VSourceV4L2 C++ library

v2.0.1

Table of contents

Overview

VSourceV4L2 C++ library provides video capture and video source (V4L2 compatible video sources) control function based on **V4L2 API**. The library supports [stepwise and discrete devices](). The library inherits interface from open source **VSource** interface class. **VSource.h** file contains data structures **VSourceParams** class, **VSourceCommand** enum, **VSourceParam** enum and includes **VSource** class declaration. **VSourceParams** class contains video source params and methods to encode and decode video source params. **VSourceCommands** enum contains IDs of commands supported by **VSource** class. **VSourceParam** enum contains IDs of params supported by **VSource** class. **VSource** class depends on libraries: open source **Frame** library (contains video frame structure and pixel formats) and open source **ConfigReader** library (provides methods to work with JSON structures (read/write)). Also **VSourceV4L2** depends on open source **Logger** library which provides method to write logs. The library provides auto detection of supported resolution, format and fps. If particular devices doesn't support requested pixel format, resolution or fps, the library will set most appropriate parameters.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	13.01.2022	First version
1.1.0	20.06.2023	- Fixed bugs. - Documentation updated.
1.2.0	29.06.2023	- Added new parameters. - Documentation updated.
2.0.0	20.08.2023	- Added license. - Added example and updated test application. - Interface update according to **VSource** class. - Added automatic detection supported format and automatic choosing right format during initialization.
2.0.1	29.08.2023	- Fixed stepwise camera support issue. - Documentation updated.

VSourceV4L2 class description

VSourceV4L2 class declaration

VSourceV4l2 class declared in **VSourceV4L2.h** file. Class declaration:

```cpp
class VSourceV4L2 : public VSource
{
public:
    /**
     * @brief Class constructor.
     */
    VSourceV4L2();
    /**
     * @brief Class destructor.
     */
    ~VSourceV4L2();
    /**
     * @brief Get string of current library version.
     * @return String of current library version.
     */
    static std::string getVersion();
    /**
     * @brief Open video source. All params will be set by default.
     * @param initString Init string. Format: <device>;<width>;<height>;<fps>
     * <device> example "/dev/video0"
     * <width> optional. Can be 0 for auto detection.
     * <height> optional. Can be 0 for auto detection.
     * <fps> optional. Can be 0 for auto detection.
     * Example: "/dev/video0;640;480;30" or "/dev/video0;640;480" or
     * "/dev/video0;0;0;0", etc.
     * @return TRUE if the video source open or FALSE if not.
     */
    bool openVSource(std::string& initString) override;
    /**
     * @brief Init video source. All params will be set according to structure.
     * @param params Video source parameters structure.
     * @return TRUE if the video source init or FALSE if not.
     */
    bool initVSource(VSourceParams& params) override;
    /**
     * @brief Get open status.
     * @return TRUE if video source open or FALSE if not.
     */
    bool isVSourceOpen() override;
    /**
     * @brief Close video source.
     */
    void closeVSource() override;
    /**
     * @brief Get new video frame.
     * @param frame Frame object to copy new data.
     * @param timeoutMsec Timeout to wait new frame data:
     * timeoutMsec == -1 - Method will wait endlessly until new data arrive.
     * timeoutMsec == 0  - Method will only check if new data exist.
     * timeoutMsec > 0   - Method will wait new data specified time.
     * @return TRUE if new video frame exist and copied or FALSE if not.
```

```cpp
        */
    bool getFrame(Frame& frame, int32_t timeoutMsec = 0) override;
    /**
     * @brief Set video source param.
     * @param id Parameter ID.
     * @param value Parameter value to set.
     * @return TRUE if property was set of FALSE.
     */
    bool setParam(VSourceParam id, float value) override;
    /**
     * @brief Get video source param value.
     * @param id Parameter ID.
     * @return Parameter value or -1.
     */
    float getParam(VSourceParam id) override;
    /**
     * @brief Get video source params structure.
     * @return Video source parameters structure.
     */
    VSourceParams getParams() override;
    /**
     * @brief Execute command.
     * @param id Command ID.
     * @return TRUE if the command accepted or FALSE if not.
     */
    bool executeCommand(VSourceCommand id) override;
};
```

getVersion method

getVersion() method returns string of current version of **VSourceV4L2** class. Method declaration:

```cpp
static std::string getVersion();
```

Method can be used without **VSourceV4L2** class instance:

```cpp
cout << "VSourceV4L2 class version: " << VSourceV4L2::getVersion() << endl;
```

Console output:

```
VSourceV4L2 class version: 2.0.1
```

openVSource method

openVSource(...) method initializes video source. Instead of **openVSource(...)** method user can call **initVSource(...)**. Method declaration:

```cpp
bool openVSource(std::string& initString) override;
```

Parameter	Value
initString	Initialization string. Valid formats: **[full device name];[width];[height];[fps];[fourcc]** or **[full device name];[width];[height];[fps]** or **[full device name];[width];[height]** or **[full device name]** Initialization string parameters: **[full device name]** - Full name of video device in Linux OS. Default name format: /dev/video0, /dev/video1 etc. or can be /dev/v4l/by-id/... in case UVC compatible cameras. **[width]** - Video frame width. Video frame height must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[height]** - Video frame height. Video frame width must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[fps]** - FPS. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters. **[fourcc]** - Pixel format. Valid values: BGR24, RGB24, GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12. If fourcc not set the library will chose appropriate format according to existing video device parameters.

Returns: TRUE if the video source open or FALSE if not.

initVSource method

initVSource(...) method designed to initialize video source by set of parameters. Instead of **initVSource(...)** method user can call **openVSource(...)**. Method declaration:

```
bool initVSource(VSourceParams& params) override;
```

Parameter	Value
params	VSourceParams structure (see **VSourceParams** class description). The video source should set parameters according to params structure.

Returns: TRUE if the video source initialized or FALSE if not.

isVSourceOpen method

isVSourceOpen() method returns video source initialization status. Initialization status also included in **VSourceParams** class. Method declaration:

```
bool isVSourceOpen() override;
```

Returns: TRUE if the video source open (initialized) or FALSE if not.

closeVSource method

closeVSource() method intended to close video source. Method declaration:

```cpp
void closeVSource() override;
```

getFrame method

getFrame(...) method intended to get input video frame. Video source should support auto reinitialization in case connection loss. Method declaration:

```cpp
bool getFrame(Frame& frame, int32_t timeoutMsec = 0) override;
```

Parameter	Value
frame	Output video frame (see **Frame** class description). Video source class determines output pixel format. Pixel format can be set in **initVSource(...)** or **openVSource(...)** methods but if specified pixel format not supported by hardware the library can return Frame object with different pixel format.
timeoutMsec	Timeout to wait new frame data: - timeoutMs == -1 - Method will wait endlessly until new data arrive. - timeoutMs == 0 - Method will only check if new data exist. - timeoutMs > 0 - Method will wait new data specified time.

Returns: TRUE if new data exists and copied or FALSE if not.

setParam method

setParam(...) method designed to set new video source parameters value. Method will set parameters to device. Method can be used only after video source initialization. Method declaration:

```cpp
bool setParam(VSourceParam id, float value) override;
```

Parameter	Description
id	Video source parameter ID according to **VSourceParam** enum.
value	Video source parameter value.

Returns: TRUE is the parameter was set or FALSE if not.

getParam method

getParam(...) method designed to obtain video source parameter value. Method will request parameters from device. Method can be used only after video source initialization. Method declaration:

```
float getParam(VSourceParam id) override;
```

Parameter	Description
id	Video source parameter ID according to **VSourceParam** enum.

Returns: parameter value or -1 of the parameter not supported.

getParams method

getParams(...) method designed to obtain video source params structures. Method declaration:

```
VSourceParams getParams() override;
```

Returns: video source parameters structure (see **VSourceParams** class description for supported parameters).

executeCommand method

executeCommand(...) method designed to execute video source command. Method declaration:

```
bool executeCommand(VSourceCommand id) override;
```

Parameter	Description
id	Video source command ID according to **VSourceCommand** enum.

Returns: TRUE is the command was executed or FALSE if not.

encodeSetParamCommand method of VSource interface class

encodeSetParamCommand(...) static method of **VSource** interface class designed to encode command to change any parameter for remote video source. To control video source remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VSource** class contains static methods for encoding the control command. The **VSource** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeSetParamCommand(...)** designed to encode SET_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, VSourceParam id, float
value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Parameter ID according to **VSourceParam** enum.
value	Parameter value.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	0x01	Major version of VSource class.
2	0x03	Minor version of VSource class.
3	id	Parameter ID **int32_t** in Little-endian format.
4	id	Parameter ID **int32_t** in Little-endian format.
5	id	Parameter ID **int32_t** in Little-endian format.
6	id	Parameter ID **int32_t** in Little-endian format.
7	value	Parameter value **float** in Little-endian format.
8	value	Parameter value **float** in Little-endian format.
9	value	Parameter value **float** in Little-endian format.
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without **VSource** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = (float)(rand() % 20);
// Encode command.
VSurce::encodeSetParamCommand(data, size, VSourceParam::EXPOSURE, outValue);
```

encodeCommand method of VSource interface class

encodeCommand(...) static method of **[VSource](#)** interface class designed to encode command for remote video source. To control a video source remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VSource** class contains static methods for encoding the control command. The **VSource** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND (action command). Method declaration:

```cpp
static void encodeCommand(uint8_t* data, int& size, VSourceCommand id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Command ID according to **VSourceParam** enum.

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	0x01	Major version of VSource class.
2	0x03	Minor version of VSource class.
3	id	Command ID **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.

encodeCommand(...) is static and used without **VSource** class instance. This method used on client side (control system). Command encoding example:

```cpp
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Encode command.
VSource::encodeCommand(data, size, VSourceCommand::RESTART);
```

decodeCommand method of VSource interface class

decodeCommand(...) static method of **VSource** interface class designed to decode command on video source side (edge device). Method declaration:

```
static int decodeCommand(uint8_t* data, int size, VSourceParam& paramId, VSourceCommand&
commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Should be 11 bytes.
paramId	Parameter ID according to **VSourceParam** enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	Command ID according to **VSourceCommand** enum. After decoding COMMAND the method will return command ID.
value	Parameter value (after decoding SET_PARAM command).

Returns: 0 - in case decoding COMMAND, **1** - in case decoding SET_PARAM command or **-1** in case errors.

Data structures

VSource.h file of **VSource** interface class defines IDs for parameters (**VSourceParam** enum) and IDs for commands (**VSourceCommand** enum).

VSourceCommand enum

Enum declaration:

```
enum class VSourceCommand
{
    /// Restart.
    RESTART = 1
};
```

Table 2 - Video source commands description. Some commands maybe unsupported by particular video source class.

Command	Description
RESTART	Not supported by VSourceV4L2 library.

VSourceParam enum

Enum declaration:

```cpp
enum class VSourceParam
{
    /// [read/write] Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal, 3 - File and terminal.
    LOG_LEVEL = 1,
    /// [read/write] Frame width. User can set frame width before initialization
    /// or after. Some video source classes may set width automatically.
    WIDTH,
    /// [read/write] Frame height. User can set frame height before
    /// initialization or after. Some video source classes may set height
    /// automatically.
    HEIGHT,
    /// [read/write] Gain mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    GAIN_MODE,
    /// [read/write] Gain value. Value: 0(min for particular video source class)
    /// - 65535(max for particular video source class).
    GAIN,
    /// [read/write] Exposure mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    EXPOSURE_MODE,
    /// [read/write] Exposure value. Value: 0(min for particular video source
    /// class) - 65535(max for particular video source class).
    EXPOSURE,
    /// [read/write] Focus mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    FOCUS_MODE,
    /// [read/write] Focus position. Value: 0(full near) - 65535(full far).
    FOCUS_POS,
    /// [read only] Video capture cycle time. **VSource** class sets this value
    /// automatically. This parameter means time interval between two captured
    /// video frame.
    CYCLE_TIME_MKS,
    /// [read/write] FPS. User can set frame FPS before initialization or after.
    /// Some video source classes may set FPS automatically.
    FPS,
    /// [read only] Open flag. 0 - not open, 1 - open.
    IS_OPEN,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_1,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_2,
    /// [read/write] Custom parameter. Depends on implementation.
    CUSTOM_3
};
```

Table 3 - Video source params description. Some params maybe unsupported by particular video source class.

Parameter	Access	Description
LOG_LEVEL	read / write	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
WIDTH	read / write	Read only. Can't be changes after initialization.
HEIGHT	read / write	Read only. Can't be changes after initialization.
GAIN_MODE	read / write	Gain mode. Values: 0 - Manual control, 1 - Auto. Some V4l2 video source may not support this parameter to control.
GAIN	read / write	Gain value. Value: 0(min) - 65535(max). Some V4l2 video source may not support this parameter to control.
EXPOSURE_MODE	read / write	Exposure mode. Values: 0 - Manual mode, 1 - Auto mode, 2 - Shutter priority, 3 - Aperture priority. Some V4l2 video source may not support this parameter to control.
EXPOSURE	read / write	Exposure value. Value: 0(min) - 65535(max). Some V4l2 video source may not support this parameter to control.
FOCUS_MODE	read / write	Focus mode. Values: 0 - Manual control, 1 - Auto. Some V4l2 video source may not support this parameter to control.
FOCUS_POS	read / write	Focus position. Value: 0(full near) - 65535(full far). Some V4l2 video source may not support this parameter to control.
CYCLE_TIME_MKS	read only	Video capture cycle time, microseconds. The library class sets this value automatically. This parameter means time interval between two captured video frame.
FPS	read / write	FPS. Can't be changes after initialization.
IS_OPEN	read only	Open flag. 0 - not open, 1 - open.
CUSTOM_1	read / write	No supported in VSourceV4L2 library. Default value -1.
CUSTOM_2	read / write	No supported in VSourceV4L2 library. Default value -1.
CUSTOM_3	read / write	No supported in VSourceV4L2 library. Default value -1.

VSourceParams class description

VSourceParams class declaration

VSourceParams class used for video source initialization (**initVSource(...)** method) or to get all actual params (**getParams()** method). Also **VSourceParams** provide structure to write/read params from JSON files (**JSON_READABLE** macro, see **ConfigReader** class description) and provide methos to encode and decode params. Class declaration:

```cpp
class VSourceParams
{
public:
    /// Logging mode. Values: 0 - Disable, 1 - Only file,
    /// 2 - Only terminal, 3 - File and terminal.
    int logLevel{0};
    /// Video source: file, video stream, video device, camera num, etc.
    std::string source{"/dev/video0"};
    /// FOURCC: RGB24, BGR24, YUYV, UYVY, GRAY, YUV24, NV12, NV21, YU12, YV12.
    /// Value says to video source class which pixel format preferable for
    /// output video frame. Particular video source class can ignore this params
    /// during initialization. Parameters should be set before initialization.
    std::string fourcc{"YUYV"};
    /// Frame width. User can set frame width before initialization
    /// or after. Some video source classes may set width automatically.
    int width{1920};
    /// Frame height. User can set frame height before
    /// initialization or after. Some video source classes may set height
    /// automatically.
    int height{1080};
    /// Gain mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int gainMode{1};
    /// Gain value. Value: 0(min for particular video source class)
    /// - 65535(max for particular video source class).
    int gain{0};
    /// Exposure mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int exposureMode{1};
    /// Exposure value. Value: 0(min for particular video source
    /// class) - 65535(max for particular video source class).
    int exposure{1};
    /// Focus mode. Focus mode. Value depends on implementation but it is
    /// recommended to keep default values: 0 - Manual control, 1 - Auto.
    int focusMode{1};
    /// Focus position. Value: 0(full near) - 65535(full far).
    int focusPos{0};
    /// Video capture cycle time. **VSource** class sets this value
    /// automatically. This parameter means time interval between two captured
    /// video frame.
    int cycleTimeMks{0};
```

```cpp
    /// FPS. User can set frame FPS before initialization or after.
    /// Some video source classes may set FPS automatically.
    float fps{0};
    /// Open flag. 0 - not open, 1 - open.
    bool isOpen{false};
    /// Custom parameter. Depends on implementation.
    float custom1{0.0f};
    /// Custom parameter. Depends on implementation.
    float custom2{0.0f};
    /// Custom parameter. Depends on implementation.
    float custom3{0.0f};

    JSON_READABLE(VSourceParams, logLevel, source, fourcc, width,
                  height, gainMode, gain, exposureMode, exposure,
                  focusMode, focusPos, fps, custom1, custom2, custom3);

    /**
     * @brief operator =
     * @param src Source object.
     * @return VSourceParams obect.
     */
    VSourceParams& operator= (const VSourceParams& src);
    /**
     * @brief Encode params. The method doesn't encode params:
     * source and fourcc fields.
     * @param data Pointer to data buffer.
     * @param size Size of data.
     */
    void encode(uint8_t* data, int& size);
    /**
     * @brief Decode params. The method doesn't decode params:
     * source and fourcc fields.
     * @param data Pointer to data.
     * @return TRUE is params decoded or FALSE if not.
     */
    bool decode(uint8_t* data);
};
```

Table 4 - VSourceParams class fields description.

Field	type	Description
logLevel	int	Logging mode. Values: 0 - Disable, 1 - Only file, 2 - Only terminal, 3 - File and terminal.
source	string	Full name of video device in Linux OS. Default name format: /dev/video0, /dev/video1 etc. or can be /dev/v4l/by-id/... in case UVC compatible cameras.
fourcc	string	Pixel format. Valid values: BGR24, RGB24, GRAY, YUV24, YUYV, UYVY, NV12, NV21, YV12, YU12. If fourcc not set the library will chose appropriate format according to existing video device parameters. Value says to video source class which pixel format preferable for output video frame.

Field	type	Description
width	int	Video frame width. Video frame height must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters.
height	int	Video frame height. Video frame width must be set as well. The library will try set this value in video capture hardware parameters. If set to **0** the library will detect frame width automatically according to existing video device parameters.
gainMode	int	Gain mode. Values: 0 - Manual control, 1 - Auto. Some V4l2 video source may not support this parameter to control.
gain	int	Gain value. Value: 0(min) - 65535(max). Some V4l2 video source may not support this parameter to control.
exposureMode	int	Exposure mode. Values: 0 - Manual mode, 1 - Auto mode, 2 - Shutter priority, 3 - Aperture priority. Some V4l2 video source may not support this parameter to control.
exposure	int	Exposure value. Value: 0(min) - 65535(max). Some V4l2 video source may not support this parameter to control.
focusMode	int	Focus mode. Values: 0 - Manual control, 1 - Auto. Some V4l2 video source may not support this parameter to control.
focusPos	int	Focus position. Value: 0(full near) - 65535(full far). Some V4l2 video source may not support this parameter to control.
cycleTimeMks	int	Video capture cycle time. The library class sets this value automatically. This parameter means time interval between two captured video frame.
fps	float	FPS. User can set frame FPS only before initialization.
isOpen	bool	Open flag. false - not open, true - open.
custom1	float	No supported in VSourceV4L2 library. Default value -1.
custom2	float	No supported in VSourceV4L2 library. Default value -1.
custom3	float	No supported in VSourceV4L2 library. Default value -1.

None: *VSourceParams class fields listed in Table 4 reflects params set/get by methods setParam(...) and getParam(...).*

Encode video source params

VSourceParams class provides method **encode(...)** to serialize video source params (fields of VSourceParams class, see Table 4). Serialization of video source params necessary in case when you need to send video source params via communication channels. Method doesn't encode fields: **initString** and **fourcc**. Method declaration:

```cpp
void encode(uint8_t* data, int& size);
```

Parameter	Value
data	Pointer to data buffer. Buffer size should be at least **43** bytes.
size	Size of encoded data. 43 bytes by default.

Example:

```cpp
// Prepare random params.
VSourceParams in;
in.initString = "alsfghljb";
in.fourcc = "skdfjhvk";
in.logLevel = "dsglbjlkfjwjgre";
in.cycleTimeMks = rand() % 255;
in.exposure = rand() % 255;
in.exposureMode = rand() % 255;
in.gainMode = rand() % 255;
in.gain = rand() % 255;
in.focusMode = rand() % 255;
in.focusPos = rand() % 255;
in.fps = rand() % 255;
in.width = rand() % 255;
in.height = rand() % 255;
in.isOpen = true;

// Encode data.
uint8_t data[1024];
int size = 0;
in.encode(data, size);

cout << "Encoded data size: " << size << " bytes" << endl;
```

Decode video source params

VSourceParams class provides method **decode(...)** to deserialize video source params (fields of VSourceParams class, see Table 4). Deserialization of video source params necessary in case when you need to receive video source params via communication channels. Method doesn't decode fields: **initString** and **fourcc**. Method declaration:

```cpp
bool decode(uint8_t* data);
```

Parameter	Value
data	Pointer to encode data buffer. Data size should be at least **43** bytes.

Returns: TRUE if data decoded (deserialized) or FALSE if not.

Example:

```cpp
// Encode data.
VSourceParams in;
uint8_t data[1024];
int size = 0;
in.encode(data, size);

cout << "Encoded data size: " << size << " bytes" << endl;

// Decode data.
VSourceParams out;
if (!out.decode(data))
    cout << "Can't decode data" << endl;
```

Read params from JSON file and write to JSON file

VSource library depends on **ConfigReader** library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```cpp
// Write params to file.
VSurceParams in;
cr::utils::ConfigReader inConfig;
inConfig.set(in, "vSourceParams");
inConfig.writeToFile("TestVSourceParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if(!outConfig.readFromFile("TestVSourceParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}
```

TestVSourceParams.json will look like:

```json
{
    "vSourceParams": {
        "custom1": 96.0,
        "custom2": 212.0,
        "custom3": 243.0,
        "exposure": 236,
        "exposureMode": 63,
        "focusMode": 167,
        "focusPos": 34,
        "fourcc": "skdfjhvk",
        "fps": 205.0,
        "gain": 22,
        "gainMode": 84,
        "height": 143,
        "initString": "alsfghljb",
        "logLevel": 92,
        "width": 204
    }
```

```
    }
```

Build and connect to your project

Typical commands to build **VSource** library:

```
git clone https://github.com/ConstantRobotics-Ltd/VSourceV4L2.git
cd VSourceV4L2
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **VSourceV4L2** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **VSourceV4L2** as submodule by commands:

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/VSourceV4L2.git
3rdparty/VSourceV4L2
git submodule update --init --recursive
```

In you repository folder will be created folder **3rdparty/VSourceV4L2** which contains files of **VSourceV4L2** repository with subrepositories **Frame** and **ConfigReader**. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    VSourceV4L2
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

################################################################################
## 3RD-PARTY
## dependencies for the project
################################################################################
project(3rdparty LANGUAGES CXX)
```

```
################################################################
## SETTINGS
## basic 3rd-party settings before use
################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

################################################################
## CONFIGURATION
## 3rd-party submodules configuration
################################################################
SET(${PARENT}_SUBMODULE_VSOURCE_V4L2                 ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_VSOURCE_V4L2)
    SET(${PARENT}_VSOURCE_V4L2                       ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_V4L2_TEST                  OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_VSOURCE_V4L2_EXAMPLE               OFF CACHE BOOL "" FORCE)
endif()

################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################
if (${PARENT}_SUBMODULE_VSOURCE_V4L2)
    add_subdirectory(VSourceV4L2)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **VSourceV4L2** to your project and excludes test application (VSourceV4L2 class test applications) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    VSourceV4L2
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include VSourceV4L2 library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} VSourceV4L2)
```

Done!

Simple example

Example shows how to initialize video source with generic VSource interface and how to capture video in loop. Also example show how to get parameter value (video capture cycle time). Example:

```cpp
#include <iostream>
#include "VSourceV4L2.h"

// Entry point.
int main(void)
{
    // Video source init string.
    std::string initString = "/dev/video0;640;480;YUYV;30";

    // Init video source.
    cr::video::VSource* source = new cr::video::VSourceV4L2();
    if (!source->openVSource(initString))
        return -1;

    // Init Frames.
    cr::video::Frame frame;

    // Main loop.
    while (true)
    {
        // Wait new frame 1 sec.
        if (!source->getFrame(frame, 1000))
        {
            std::cout << "No input frame" << std::endl;
            continue;
        }

        std::cout << "New frame " << frame.frameId <<
        " (" << frame.width << "x" << frame.height << ") cycle time" <<
        (int)source->getParam(cr::video::VSourceParam::CYCLE_TIME_MKS) <<
        " mksec" << std::endl;
    }

    return 1;
}
```