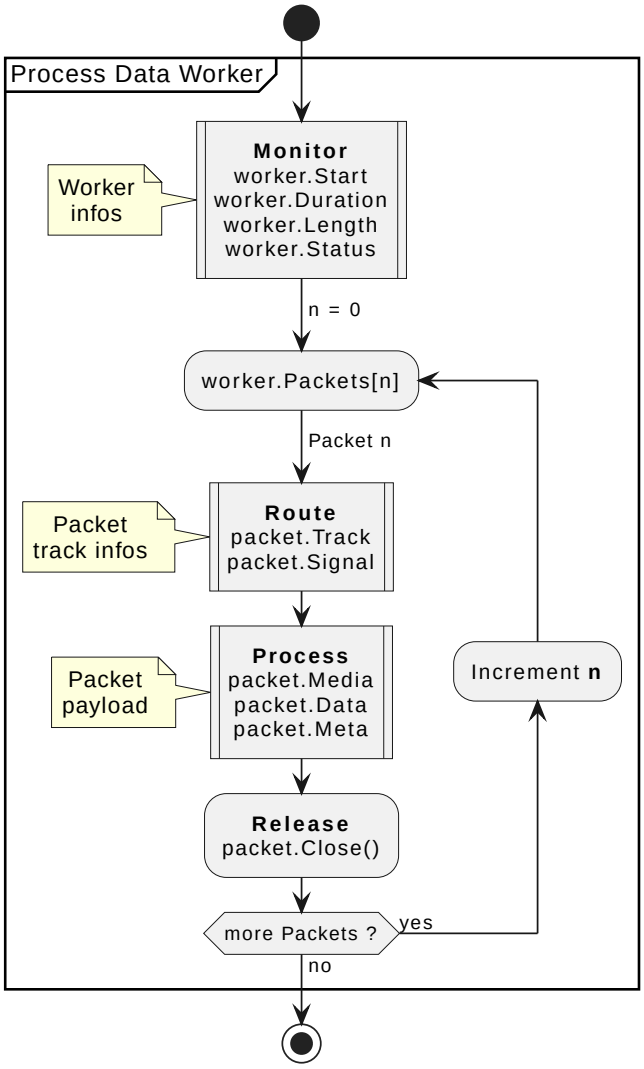
- Grab one packet.
- Check the packet track ID.
- Check the packet track signal.

> 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.



4.9.5. 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 lt100:/canvas/0/png/file`

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

`> GET worker`

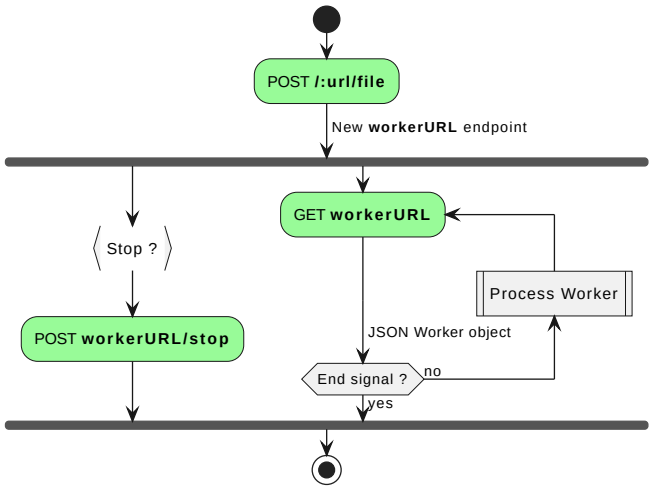
- JSON Worker object is returned.
- Check non-null error.

`> 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 lt100:/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
```

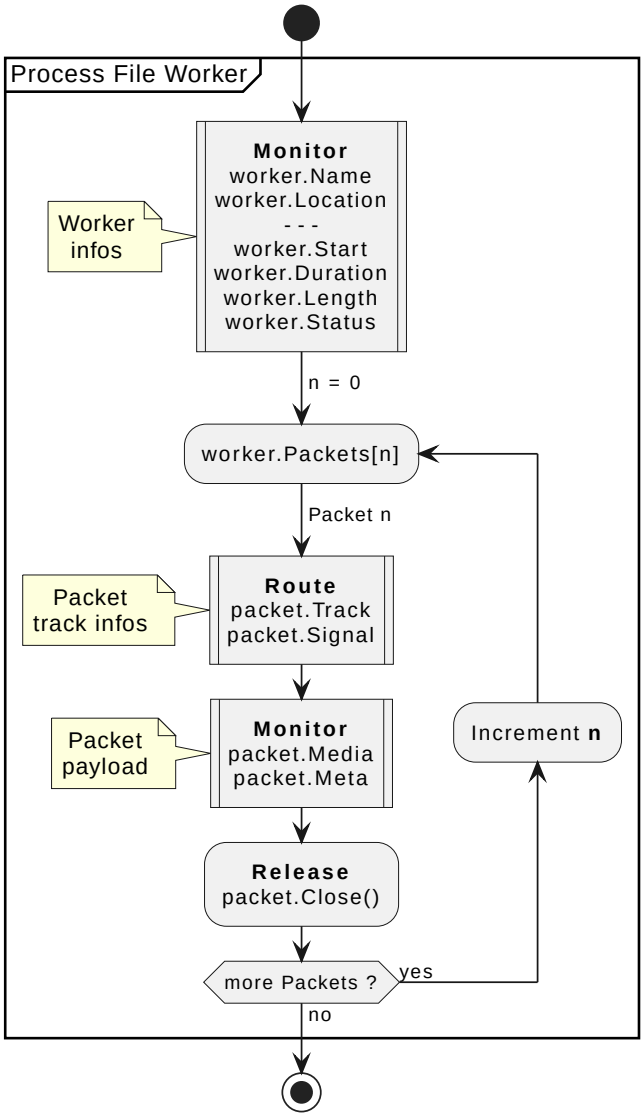
- Grab one packet.
- Check the packet track ID.
- Check the packet track signal.

```
> 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 5. Cheatsheet

/ GET

:board GET

:board GET	cvbs-in	:id GET	data POST
			file POST
			net POST

:board GET	svideo-in	:id GET	data POST
			file POST
			net POST

:board GET	dvi-in	:id GET	data POST
			file POST
			net POST

:board GET	sdi-in	:id GET	data POST
			file POST
			net POST

canvas	:id GET DELETE	data POST
		file POST
		net POST

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

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

Chapter 6. Changelog

```
# 3.2.1 (28/10/2024):

# 3.2.0 (17/10/2024):

# 3.1.0 (10/10/2024):

# 3.0.0 (17/09/2024):
```